

DATABASE SYSTEMS

**DESIGN, IMPLEMENTATION,
AND MANAGEMENT**

FIFTH EDITION

FIFTH EDITION

DATABASE SYSTEMS

DESIGN, IMPLEMENTATION, AND MANAGEMENT

PETER ROB CARLOS CORONEL

**COURSE
TECHNOLOGY**



THOMSON LEARNING

Australia • Canada • Mexico • Singapore • Spain • United Kingdom • United States

**Питер Роб
Карлос Коронел**

Системы баз данных: проектирование, реализация и управление

5-е издание

Санкт-Петербург
«БХВ-Петербург»
2004

УДК 681.3.06
ББК 32.973.26-018
P58

Роб П., Коронел К.

P58 Системы баз данных: проектирование, реализация и управление. — 5-е изд., перераб. и доп.: Пер. с англ. — СПб.: БХВ-Петербург, 2004. — 1040 с.: ил.

ISBN 5-94157-299-9

Книга содержит последовательное и всестороннее описание процесса разработки и реализации базы данных. Подробно представлены теоретические основы баз данных: реляционная модель базы данных, ER-моделирование, нормализация таблиц базы данных, язык структурированных запросов (SQL). Приведен пример проектирования и реализации базы данных для университетской лаборатории. Рассматриваются управление транзакциями и параллельным выполнением, системы управления распределенными базами данных, объектно-ориентированные базы данных, клиент/серверные системы, информационное хранилище, электронная коммерция, особенности разработки баз данных для Web (с помощью ColdFusion 4.0), хранилища данных и администрирование баз данных на примере СУБД Oracle.

Для студентов и преподавателей

УДК 681.3.06
ББК 32.973.26-018

Группа подготовки издания:

Главный редактор	<i>Екатерина Кондукова</i>
Зав. редакцией	<i>Григорий Добин</i>
Перевод с английского	<i>Андрея Никифорова</i>
Редактор	<i>Татьяна Темкина</i>
Компьютерная верстка	<i>Натальи Смирновой</i>
Корректор	<i>Наталья Першакова</i>
Дизайн обложки	<i>Игоря Цырульников</i>
Зав. производством	<i>Николай Тверских</i>

ALL RIGHTS RESERVED. No part of this work covered by the copyright hereon may be reproduced or used in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, Web distribution, or information storage and retrieval systems—without the written permission of the publisher. Authorized translation by BHV-Petersburg, 2004

Все права защищены. Никакая часть настоящей книги не может быть воспроизведена или передана в какой бы то ни было форме и какими бы то ни было средствами, будь то графические, электронные или механические, включая фотокопирование и запись на магнитный носитель, а также через Интернет и другие информационные системы, если на то нет письменного разрешения издательства. Авторизованный перевод "БХВ-Петербург", 2004.

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 02.12.03

Формат 70×100¹/₁₆. Печать офсетная. Усл. печ. л. 83,85.

Тираж 4000 экз. Заказ № 451

"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02 953.Д.001537 03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов

в ФГУП ордена Трудового Красного Знамени "Техническая книга"
Министерства Российской Федерации по делам печати,
телерадиовещания и средств массовых коммуникаций.
198005, Санкт-Петербург, Измайловский пр., 29.

ISBN 0-619-06269-X (англ.)

ISBN 5-94157-299-9 (рус.)

© 2002 Course Technology, a division of Thomson Learning, Inc
Thomson LearningTM is a trademark used herein under license

© Перевод на русский язык "БХВ-Петербург", 2004

Содержание

Посвящение	1
Предисловие	3
Изменения в пятом издании.....	3
Наш подход: акцент на проектировании.....	4
Охватываемая тематика.....	6
Обучение базам данных: все зависит от цели.....	7
Дополнительные возможности.....	9
1. Базы данных, имеющиеся в книге.....	9
2. Подробное руководство для преподавателей.....	10
3. Новые задачи.....	10
4. Информационные хранилища и анализ данных.....	10
5. Все имеющиеся проекты готовы к реализации.....	11
6. Наиболее употребительные графические модели.....	11
7. Самодокументированное соглашение об именах.....	12
8. Таблицы, иллюстрирующие сложные проблемы проектирования.....	12
9. Контрольные вопросы.....	12
Благодарности.....	12
Часть I. Основы баз данных	17
Глава 1. Системы файлов и базы данных	19
Обзор.....	19
1.1. Введение в базы данных.....	20
1.1.1. Почему так важно проектирование базы данных.....	24
1.1.2. Практический подход к проектированию базы данных.....	25
1.2. Предыстория баз данных: файлы и системы файлов.....	25
1.3. Оценка системы файлов.....	30
1.3.1. Управление системой файлов.....	31
1.3.2. Структурная зависимость и зависимость по данным.....	33
1.3.3. Определение полей и соглашение об именах.....	33
1.3.4. Избыточность данных.....	35
1.4. Системы баз данных.....	36
1.4.1. Среда системы базы данных.....	37
1.4.2. Типы систем управления базами данных.....	40
1.4.3. Предназначение СУБД.....	42
1.4.4. Управление системой базы данных: смещение акцентов.....	44
1.4.5. Проектирование базы данных и моделирование.....	45

1.5. Модели баз данных	45
1.5.1. Иерархическая модель	46
1.5.2. Сетевая модель.....	52
1.5.3. Реляционная модель.....	56
1.5.4. Модель "сущность-связь".....	61
1.5.5. Объектно-ориентированная модель	66
1.6. Эволюция моделей данных	70
1.6.1. Модели баз данных и Интернет	72
Резюме	73
Основные термины	79
Вопросы.....	83
Задачи	84
Глава 2. Реляционная модель базы данных	90
Обзор	90
2.1. Логическое представление данных.....	91
2.1.1. Сущности и атрибуты	91
2.1.2. Таблицы и их свойства	92
2.2. Ключи.....	96
2.3. Еще раз о правилах целостности.....	102
2.4. Реляционные операторы	104
2.5. Словарь данных и системный каталог.....	112
2.6. Связи в реляционной базе данных.....	115
2.7. Еще раз об избыточности данных.....	125
2.8. Индексы	128
Резюме	129
Основные термины	131
Вопросы.....	132
Задачи	135
Часть II. Основы проектирования и реализации	147
Глава 3. ER-моделирование.....	149
Обзор	149
3.1. Основы моделирования.....	150
3.2. Модели данных: уровни абстракции данных.....	152
3.2.1. Концептуальная модель	153
3.2.2. Внутренняя модель.....	156
3.2.3. Внешняя модель	157
3.2.4. Физическая модель.....	160
3.3. Модель "сущность-связь" (ER-модель).....	160
3.3.1. Сущности.....	161
3.3.2. Атрибуты.....	161
3.3.3. Связи.....	167
3.3.4. Связность и мощность связи	167

3.3.5. Сила связей	169
3.3.6. Участие в связи	173
3.3.7. Сила связи и слабые сущности	177
3.3.8. Степень связи	179
3.3.9. Составные сущности	185
3.3.10. Супертипы и подтипы сущности	188
3.4. Сравнение обозначений в ER-моделировании	191
3.5. Разработка ER-диаграмм	195
3.6. Проблемы проектирования базы данных: противоречивые цели	203
Резюме	206
Основные термины	207
Вопросы	209
Задачи	211
Глава 4. Нормализация таблиц базы данных	226
Обзор	226
4.1. Таблицы базы данных и нормализация	227
4.1.1. Необходимость нормализации	228
4.1.2. Приведение к первой нормальной форме	232
4.1.3. Приведение ко второй нормальной форме	235
4.1.4. Преиведение к третьей нормальной форме	237
4.1.5. Нормальная форма Бойса—Кодда (БКНФ)	243
4.2. Нормализация и проектирование базы данных	246
4.3. Нормальные формы более высокого уровня	251
4.4. Денормализация	253
Резюме	255
Основные термины	258
Вопросы	259
Задачи	260
Глава 5. Язык структурированных запросов (SQL)	270
Обзор	270
5.1. Введение в SQL	271
5.2. Команды описания данных	272
5.2.1. Модель базы данных	272
5.2.2. Таблицы и их компоненты	273
5.2.3. Создание базы данных и табличных структур	274
5.2.4. Создание структуры таблиц	275
5.2.5. Домены	283
5.2.6. Ограничения целостности SQL	285
5.3. Команды манипулирования данными	286
5.3.1. Ввод данных	286
5.3.2. Сохранение содержимого таблицы	289
5.3.3. Распечатка содержимого таблицы	289
5.3.4. Изменение данных	290
5.3.5. Восстановление содержимого таблиц	291
5.3.6. Удаление строк таблицы	291

5.4. Запросы	292
5.4.1. Распечатка части содержимого таблицы	292
5.4.2. Логические операторы: AND, OR и NOT	299
5.4.3. Специальные операторы	301
5.5. Дополнительные команды управления данными	304
5.5.1. Изменение типа данных столбца	305
5.5.2. Изменение свойств атрибута	305
5.5.3. Добавление столбцов	306
5.5.4. Удаление столбцов	306
5.5.5. Занесение данных в новый столбец	306
5.5.6. Арифметические операторы и правило старшинства	309
5.5.7. Копирование части таблицы	310
5.5.8. Удаление таблицы из базы данных	312
5.5.9. Обозначение первичных и внешних ключей	312
5.6. Более сложные запросы и функции SQL	313
5.6.1. Упорядочивание списка	313
5.6.2. Вывод уникальных значений	316
5.6.3. Агрегатные функции SQL	317
5.6.4. Группирование данных	321
5.6.5. Виртуальные таблицы: создание представления	325
5.6.6. Индексы в SQL	326
5.6.7. Объединение таблиц базы данных	327
5.7. Обновляемые представления	333
5.8. Процедурный язык SQL	336
5.8.1. Триггеры	338
5.8.2. Хранимые процедуры	345
5.8.3. Хранимые функции PL/SQL	348
5.9. Конвертирование ER-модели в структуру базы данных	348
5.10. Общие правила, обеспечивающие связь таблиц в базе данных	354
Резюме	367
Основные термины	373
Вопросы	374
Задачи	378

Часть III. Эффективное проектирование и реализация баз данных.....391

Глава 6. Проект базы данных.....393

Обзор	393
6.1. Структурирование данных	394
6.2. Информационная система	395
6.3. Жизненный цикл разработки систем (SDLC)	397
6.3.1. Планирование	398
6.3.2. Анализ	399
6.3.3. Детальное проектирование систем	400

6.3.4. Реализация.....	400
6.3.5. Сопровождение.....	401
6.4. Жизненный цикл базы данных (DBLC).....	402
6.4.1. Этап начальной разработки.....	403
6.4.2. Проектирование базы данных.....	407
6.4.3. Реализация и загрузка данных.....	425
6.4.4. Тестирование и оценка.....	431
6.4.5. Функционирование.....	431
6.4.6. Сопровождение и развитие.....	432
6.5. О стратегии проектирования базы данных.....	432
6.6. Централизованное и децентрализованное проектирование.....	433
Резюме.....	437
Основные термины.....	438
Вопросы.....	439
Задачи.....	439
Глава 7. Университетская лаборатория: концептуальный проект	442
Обзор.....	442
7.1. Этап начальной разработки базы данных.....	444
7.1.1. Задачи UCL.....	445
7.1.2. Организационная структура.....	445
7.1.3. Описание операций.....	447
7.1.4. Проблемы и ограничения.....	452
7.1.5. Предназначение системы.....	455
7.1.6. Сфера действия и границы.....	456
7.2. Этап проектирования базы данных: концептуальный проект.....	459
7.2.1. Источники информации и пользователи.....	459
7.2.2. Необходимая информация: требования пользователя.....	462
7.2.3. Разработка первоначальной ER-модели.....	465
Резюме.....	479
Основные термины.....	480
Вопросы.....	480
Задачи.....	481
Глава 8. Университетская лаборатория: тестирование концептуального проекта, логическое проектирование и реализация	488
Обзор.....	488
8.1. Завершение концептуального и логического проектирования базы данных.....	489
8.2. Завершение концептуального проекта: сущности, атрибуты и нормализация.....	491
8.2.1. Модуль "Система Управления Лабораторией".....	492
8.2.2. Модуль "Система Управления Материально-Техническим Снабжением".....	508

8.3. Проверка ER-модели	529
8.4. Логическое проектирование.....	537
8.4.1. Таблицы	537
8.4.2. Индексы и представления	539
8.5. Физическое проектирование.....	540
8.6. Реализация	542
8.6.1. Создание базы данных	545
8.6.2. Загрузка базы данных и конвертирование.....	545
8.6.3. Системные процедуры	546
8.7. Тестирование и оценка	546
8.7.1. Оценка производительности	546
8.7.2. Меры по обеспечению безопасности	547
8.7.3. Процедуры резервного копирования и восстановления	548
8.8. Работа с базой данных	548
8.8.1. Работоспособная база данных	548
8.8.2. Рабочие процедуры	549
8.8.3. Управление базой данных: обслуживание и развитие.....	549
Резюме	549
Основные термины	551
Вопросы.....	551
Задачи	553

Часть IV. РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ КОНЦЕПЦИЙ БАЗ ДАННЫХ555

Глава 9. Управление транзакциями и параллельным выполнением557

Обзор	557
9.1. Что такое транзакция?	558
9.1.1. Оценка результатов транзакции.....	559
9.1.2. Свойства транзакции.....	561
9.1.3. Управление транзакциями с помощью SQL.....	562
9.1.4. Журнал транзакций	563
9.2. Управление параллельным выполнением транзакций	564
9.2.1. Потеря изменений.....	565
9.2.2. Несвязность данных	566
9.2.3. Неоднозначность поиска	567
9.2.4. Планировщик.....	569
9.3. Управление параллельным выполнением транзакций методом блокировки	570
9.3.1. Степень детализации блокировок	571
9.3.2. Типы блокировок	574
9.3.3. Обеспечение сериализуемости при помощи двухфазной блокировки.....	576
9.3.4. Тупики	578
9.4. Управление параллельным выполнением транзакций при помощи меток времени.....	579

9.5. Управление параллельным выполнением транзакций при помощи оптимистических методов	580
9.6. Управление восстановлением базы данных	580
9.6.1. Восстановление транзакции	582
Резюме	584
Основные термины	585
Вопросы	587
Задачи	587
Глава 10. Системы управления распределенными базами данных.....	589
Обзор	589
10.1. Этапы развития систем управления распределенными базами данных	590
10.1.1. Преимущества СУРБД	592
10.1.2. Недостатки СУРБД.....	593
10.2. Распределенная обработка данных и распределенные базы данных.....	594
10.3. Что такое система управления распределенной базой данных?	596
10.4. Компоненты СУРБД.....	599
10.5. Уровни распределения данных и их обработки.....	600
10.5.1. Обработка и размещение данных на одном сайте	601
10.5.2. Обработка данных на нескольких сайтах, размещение данных на одном сайте (MPSD).....	601
10.5.3. Обработка и размещение данных на нескольких сайтах (MPMD)	603
10.6. Прозрачные свойства распределенной базы данных	605
10.7. Прозрачность распределения.....	606
10.8. Прозрачность транзакций	609
10.8.1. Распределенные запросы и распределенные транзакции	609
10.8.2. Управление параллельным выполнением в распределенной среде	613
10.8.3. Протокол двухфазного подтверждения транзакции	614
10.9. Прозрачность производительности и оптимизация запроса	616
10.10. Проектирование распределенной базы данных	618
10.11. Фрагментация данных.....	618
10.11.1. Горизонтальная фрагментация.....	619
10.11.2. Вертикальная фрагментация.....	620
10.11.3. Смешанная фрагментация	621
10.12. Репликация данных.....	623
10.13. Распределение данных	625
10.14. Сравнение СУРБД и архитектуры "клиент/сервер".....	626
10.15. Двенадцать правил Дейта для распределенных баз данных	627
Резюме	628
Основные термины	629
Вопросы.....	631
Задачи	632

Часть V. НОВЫЕ РАЗРАБОТКИ	635
Глава 11. Объектно-ориентированные базы данных.....	637
Обзор	637
11.1. Преимущества объектно-ориентированного подхода	638
11.2. История развития объектно-ориентированного подхода	639
11.3. Общие принципы объектно-ориентированного подхода	640
11.3.1. Объекты: компоненты и свойства	640
11.3.2. Идентификация объекта	641
11.3.3. Атрибуты (переменные экземпляра)	641
11.3.4. Состояние объекта.....	643
11.3.5. Сообщения и методы	643
11.3.6. Классы	646
11.3.7. Протокол	647
11.3.8. Суперклассы, подклассы и наследование.....	648
11.3.9. Переопределение методов и полиморфизм	652
11.3.10. Абстрактные типы данных.....	654
11.3.11. Классификация объектов.....	655
11.4. Свойства объектно-ориентированной модели данных	656
11.4.1. Схемы объектов: графическое представление объектов.....	657
11.4.2. Связи "класс-подкласс".....	660
11.4.3. Межобъектная связь "атрибут-класс".....	662
11.4.4. Позднее и раннее связывание: предназначение и использование.....	669
11.4.5. Поддержка контроля версий	671
11.5. ООМД и прежние модели данных: сходство и различие	671
11.5.1. Объект, сущность и кортеж.....	671
11.5.2. Класс, набор сущностей и таблица	672
11.5.3. Инкапсуляция и наследование.....	673
11.5.4. Идентификатор объекта (OID)	673
11.5.5. Связи.....	673
11.5.6. Доступ	674
11.6. Объектно-ориентированная система управления базой данных (ООСУБД).....	675
11.6.1. Возможности ООСУБД.....	677
11.7. Влияние объектно-ориентированного подхода на проектирование баз данных.....	679
11.8. ООСУБД: преимущества и недостатки.....	682
11.9. Влияние объектно-ориентированных представлений на реляционную модель	684
11.10. Новое поколение систем управления базой данных	686
Резюме.....	686
Основные термины	687
Вопросы.....	689
Задачи	690

Глава 12. Клиент/серверные системы.....	695
Обзор	695
12.1. Что такое клиент/серверные вычисления?.....	695
12.2. Причины использования клиент/серверных систем.....	698
12.3. История развития клиент/серверных информационных систем.....	698
12.4. Что ожидает руководство от клиент/серверных систем.....	701
12.4.1. Что ожидает от клиент/серверных вычислений отдел MIS.....	702
12.4.2. Что ожидает от клиент/серверных вычислений предприятие.....	702
12.5. Архитектура "клиент/сервер"	702
12.5.1. Как взаимодействуют компоненты.....	703
12.5.2. Правила архитектуры "клиент/сервер".....	705
12.5.3. Компоненты клиента	707
12.5.4. Компоненты сервера.....	709
12.5.5. Компоненты коммуникационного промежуточного программного обеспечения (ППО)	712
12.5.6. Сетевые протоколы	718
12.5.7. Компоненты ППО базы данных.....	719
12.5.8. Классификация ППО.....	724
12.6. В поисках стандартов.....	724
12.7. Клиент/серверные базы данных	727
12.8. Стили архитектуры "клиент/сервер"	728
12.9. Проблемы реализации клиент/серверных систем	734
12.9.1. Клиент/серверные системы и традиционные системы обработки данных.....	735
12.9.2. Администрирование	737
12.9.3. Инструментарий разработки клиент/серверных приложений	739
12.9.4. Интегрированный подход.....	740
Резюме	741
Основные термины	742
Вопросы.....	744
Глава 13. Информационное хранилище	745
Обзор	745
13.1. Необходимость анализа информации	746
13.2. Системы поддержки решений	748
13.2.1. Операционные данные и данные системы поддержки решений.....	751
13.2.2. Требования к базе данных DSS.....	755
13.3. Хранилище данных	760
13.3.1. Стили архитектуры DSS.....	765
13.3.2. Двенадцать правил для хранилища данных.....	767
13.4. Оперативный анализ данных (OLAP).....	768
13.4.1. Архитектура OLAP.....	772
13.4.2. Реляционные OLAP-системы.....	776
13.4.3. Многомерные OLAP-системы.....	780
13.4.4. Сравнение реляционных и многомерных систем OLAP.....	782

13.5. Схема "звезда"	784
13.5.1. Факты.....	784
13.5.2. Качественные характеристики (измерения)	784
13.5.3. Атрибуты.....	785
13.5.4. Иерархии атрибутов	788
13.5.5. Представление в виде схемы "звезда"	790
13.5.6. Технические приемы повышения производительности	793
13.6. Реализация хранилища данных	796
13.6.1. Хранилище данных как активная среда поддержки решений.....	797
13.6.2. Усилия всего предприятия с привлечением пользователей и согласованием на всех уровнях	797
13.6.3. Данные, анализ и пользователи	798
13.6.4. Использование процедур проектирования баз данных	798
13.7. Добыча данных (data mining)	799
Резюме	804
Основные термины	806
Вопросы.....	807
Задачи	809
Глава 14. Базы данных в электронной коммерции	814
Обзор	814
14.1. Что такое электронная коммерция?.....	815
14.2. Путь к электронной коммерции.....	816
14.3. Влияние е-коммерции	817
14.3.1. Преимущества е-коммерции	817
14.3.2. Недостатки е-коммерции.....	818
14.4. Стили е-коммерции	818
14.5. Архитектура е-коммерции	819
14.5.1. Основные службы Интернета.....	821
14.5.2. Службы обеспечения бизнеса	825
14.5.3. Службы обеспечения е-коммерции	827
14.6. Безопасность	828
14.7. Обработка платежей.....	830
14.7.1. Электронные деньги.....	831
14.7.2. Обработка кредитных карт	832
14.7.3. Электронный бумажник	833
14.8. Проектирование баз данных для приложений е-коммерции	833
14.9. Расширяемый язык разметки (XML)	847
14.9.1. Определение типа документа (DTD) и XML-схемы	849
14.9.2. XML-представление	853
14.9.3. XML-приложения	856
Резюме	858
Основные термины	859
Вопросы.....	861
Задачи	861

Глава 15. Разработка баз данных для Web	863
Обзор	863
15.1. Интернет-технологии и базы данных	863
15.2. Типичные проблемы баз данных Интернета	865
15.3. Промежуточное программное обеспечение баз данных Web: серверные расширения	866
15.3.1. Интерфейсы Web-сервера	868
15.3.2. Открытый интерфейс доступа к базам данных (ODBC)	870
15.4. Web-обозреватель	872
15.4.1. Клиентские расширения	873
15.5. Инструменты Web-БД: ColdFusion	874
15.5.1. Как работает ColdFusion	876
15.5.2. Демонстрационная база данных компании RobCor	877
15.5.3. Создание простого запроса с помощью тегов <CFQUERY> и <CFOUTPUT>	878
15.5.4. Создание простого запроса с помощью тегов <CFQUERY> и <CFTABLE>	883
15.5.5. Создание страниц динамического поиска	886
15.5.6. Web как система без сохранения состояния	891
15.5.7. Ввод данных	893
15.5.8. Обновление данных	899
15.5.9. Удаление данных	906
15.6. Системы баз данных Интернета: замечания	914
15.6.1. Какие типы данных поддерживаются?	915
15.6.2. Безопасность данных	916
15.6.3. Управление транзакциями	917
15.6.4. Денормализация таблиц базы данных	917
Резюме	918
Основные термины	919
Вопросы	920
Задачи	920

Часть VI. АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

923

Глава 16. Администрирование баз данных

925

Обзор	925
16.1. Информация как достояние корпорации	926
16.2. Роль базы данных на предприятии	927
16.3. Внедрение базы данных: анализ проблем	929
16.4. Эволюция функций администратора базы данных	931
16.5. Человеческий фактор в среде базы данных	935
16.5.1. Организационные функции DBA	938
16.5.2. Технические функции DBA	948
16.6. Инструментальные средства администрирования баз данных	956
16.6.1. Словарь данных	956
16.6.2. Инструментальные средства CASE	959

16.7. Разработка стратегии администрирования данных	962
16.8. Работа DBA: использование СУБД Oracle для администрирования базы данных	964
16.8.1. Инструментальные средства администрирования Oracle	965
16.8.2. Регистрация по умолчанию	966
16.8.3. Автоматический запуск РСУБД	967
16.8.4. Использование менеджера хранения для создания пространства таблиц и файлов данных	968
16.8.5. Управление объектами базы данных: таблицы, представления, триггеры и процедуры	971
16.8.6. Управление пользователями и безопасность	973
16.8.7. Настройка начальных параметров базы данных	975
16.8.8. Создание новой базы данных	976
Резюме	979
Основные термины	981
Вопросы	983
Приложение 1. Клиент/серверная сетевая инфраструктура	985
Обзор	985
Сетевые кабели	985
Топология сети	986
Типы сетей	988
Сетевые коммуникационные устройства	989
Глоссарий	991
Предметный указатель	1017

Посвящение

Посвящается Анне (Ann), которая после сорока лет совместной жизни остается моим лучшим другом. Нашему сыну, Питеру Уильяму (Peter William), проявившему настоящую мудрость, приведя в нашу семью любимую невестку Шину (Sheena). Шине, которую мы очень любим на протяжении многих лет. Нашим внукам Адаму Ли (Adam Lee) и Алану Генри (Alan Henry), которые растут, чтобы стать такими же хорошими людьми, как и их родители. Моим тестю и теще Генри (Henry) и Нини Фонтейн (Nini Fontein), получившим столь большой жизненный опыт в Европе и Юго-Восточной Азии, достойный написания целой исторической повести, которые верили в меня и в будущее своей дочери, и являются для нас самым сокровенным в жизни. Памяти моих родителей Хендрика и Эрмины Роб (Hendrick and Hermina Rob), сумевших заново построить свою жизнь после ужасов Второй мировой войны, которые смогли это сделать и во второй раз, после неудавшегося мятежа в Индонезии, и, в конце концов, нашедших землю обетованную здесь, в Соединенных Штатах Америки. И памяти Хейнца (Heinz), который всегда преподносил мне уроки лояльности, внимательного отношения и безграничного понимания. Всем им я с любовью посвящаю эту книгу.

Питер Роб (Peter Rob)

Моим родителям за мое воспитание. Моей красавице жене Виктории (Victoria), не позволявшей мне впадать в уныние во время бесконечной работы над книгой. Я очень благодарен ей за заботу и поддержку в самые трудные моменты. Карлосу Энтони (Carlos Anthony), моему сыну, который является гордостью отца и постоянно учит меня чему-то новому. Габриэле Виктории (Gabriela Victoria), моей дочери и принцессе нашего дома. Кристиану Хавьеру (Cristian Javier), нашему маленькому жизнерадостному сверточку. Моим детям с благодарностью за их смех, их любимые голоса, их прекрасные улыбки и их частые объятия. Я люблю вас; вы мое бесценное сокровище.

Карлос Коронел (Carlos Coronel)

Предисловие

Данное пятое¹ издание продолжает тему формирования основ проектирования, внедрения и управления систем баз данных (БД). Главная идея, положенная нами в этот фундамент, состоит в том, что успешное проектирование хороших баз данных, без сомнения, очень полезных, зависит от понимания их важнейших основополагающих концепций. Всегда очень нелегко разумно сочетать в учебном пособии теорию и практику, и мы очень благодарны за дельные советы предыдущим редакторам и нашим студентам. Многие рецензенты пятого издания нашей книги считают, что мы достаточно преуспели в поисках такого баланса.

Изменения в пятом издании

В пятом издании мы продолжаем детально изучать проектирование баз данных. Однако меняющаяся среда баз данных, большое число предложений от редакторов четвертого издания и собственный накопленный опыт в преподавании и практической деятельности послужили поводом к ряду изменений, которые ниже приводятся в порядке возрастания их важности. В этом издании уделено внимание влиянию, которое оказывает Интернет на процесс проектирования, внедрения и использования баз данных. Кроме того, из практических соображений мы достаточно подробно рассмотрели модель "птичья лапка", широко используемую при проектировании баз данных. Многие проблемы проектирования баз данных, обсуждение которых начато в четвертом издании, здесь рассмотрены более детально, а некоторые вновь поставленные нами проблемы приоткрывают окно в сегодняшний мир проектирования. Все главы в настоящем издании были обновлены, некоторые из них подверглись полной переработке. Также была добавлена новая глава, в которой обсуждаются проблемы электронного обращения денежных средств.

- Глава 5 "*Язык структурированных запросов (SQL)*" значительно расширена. В четвертом издании в нее было добавлено обсуждение триггеров и хранимых процедур с указанием того, что эти возможности делают язык SQL еще более полезным. В пятом издании добавлен раздел, где описаны основные правила, влияющие на управление отношениями между таблицами. Был также добавлен раздел, в котором исследуется, каким образом модель отношения "сущность-связь" конвертируется в структуру баз данных. Однако мы убедились, что это дополнение не очень заинтересовало наших читателей, которые вообще предпочитают пропустить эту главу с тем, чтобы сосредоточиться на других аспектах проектирования, реализации и управления базами данных.
- В главе 9 "*Управление транзакциями и параллельным выполнением*" сделано несколько важных изменений. Сюда относятся и обсуждение использования цвета для удобства отслеживания транзакций, и новый раздел, посвященный восстановлению транзакций.

¹ Речь идет об оригинальном издании книги. — Прим. ред.

- Добавлена новая глава 14 *"Базы данных в электронной коммерции"*. Все организации, от учебного института до коммерческих предприятий любого размера и государственных учреждений всех уровней пользуются услугами *e*-коммерции для представления своих товаров и услуг на рынке. Поэтому будет правильным исследовать компоненты инфраструктуры транзакций и проанализировать влияние требований к транзакциям *e*-коммерции на проектирование рабочих (операционных) баз данных. Это дополнение имеет большое значение для более полного описания баз данных.
- Глава 14 четвертого издания в пятом издании стала главой 15 *"Разработка баз данных для Web"*, новое название точнее отражает ее содержимое. В этой главе исследуется влияние Интернета на проектирование и внедрение баз данных. В продолжение наших традиций практического подхода к проектированию баз данных в главе 15 будут продемонстрированы возможности простого в изучении инструмента разработки интернет-приложений ColdFusion.
- Хотя глава 15 четвертого издания в этом издании является главой 16 *"Администрирование баз данных"*, ее практическое предназначение отражает разд. 16.8 *"Работа DBA: использование Oracle для администрирования базы данных"*. В этом разделе представлены доступные администратору баз данных инструменты, обсуждаются процедура регистрации, создание структур БД — области таблиц и файлов данных, управление объектами базы данных, обеспечение безопасности, настройка начальных параметров БД и как собственно создается база данных.
- Поскольку материал, представленный в книге, постоянно расширяется, все новейшие сведения, касающиеся некоторых материалов книги, размещены на Web-сайте Course Technology (подразделение корпорации Thomson Learning Inc.) www.course.com (ищите эту книгу по индексу ISBN: 0-619-06269-X).

Наш подход: акцент на проектировании

На пятом издании отразились комментарии и предложения читателей четвертого издания и рецензентов, проверяющих наши исправления и дополнения. Кроме того, очень важными рецензентами для нас были студенты, обратная связь с которыми поддерживалась при оценке материала книги на практических занятиях. Как преподаватели, мы продолжали открывать лучшие способы обоснования тех или иных положений, и обнаружили, что несмотря на тщательное изучение и редактирование, предшествовавшее четвертому изданию, все же остались некоторые ошибки, упущения и недочеты, которые необходимо было исправить. Развитие технологии баз данных потребовало обсуждения новых тем и нового подхода к изложению "старых" тем. Наконец, используя практический опыт применения баз данных в реальной жизни, мы нашли лучшие методы разработки и реализации некоторых проектов и лучшее объяснение методов, применяющихся для других проектов. Короче говоря, именно практика обусловила существенные изменения, нашедшие отражение в новом издании.

Как следует из названия — *"Системы баз данных: проектирование, реализация и управление"* — в книге обсуждаются три основных аспекта, касающихся систем баз

данных. Авторы уверены, что практический подход к проектированию, разработке и реализации баз данных заслуживает особого внимания по следующим причинам.

- Доступность превосходного программного обеспечения баз данных дает возможность даже новичкам создавать базы данных и приложения баз данных. К сожалению, подход "разработка без проекта" становится причиной множества провалов. Наша практика показывает, что многие, если не все, ошибки в плохо спроектированных системах баз данных не могут устранить даже лучшие программисты и менеджеры. При этом даже превосходная система управления базой данных (СУБД), вероятнее всего, не поможет разрешить проблемы, порожденные или усиленные плохим проектированием. Можно провести аналогию со строительством: даже лучшие каменщики и плотники не смогут построить хорошее здание по плохому проекту.
- Большинство проблем в системах управления базами данных чаще всего возникает из-за плохого проектирования. Едва ли стоит использовать дефицитные ресурсы для совершенствования навыков работы с превосходной и большой системой управления базы данных для того, чтобы упражняться в разрешении проблем, возникших из-за плохо спроектированной базы данных.
- Проектирование также представляет собой хорошее средство получения дополнительной информации. Клиенты, скорее всего, получают то, что им необходимо, в том случае, если к проектированию базы данных подходить со всей тщательностью и вниманием. Кроме того, хороший проект базы данных позволит клиентам по-новому взглянуть на работу своего предприятия.
- Близкое знакомство с техникой проектирования базы данных поможет лучше понять современные технологии БД. Например, поскольку хранилища данных (data warehouse) получают информацию от рабочих (операционных) баз данных, концепции, структура и процедуры хранилищ данных станут более обоснованными, если вы будете достаточно хорошо разбираться в структуре и реализации рабочих БД.

Иными словами, проектирование баз данных, несмотря на недостаток точных теоретических формулировок, имеет чрезвычайно важное практическое значение, чем и объясняется то внимание, которое мы уделяем ему в этой книге.

Поскольку основной предмет книги — практические аспекты разработки баз данных, мы очень подробно рассматриваем процедуры и понятия, связанные с проектированием; проблемы, поставленные в заключительных главах книги, мы уверены, станут стимулом к развитию реальных и полезных навыков проектирования современных баз данных. Мы также уверены, что студенты понимают потенциальный и реальный конфликт между внешней элегантностью проекта БД и требованиям к информации и скорости выполнения транзакций. Например, нет смысла проектировать базы данных с очень красивым пользовательским интерфейсом, в то же время не удовлетворяющие требованиям конечных пользователей к обработке информации. Необходимо стараться найти компромисс, при котором удовлетворяются все требования конечных пользователей и, в то же время, обеспечивается высококачественный дизайн проекта.

Охватываемая тематика

Название нашей книги начинается со слов "*Системы баз данных*". Поэтому с *гл. 1* по *гл. 5* мы исследуем базы данных и основные понятия проектирования как часть рабочей среды, системный анализ которой приведен в *гл. 6*. Мы полагаем, что проектировщики баз данных, не понимающие, что база данных является частью большой системы, вероятнее всего не заметят важнейшие требования к проекту БД. Фактически в *гл. 6* представлена схема расширенного проектирования БД, обсуждение которой будет продолжено в *гл. 7* и *8*. В рамках рабочей среды большой системы мы можем исследовать такие проблемы, как управление транзакциями и контроль совпадений (Transaction Management and Concurrency Control, *гл. 9*), системы управления распределенными базами данных (Distributed Database Management Systems, *гл. 10*), информационные хранилища (Data Warehouse, *гл. 13*), базы данных в электронной коммерции (Databases in Electronic Commerce, *гл. 14*) и администрирование баз данных (Database Administration, *гл. 16*).

Первым в подзаголовке названия книги указано *проектирование* (design), и изучение проектирования баз данных в книге является поистине всесторонним. Например, в *гл. 1* сформулирована необходимость проектирования, в *гл. 2* закладываются основы проектирования реляционной модели, в *гл. 3* приводится подробное описание практики проектирования баз данных, а *гл. 4* целиком посвящена проблемам нормализации, влияющим на производительность баз данных. В *гл. 5* рассматривается процесс проектирования и реализация баз данных. В *гл. 6* исследуется проектирование БД в системной рабочей среде, а действия, необходимые для успешного проектирования и внедрения реальных сложных баз данных, обсуждаются в *гл. 7* и *8*. Такая предварительная подготовка позволит студенту понять все тонкости разработки проектирования информационных хранилищ, которому посвящена *гл. 13*.

Поскольку модель базы данных оказывает существенное влияние на проектирование БД, мы досконально исследуем основные модели баз данных. Например, обсуждение преобладающей на рынке реляционной модели начинается в конце *гл. 1* и продолжается в *гл. 2*. *Гл. 3* и *4* завершают обсуждение проектирования и методов контроля качества, играющих важнейшую роль в успешном проектировании баз данных. Язык SQL, обсуждаемый в *гл. 5*, демонстрирует мощь использования запросов в реляционных базах данных. Объектно-ориентированным базам данных посвящена *гл. 11*, а клиент-серверные системы рассмотрены в *гл. 12*. В *гл. 13* подробно анализируются информационные хранилища и исследуется их влияние на проектирование БД. В *гл. 14* раскрываются основы проектирования баз данных для электронной коммерции.

На втором месте в подзаголовке книги указана *реализация* (implementation). Поскольку в *гл. 7* и *8* приводятся полностью реализуемые методы проектирования БД, в этих главах обсуждается широкий спектр практических проблем внедрения. Естественно, зачастую мы встречаем взаимоисключающие цели проектирования: элегантность дизайна, требования к информативности и скорости обработки данных. Поэтому мы тщательно перепроверили изначальный проект *гл. 7* на предмет удовлетворения потребностям конечных пользователей и выработки надлежащих соглашений реализации. Результатом этой проверки стал окончательный дизайн базы дан-

ных, разработанный в *гл. 8*. (На основе этой рабочей базы данных рассматриваются проблемы информационных хранилищ в *гл. 13*.) Особые проблемы проектирования баз данных в среде Интернет рассмотрены в *гл. 14* и *15*.

Заключительная часть подзаголовка книги — *управление* (management). Проблемам управления посвящены *гл. 9, 10* и *16*.

Обучение базам данных: все зависит от цели

При столь широкой тематике преподаватели могут "смешивать и подстраивать" главы книги в своих целях. В зависимости от того, как курс "Базы данных" вписывается в учебный план, преподаватели могут уделить основное внимание проектированию, современным технологиям БД или проблемам управления базами данных. Рис. П.1 можно использовать как основу для разработки программы курсов. Хотя во всех трех курсах используются основные концепции разработки баз данных, в курсе проектирования (см. *гл. 6—8*) создается, проверяется и реализуется детальный и сложный проект БД. Обсуждение этой темы дополняется исследованиями в *гл. 13* звездобразных схем (star schemas) и фактическим исследованием в *гл. 14* и *15* влияния Интернета на проектирование, разработку, использование и управление.

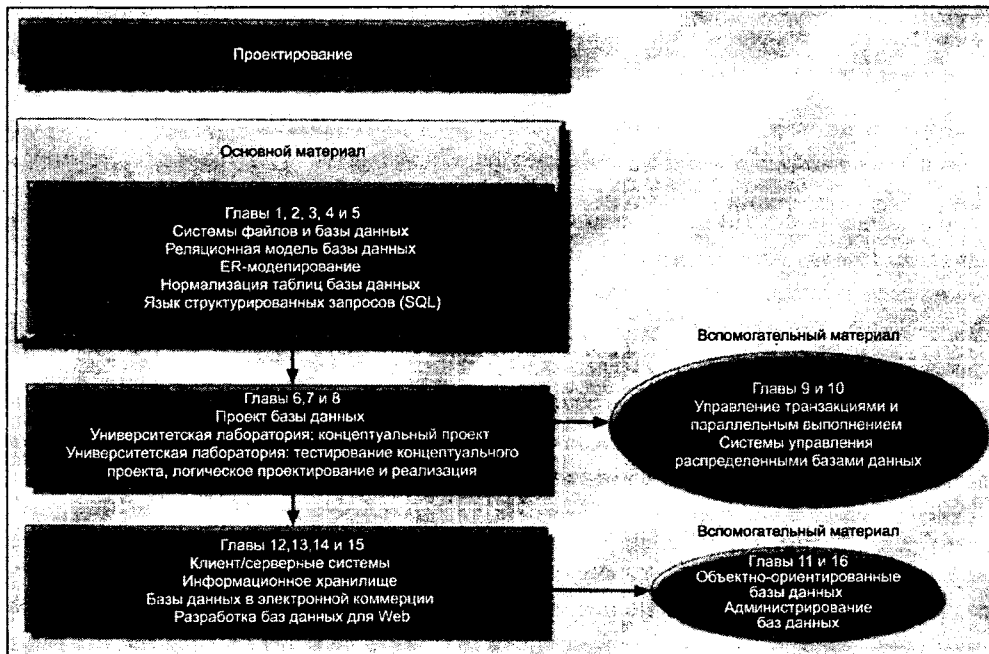


Рис. П.1. Курс проектирования баз данных и проблем реализации

Прикладная основа курса проектирования лучше всего проявляется в лабораторных проектах, когда студенты используют избранное преподавателем ПО при моделировании разрабатываемых ими систем для конечного пользователя. Некоторые из проблем, поставленных в конце глав, достаточно сложны, чтобы использовать их в качестве материала для проекта, но чтобы снабдить студентов подходящим практическим материалом, преподаватель может обратиться к местным фирмам.

Те преподаватели, которые хотят сосредоточиться на технологиях современных баз данных, могут взять рис. П.2 за основу для разработки программы учебного курса. При таком подходе *гл. 6–8* можно пропустить или использовать их только в качестве факультативного чтения, а основное внимание уделить *гл. 9–15*. Поскольку очень трудно разобратся в деталях современных технологий баз данных, не зная базовых понятий и концепций, *гл. 1–5* могут послужить основой последующего изучения такого курса.

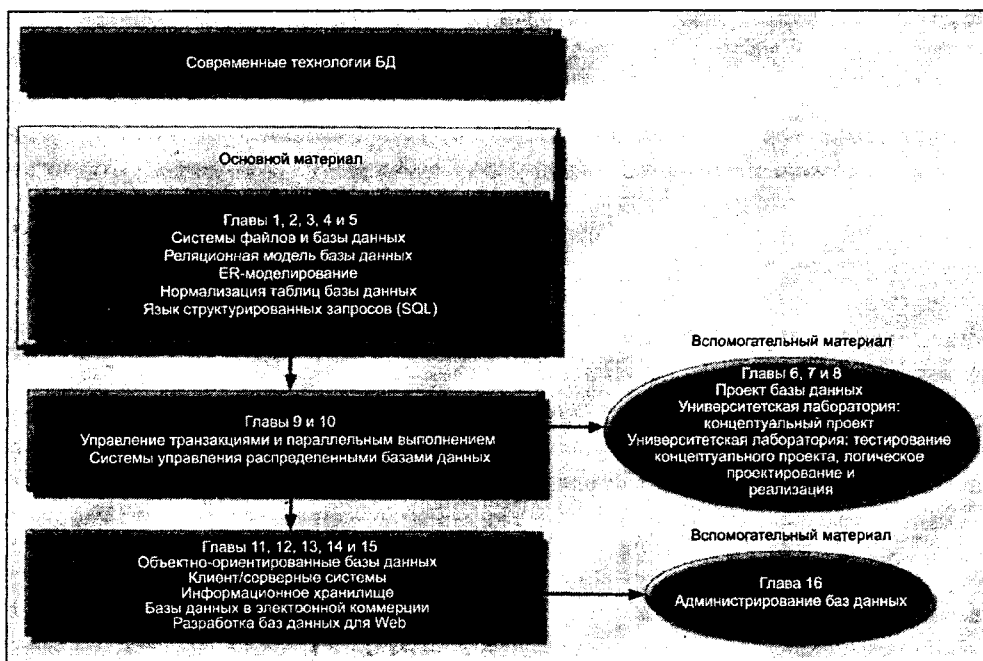


Рис. П.2. Курс изучения современных технологий баз данных

Если преподаватель решил основное внимание уделить проблемам управления базами данных, в этом случае при разработке учебного плана поможет рис. П.3.

Обратите внимание, что некоторые элементы проектирования БД и современные технологии баз данных также нашли отражение в курсе, посвященном исследованию управления базами данных. Это объясняется тем, что невозможно изучать управление базами данных, не понимая принципы их организации.

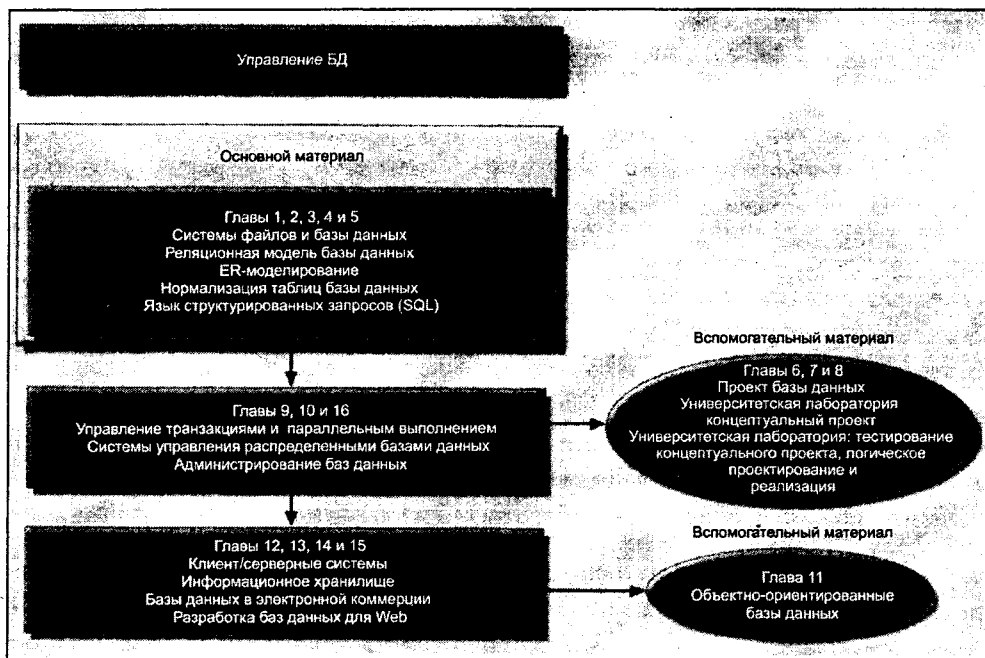


Рис. П.3. Курс изучения проблем управления базами данных

Дополнительные возможности

Мы осознаем, что изучение систем баз данных — тяжкий труд. Если тематика достаточно широка, привязана к практике и весьма детализована, то при этом потребуются разрабатывать реальные проекты и реализовывать их, создавая базы данных, таблицы и связи. Чтобы облегчить изучение, мы обеспечили преподавателей большим вспомогательным материалом².

1. Базы данных, имеющиеся в книге

В это (пятое) издание в качестве методического материала для преподавателя включены все структуры баз данных и содержимое таблиц. При этом гарантируется следующее:

- ☐ проекты баз данных, имеющиеся в тексте книги, отвечают требованиям реализации;
- ☐ проекты отвечают установленным требованиям к информационному обеспечению;
- ☐ иллюстрации всегда соответствуют реальному содержанию баз данных;
- ☐ преподавателю не нужно тратить время на создание структур баз данных и содержимого таблиц для иллюстрации важности проектирования и управления информацией.

² Вспомогательные материалы для преподавателей можно получить на Web-сайте www.course.com, пройдя авторизацию. — Прим. ред.

Несмотря на то что в тексте книги для "снимков экрана" (screenshots) чаще всего использовалось приложение Microsoft Access 2000, тем не менее, данная книга отнюдь не является пособием по Access — это приложение использовалось, в основном, благодаря его исключительной гибкости: таблицы Access 2000 могут быть экспортированы в очень широкий спектр форматов, включая SQL. Фактически для демонстрирования некоторых возможностей SQL в гл. 5 мы экспортировали таблицы Access 2000 в Oracle SQL, а затем использовали Oracle SQL *Plus для получения необходимых результатов запроса.

2. Подробное руководство для преподавателей

Поскольку книга охватывает очень много проблем, преподавателям будет весьма полезен представленный здесь набор методических материалов. Мы уделяли основное внимание практическим аспектам конфигурации баз данных, поэтому посчитали удобным представить процесс проектирования во всех подробностях. Мы также представили замечания о некоторых процедурах проектирования и об альтернативных подходах к этим процессам, допустимых при решении практических задач. Наконец, мы решили сопроводить книгу вопросами и задачами. Мы надеемся, что это поможет вам организовать учебный процесс и сделать его более действенным.

3. Новые задачи

Мы добавили несколько новых задач, тем самым усилив обучающее значение книги, причем некоторые из задач имеют вложенные подзадачи. Описание задач значительно улучшено. Поскольку студенты обучаются, решая определенный набор задач, задачи в книге становятся все более сложными по мере освоения материала предыдущих уроков. Решение таких всеобъемлющих и тщательно проработанных наборов задач дает студентам необходимый опыт разработки реальных проектов, представленных в гл. 7 и 8. Многие из задач являются достаточно сложными и могут служить частью реальных проектов, особенно если они прошли этап реализации и разработки приложений.

4. Информационные хранилища и анализ данных

Гл. 13 представляет информационные хранилища и их роль в поддержке принятия решений. Должным образом разработанные системы поддержки решений применяются как компьютеризованный интерфейс для бизнесменов, помогающий анализировать, творчески осмысливать и понимать бизнес-проблемы. В этой главе рассматривается несколько подходов к реализации систем поддержки решений.

- ☐ Средства оперативного анализа (OLAP — on-line analytical processing), использующие реляционные СУБД.
- ☐ Многомерные базы данных (Multidimensional databases).

Любой из подходов основан на оперативных данных, которые извлекаются, суммируются и хранятся в определенных типах информационных хранилищ. Поскольку мы обсуждаем проектирование и реализацию баз данных транзакционного типа (т. е. операционных, или рабочих), очевидно, нам необходимо рассмотреть реальную среду информационных хранилищ, но так, чтобы студенты не запутались в этих сложных проблемах.

Создание информационных хранилищ привело к тому, что анализ данных (data mining) стал важнейшим компонентом в новейших системах принятия решений. Системы анализа, интерпретации и представления данных предоставляют автоматизированные средства для извлечения данных и их анализа. Процедуры анализа данных разрабатываются для выяснения отношений между ними, что позволяет подготовить почву для решения проблем бизнеса.

Поскольку основное внимание мы уделяем практическому подходу к проектированию БД, мы исследуем критерии, формирующие базис проекта и разработки информационного хранилища. Поэтому мы можем, оставаясь вне основных концепций, показать, каким образом "реалии" информационного хранилища используются в качестве основы для звездообразных схем и многомерных кубов. Обеспечив студентов практическими примерами, мы даем возможность реального проектирования и реализации небольшого информационного хранилища.

5. Все имеющиеся проекты готовы к реализации

Когда мы разрабатывали проекты баз данных в аудитории, мы обнаружили, что очень многие студенты сталкиваются с проблемами при попытке транслировать некоторые из проектов в соответствующие табличные структуры и связи. Эти трудности были связаны с тем, что связи иногда отображались на логическом уровне, а не на уровне реализации. Теперь этот недостаток устранен. Дополнительные связи в среде M:N причиняли особое беспокойство, хотя мы разбивали связь M:N на связи 1:M с помощью составного объекта; некоторые студенты считали такой переход к новой структуре достаточно трудным. В *гл. 3* представлены исправленные иллюстрации связей объектов, сопровождаемые соответствующими примерами, устраняющими эти трудности. Преподаватели, которые используют материалы, представленные в *гл. 5*, для обучения языку SQL, могут также проиллюстрировать влияние надлежащим образом заданных дополнительных связей, исследуя действие SQL-операторов удаления и обновления, записанных в каждом определении табличной структуры.

6. Наиболее употребительные графические модели

Простые ER-модели связей в виде схем Чена (Chen) или "птичьей лапки" (Crow's Foot) сразу же представлены в *гл. 1*. Поскольку многие современные инструментальные средства CASE (computer-aided software engineering — автоматизированное создание программ) используют модель "птичья лапка" в качестве стандарта проек-

тирования БД, мы более подробно исследовали эту модель в гл. 2 и окончательно перешли к ее использованию в гл. 3. Однако некоторые инструменты компьютерного моделирования основаны на альтернативных методах представления ER-модели. Поэтому мы также рассмотрели применение модели "птичья лапка" в методах Rein85 и IDEFIX в новом разделе *"Сравнение обозначений в ER-моделировании"* гл. 3. Используя знакомство с моделями Чена и "птичья лапка" в качестве отправной точки для разработки простой системы управления счетами, студенты смогут легче разобраться с использованием альтернативных моделей.

7. Самодокументированное соглашение об именах

Используя опыт практического преподавания баз данных и комментарии наших читателей к предыдущим изданиям, мы исправили все структуры таблиц для улучшения их восприятия. Мы предложили для наглядности начинать имя атрибута с имени (или части имени) таблицы, к которой привязан атрибут, что делает понятным, какой атрибут используется в качестве внешнего ключа. Последовательно придерживаясь этого соглашения, мы упрощаем отслеживание атрибутов в процессе проектирования. Кроме того, использование такого самодокументированного соглашения об именах упрощает разработку приложений пользователя, особенно при использовании техники drag-and-drop (перетаскивание).

8. Таблицы, иллюстрирующие сложные проблемы проектирования

Некоторые пользователи (не только наши студенты) указывают, что включение содержимого таблиц делает сложные связи более понятными. Поэтому мы включили такие примеры: особое внимание обратите на обсуждение тернарных связей, представленных в гл. 3.

9. Контрольные вопросы

Поскольку курс по базам данных требует практической разработки и реализации, проверка знаний становится неременным условием при таком подходе. Однако элементарные знания понятий и определений могут быть проверены с помощью обширного банка вопросов, предоставленного в распоряжение читателя. Банк вопросов содержит рисунки и представляет собой систему вопросов с ответами в виде множественного выбора, типа "да-нет", заполнения бланка и изложения.

Благодарности

Несмотря на число изданий этой книги, все они всегда были и будут основаны на фундаменте, заложенном в первом издании. Мы убеждены, что наша работа стала успешной, именно потому, что первое издание готовилось под руководством Фрэнка

Раджирелло (Frank Ruggirello), некогда старшего редактора издательства Wadsworth, а впоследствии его владельца. Кроме руководства изданием этой книги, Фрэнк сумел получить рецензию от великого Питера Кина (Peter Keen, РК) — к счастью, доброжелательную — и впоследствии убедил РК написать предисловие к первому изданию. Мы быстро поняли, что предисловие, написанное РК, мгновенно обеспечило книге большую популярность. Хотя мы иногда и считали Фрэнка слишком требовательным начальником, впоследствии мы поняли, что он прекрасный профессионал и замечательный друг. Мы подозреваем, что Фрэнк увидит следы своей деятельности и в нашей сегодняшней работе. Большое спасибо.

Во многом *переделать* успешную книгу труднее, чем написать новую. Одна из самых трудных проблем состоит в том, чтобы решить, какие новые подходы, темы и изменения глубины их рассмотрения могут или не могут быть вписаны в книгу, успешно выдержавшую испытание рынком. Успех легко покрывает недостатки. Именно потому авторы должны помнить, что успех книги только частично зависит от их взгляда на базы данных и широты охвата материала. Комментарии и предложения, сделанные читателями, студентами и рецензентами, играют важную роль в принятии решения относительно рассматриваемой тематики и способов ее представления. Однако в эпоху Интернета получить огромное количество откликов, как говорится, "проще пареной репы".

Поэтому нам выпала непростая задача решить, как вписать все эти отзывы в общую структуру книги и определить, что соответствует ее предназначению, а что нет. Результаты наших усилий — и ваших — нашли отражение в пятом издании. При этом мы старались учесть такие советы, как "глубина охвата книги стала возможной только благодаря вашему ясному повествовательному стилю. Что бы вы ни делали, не измените свой стиль". Когда вы внимательно прочтете это издание, то убедитесь в этом.

Некоторые читатели стали выдающимися рецензентами, предоставив чрезвычайно подробную и обоснованную критику, даже несмотря на в целом положительную оценку стиля книги и широты охвата материала. Наиболее видный из таких рецензентов — доктор Дэвид Хатерли (David Hatherly), отличный профессионал в области баз данных, старший преподаватель School of Information Technology и Charles Sturt University (Mitchell, Bathhurst, Australia). Доктор Хатерли первым передал нам по Интернету свои детально проработанные модули классов базы данных и затем использовал их как основу для всесторонней критики нашего второго издания, предварив его замечанием "весьма доволен книгой и отзывами моих студентов о ней". В этом критическом обзоре и в нескольких последующих сообщениях по электронной почте доктор Хатерли не только указал некоторые ошибки и недочеты, но, более того, предложил соответствующие исправления, которые были включены в третье издание. Ко всему прочему, мы использовали критические замечания доктора Хатерли в качестве руководства при подготовке четвертого издания. Убедившись в том, что мы в точности поняли, чем вызваны его критические замечания, доктор Хатерли представил предложения, позволившие нам улучшить содержание книги. Вся помощь была оказана им абсолютно бескорыстно и без всяких подсказок с нашей стороны. Его усилия приняты нами с сердечной благодарностью.

Каждая техническая книга подвергается тщательному исследованию со стороны нескольких групп рецензентов, выбираемых издателем. Учитывая их огромную работу

по выявлению недостатков, мы выражаем глубокую признательность за вклад в подготовку пятого издания:

- доктору Ахмаду Абухеле (Ahmad Abuhejleh), University of Wisconsin, River Falls;
- Энтони Камерону (Anthony Cameron), Fayetteville Technical Community College;
- доктору Маурин Гринбаум (Maureen Greenbaum), Union County College;
- доктору Хамиду Немати (Hamid Nemati), The University of North Carolina at Greensboro;
- Янушу Сципуле (Janusz Szczypula), Carnegie Mellon University;
- доктору Стефани Тиллман (Stephen Tillman), Wilkes University.

Мы уже отметили, что пятое издание опирается на фундамент, созданный в предыдущих изданиях. Поэтому мы полагаем, что необходимо поблагодарить рецензентов за их усилия при подготовке предыдущих изданий: доктора Реза Бархи (Reza Barkhi, Pamplin College of Business, Virginia Polytechnic Institute and State University), доктора Вэнса Куни (Vance Cooney, Xavier University), Януша Сципулу (Janusz Szczypula, Carnegie Mellon University), доктора Ахмада Абухеле (Ahmad Abuhejleh, University of Wisconsin, River Falls), доктора Теренса М. Барона (Terence M. Baron, University of Toledo), доктора Хуана Эстава (Juan Estava, Eastern Michigan University), доктора Кевина Гормана (Kevin Gorman, University of North Carolina, Charlotte), доктора Джеффа Хедрингтона (Jeff Hedrington, University of Wisconsin, Eau Claire), доктора Германа П. Хоплина (Herman P. Hoplin, Syracuse University), доктора Софи Ли (Sophie Lee, University of Massachusetts, Boston), доктора Майкла Манино (Michael Mannino, University of Washington), доктора Кэрола Крисмана (Carol Chrisman, Illinois State University), доктора Тимоти Хейнца (Timothy Heintz, Marquette University), доктора Германа Хоплина (Herman Hoplin, Syracuse University), доктора Дина Джеймса (Dean James, Embry-Riddle University), доктора Констанс Кнапп (Constance Knapp, Pace University), доктора Мэри Энн Роберт (Mary Ann Robbert, Bentley College), доктора Фрэнсиса Дж. Ван Ветеринга (Francis J. Van Wetering, University of Nebraska), доктора Джозефа Уолса (Joseph Walls, University of Southern California), доктора Стефана К. Солоски (Stephen C. Solosky, Nassau Community College), доктора Роберта Чанга (Robert Chiang, Syracuse University), доктора Криста Коста (Crist Costa, Rhode Island College), доктора Садеш М. Даджел (Sudesh M. Duggal, Northern Kentucky University) и доктора Чанг Кох (Chang Koh, University of North Carolina, Greensboro).

В некотором отношении написание книги напоминает строительство здания. Когда кажется, что 90% работы выполнено, остается выполнить 90% работы. И успех оставшихся 90% зависит, без малейшего преувеличения, от работы редактора. К счастью для нас, Деб Кауфманн (Deb Kaufmann) — выдающийся редактор. Ее непревзойденное мастерство в подведении итогов и классификации комментариев рецензентов сэкономили нам огромное количество времени на то, чтобы увидеть "лес за деревьями". Мы оценили способность Деб всегда задать верный вопрос и дать только правильные рекомендации еще при работе над четвертым изданием. Когда наступило время подготовки пятого издания, мы сразу решили, что нашим редактором будет Деб. Ничего удивительного: Деб лучшая из лучших. Ее предложе-

ния были всегда "по делу", а ее широкие познания по различным проблемам, которыми мы постоянно пользовались, служили для нас указателем в практических решениях (она не только *действительно* знает работу редактора, но и очень хорошо разбирается в базах данных и проблемах телекоммуникаций). Деб, мы уверены, что с вашими обширными познаниями и навыками общения с людьми вы можете поднять благосостояние и 500 компаний. Вы, несомненно, лучшая среди людей и среди редакторов. Нам приятно думать, что именно ваши профессиональные действия позволяют надеяться, что это пятое издание значительно лучше своих предшественников. Огромное спасибо! (Мы собираемся удочерить Деб, поскольку понимаем, что без нее будет весьма проблематично выпустить в свет последующие издания.)

Когда книга была написана, к работе приступили редакторы рукописи. Редактирование рукописи требует необычайного внимания к деталям, критического взгляда для наведения окончательного глянца. По мере выполнения этого редактирования становилось ясно, что наш редактор рукописей из службы Foxxe Editorial Services является высочайшим профессионалом своего дела.

Дженнифер Гоген (Jennifer Goguen), выпускающий редактор, сумела направить весь процесс издания в правильное русло. Она управляла редакторами рукописей, корректорами и текстовыми процессорами, пока не убедилась, что рукопись собрана воедино. Это большая и трудная работа, но Дженнифер все время держала нас в курсе. Кроме того, что в деле для нее ничего невозможного нет, ее способность работать с совершенно измотанными авторами была для нас просто подарком. Даже по электронной почте Дженнифер внушала нам ощущение спокойствия, компетентности и присутствия юмора.

Маргарита Донован (Margarita Donovan), ответственная за выпуск, ловко обходилась с каждодневными корректурами и продвижением рукописи. Она работала совместно с Деб Кауфманн, Дженнифер Гоген, Дженнифер Лок (Jennifer Locke) и нами, обеспечивая доставку рукописи в нужное место и в нужное время. Короче говоря, Маргарита может сделать так, чтобы и волки были сыты, и овцы целы. Мы верим, что Маргарита никогда не засидится, и полагаем, что она вообще может руководить чем угодно с легкостью и улыбкой.

Дженнифер Лок, старший редактор, отвечает за все серии книг издательства Course Technology, посвященные информационным системам. Ее познания в этой сфере помогли всем — и студентам, и профессорам, — а также авторам и всем, кто был в нашей команде, выпустить книгу по базам данных, которая, как нам кажется, будет лучшей книгой на рынке. Мы подозреваем, что успех или неудача книги на рынке определяется множеством вещей. Нам также кажется, что для гарантии успеха самая важная вещь — способность старшего редактора правильно позиционировать книгу. Умение Дженнифер напряженно работать, ее знания и энтузиазм ставят эту книгу в ряд лучших.

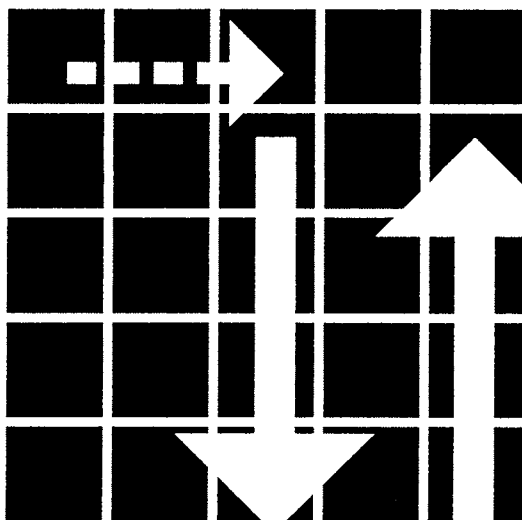
Мы также благодарны нашим студентам за их комментарии и предложения. Они-то и есть самая главная причина, побудившая нас написать эту книгу. На самом деле, если не общаться со студентами, справедливо полагающими, что их преподаватели обеспечат их высококачественной и современной литературой, то написание книги можно считать просто проявлением мазохизма. Напряженная работа в университете предполагает, что писать придется по ночам, в выходные, а также во время отпуска

(а мне еще говорят, что есть такая вещь, как свободное время!). И все же здесь есть положительный момент: наши усилия в написании книги основаны на большой исследовательской практике, что в конечном счете гарантирует наилучшее понимание материалов книги нашими студентами. По-видимому, это и есть причина того, что в университетах благосклонно относятся к написанию книг.

В конце, но, конечно же, не в последнюю очередь, мы благодарим за поддержку наши семьи. Они смирились с нередкими бессонными ночами, с отсутствием свободного времени и даже с тем, что во время более чем годовой подготовки издания у нас вообще не было выходных. Мы вам обязаны всем, и посвящение вам — всего лишь малое отражение того важного места, которое вы занимаете в наших сердцах.

Питер Роб (Peter Rob)

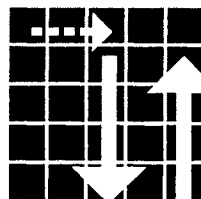
Карлос М. Коронел (Carlos M. Coronel)



ЧАСТЬ I

ОСНОВЫ БАЗ ДАННЫХ

Глава 1



Системы файлов и базы данных

В этой главе мы обсудим следующие темы:

- ☐ что такое база данных, для чего она нужна, и почему так важно проектирование базы данных;
- ☐ происхождение современных баз данных от файлов и файловых структур;
- ☐ недостатки управления данными в файловых структурах;
- ☐ что такое система управления базой данных (СУБД) и как она взаимодействует с базой данных;
- ☐ типы систем и модели баз данных.

Обзор

Хорошие решения требуют достоверной информации, которую приходится извлекать из неупорядоченных данных. Данными можно более эффективно управлять, если их разместить в базе данных (БД). В этой главе мы обсудим, что представляет собой база данных, для чего она нужна, и почему с ее помощью можно добиться лучших результатов, чем с помощью других методов управления данными. Эта глава поможет вам понять и оценить значение проектирования базы данных.

Прародителями современных баз данных можно считать компьютерные файловые структуры (системы файлов). Хотя системы файлов сейчас постепенно отходят в прошлое, все же понимать их очень важно, поскольку основные свойства систем файлов являются источником некоторых важнейших ограничений в управлении данными. Если как следует разобраться в недостатках файловых структур, то станет понятнее, какие (и почему) свойства баз данных являются наиболее важными, и как найти им верное применение.

После обсуждения развития систем файлов и изучения их основных свойств мы рассмотрим проблемы, которые имеют место при использовании системы файлов. Затем будут представлены основные идеи, помогающие устранить дефекты в управлении базой данных. Наконец, мы сделаем краткий экскурс в историю развития баз данных, рассмотрим основные свойства, достоинства и недостатки основных моделей БД: иерархической, сетевой, реляционной, модели "сущность-связь" (ER-модели) и объектно-ориентированной модели. Рассматривая историю развития этих моделей и выясняя преимущества и недостатки каждой из них, мы заложим основы понимания проектирования баз данных, проблем, связанных с их внедрением, что

впоследствии будет детально изучаться в остальных главах книги. Обсуждение развития баз данных мы завершим кратким анализом влияния, которое оказывает Интернет на рынок баз данных. Затем мы поясним, почему мы уделяем основное внимание реляционной модели баз данных, которую и в настоящее время все еще можно считать доминирующей на рынке.

1.1. Введение в базы данных

Чтобы понять движущую идею проектирования баз данных, необходимо уяснить различие между *неупорядоченными данными* и *упорядоченными, структурированными данными*. Неупорядоченные данные это "сырье", необработанные сведения. Термин "сырье" здесь использован, чтобы подчеркнуть, что этот материал не подвергался никакой обработке с целью его структуризации. Предположим, что компания ROBCOR контролирует все сделки по счетам-фактурам в своих двух подразделениях. Каждый из этих счетов содержит сведения примерно такого вида:

invoice number (номер счета-фактуры) = 300124

invoice date (дата выписки счета) = 12-JAN-2002

sales amount (сумма платежа) = \$125.98

Далее предположим, что эти два подразделения компании ROBCOR в первый квартал 1997 года и первый квартал 2001 года выписали 1 380 456 и 1 453 907 счетов-фактур соответственно. Следовательно, множество неупорядоченных сведений компании ROBCOR будут содержать 2 834 363 номеров счетов-фактур, 2 834 363 дат выписки этих счетов и 2 834 363 сумм платежей. Принимая во внимание столь обильный фактический материал, к которому необходимо присовокупить информацию о сотрудниках каждого из этих двух подразделений, работавших в каждом из этих кварталов, можно ли всерьез думать о том, что руководство компании сможет извлечь какую-то продуктивную информацию об эффективности сделок в расчете на одного сотрудника каждого из этих двух подразделений? Скорее всего, нет. Управленческий персонал просто не сможет обработать такое количество счетов. С другой стороны, если мы предварительно обработаем этот материал, рассчитав поквартальный объем продаж для каждого из двух подразделений, а затем, разделив квартальные суммы сделок на расчетное число сотрудников в данном квартале, то получим результаты, представленные на рис. 1.1. Этот график наглядно демонстрирует, что эффективность сделок в расчете на одного сотрудника в Подразделении 1 (Division 1) выше, чем в Подразделении 2 (Division 2). Более того, очевидно, что имеющийся разрыв в эффективности продаж в двух подразделениях увеличивается. Короче говоря, у руководителей компании ROBCOR теперь есть структурированная (упорядоченная) информация, которая может служить основой для принятия решения.

Говорят, что мы живем в "век информации". Это означает, что получение достоверной, существенной и своевременной информации — ключ к решению любой задачи. В свою очередь выработка верных решений — ключ к успешному ведению бизнеса на мировом рынке.

Выделим из всего сказанного выше некоторые основные положения.

- ☐ Неструктурированные, "сырые" сведения — это основной источник информации.
- ☐ Информация структурируется путем обработки "сырых" сведений.

- ☐ В структурированной информации выявляется ее предназначение.
- ☐ Достоверная, существенная и своевременная информация — ключ к принятию верных решений.
- ☐ Верные решения — ключ к успеху предприятия на мировом рынке.



Рис. 1.1. Сумма сделок в расчете на одного сотрудника для двух подразделений компании ROBCOR

Для достоверной, своевременной и значимой информации необходимы точные данные. Такие данные должны создаваться и храниться должным образом, в формате, обеспечивающем простоту доступа и обработки. Поскольку данные представляют собой важнейший базовый ресурс, ими необходимо тщательнейшим образом управлять. *Управление данными* это дисциплина, которая изучает методы создания, надлежащего хранения и извлечения данных. Принимая во внимание столь важную роль, которую играет информация в нашей жизни, стоит ли удивляться, что управление данными является основой деятельности любых предприятий, правительственных органов, сферы услуг и благотворительных организаций.

Как правило, эффективное управление данными предполагает использование компьютерных баз данных. *База данных* это интегрированная компьютерная структура совместного доступа, в которой размещаются следующие сведения:

- ☐ данные конечных пользователей, т. е. сведения, отражающие сферу интересов конечного пользователя;
- ☐ *метаданные (metadata)*, или данные о данных, с помощью которых осуществляется интегрирование (объединение) данных.

Метаданные описывают свойства данных и совокупность отношений, которыми связаны данные, хранящиеся в БД. В некотором смысле база данных представляет собой очень хорошо организованную электронную картотеку, в которой мощное программное обеспечение, называемое *системой управления базой данных*, помогает управлять содержимым этой картотеки. *Система управления базой данных* (СУБД) представляет собой совокупность программ, с помощью которых осуществляются управление структурой базы данных и контроль доступа к данным, хранящимся в ней. СУБД позволяет нескольким приложениям или пользователям осуществлять совместный доступ к данным.

Поскольку данные несут в себе весьма важные сведения, на основе которых и происходит структурирование информации, имеется целый ряд веских причин, по которым СУБД считается важнейшей составляющей всего информационного сообщества. Вот наиболее значимые из них.

- ❑ Поскольку роль данных так важна, в нашем распоряжении должен иметься надежный способ управления ими. Далее в этой книге будет показано, что СУБД помогает сделать управление данными более действенным и эффективным, чем оно было до появления СУБД.
- ❑ В состав СУБД входит язык запросов, позволяющий оперативно реагировать на нерегламентированные запросы (*запрос* это просто вопрос к БД, а *нерегламентированный запрос* это запрос к БД, возникающий в определенной ситуации и требующий немедленного выполнения). Например, если конечный пользователь имеет дело с большим количеством данных о продажах, то ему могут потребоваться оперативные ответы, допустим, на такие вопросы:
 - Каков объем продаж (в долларах) продукции за последние шесть месяцев?
 - Каков размер вознаграждения для каждого коммивояжера за последние три месяца?
 - Сколько клиентов имеют долг по кредиту в размере \$3000 и более?
- ❑ СУБД помогает создать рабочую среду, в которой конечным пользователям обеспечивается более удобный и хорошо управляемый доступ к данным, нежели имевшийся у них перед тем, как СУБД стала стандартным средством управления данными. Такой доступ позволяет конечным пользователям быстро реагировать на изменения в рабочей среде. Доступность данных в сочетании с инструментальными средствами, преобразующими данные в удобную форму представления, дает возможность конечным пользователям получать быстрые и информационно емкие решения, что и определяет успех или неудачу предприятий в сфере мировой экономики.
- ❑ Широкий доступ к хорошо управляемым данным создает обобщенное представление обо всех операциях предприятия. Так, можно легко проследить, каким образом действия в одном сегменте предприятия оказывают влияние на другие сегменты. Одним словом, СУБД позволяет получить верное представление о картине деятельности предприятия в целом.
- ❑ В этой книге будет показано, что вероятность противоречивости данных значительно уменьшается в тех БД, которые спроектированы должным образом и управляются с помощью СУБД. Более достоверные данные позволяют структурировать информацию наилучшим образом, что является основой наилучших решений.

здесь перечислены далеко не все преимущества использования СУБД. На самом деле, когда вы приступите к изучению технических сведений о базах данных и их правильном проектировании, вы откроете для себя еще очень много достоинств СУБД.

рис. 1.2 иллюстрирует роль СУБД при взаимодействии между пользователем и базой данных. В сущности СУБД играет роль посредника между пользователем и БД, транслируя запросы пользователя в сложный код, необходимый для выполнения таких запросов. СУБД скрывает от прикладных программ, использующих БД, ее внутреннюю структуру. Прикладные программы могут быть написаны программистом на каком-либо языке программирования (например, на COBOL), или созданы с помощью сервисных программ СУБД.

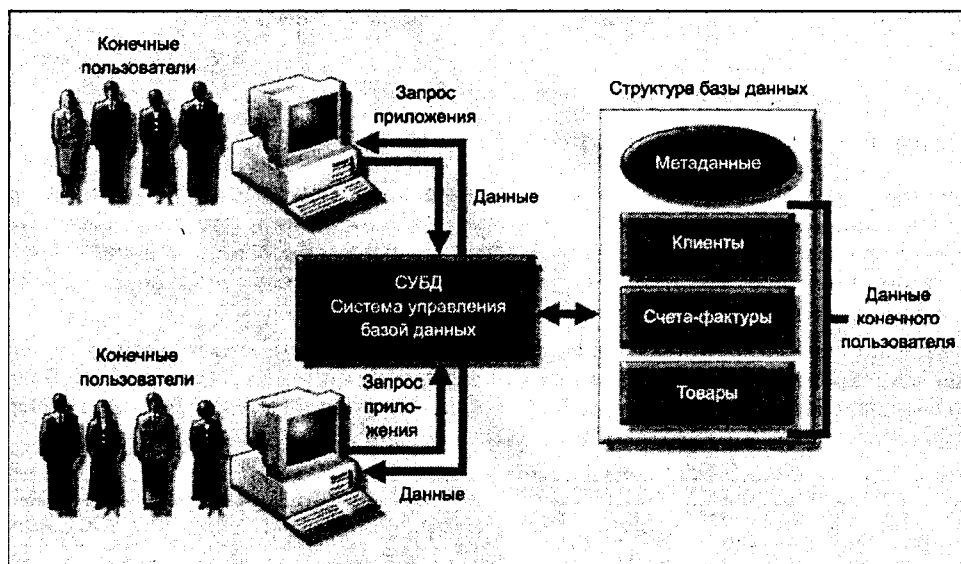


Рис. 1.2. СУБД управляет взаимодействием между конечным пользователем и базой данных

Важно всегда помнить, что СУБД представляет собой приобретенный вами коммерческий программный продукт, и у вас нет возможности вносить в него какие-либо изменения. Поэтому когда мы говорим о проектировании базы данных, то имеем в виду проектирование структуры БД, которая будет использована для хранения и управления данными, а не разработку программного обеспечения СУБД. После завершения проектирования базы данных СУБД берет на себя управление всеми важными процедурами, необходимыми для перевода представления структуры данных проектанта в форму, удобную для компьютера.

Примечание

Большинство вопросов, связанных с проектированием, внедрением и управлением базами данных, которые встречаются в этой книге, относятся к так называемым *операцион-*

ным базам данных (*operational databases*), раньше их еще называли продуцирующими, или рабочими базами данных (*production databases*), или к *транзакционным* базам данных (*transactional databases*). Это базы данных, с которыми вы встречаетесь при выполнении таких действий, как запись на курсы обучения, регистрация автомобиля, покупка товара, занесение денег на счет или получение денег со счета. Хотя в таких базах данных хранится масса полезной информации, тем не менее, существуют специализированные БД (которые называются *информационными хранилищами* — *data warehouses, DW*), разработанные специально для информационных нужд. Мы подробно рассмотрим разработку, внедрение и использование информационных хранилищ во всем их многообразии в гл. 13. Основное внимание в этой книге уделяется операционным базам данных, что основано на двух соображениях. Во-первых, с такими базами данных мы чаще всего сталкиваемся в повседневной жизни. Во-вторых, информационные хранилища (банки данных) в основном получают данные из операционных баз данных. Поэтому если операционная база данных плохо спроектирована и плохо управляема, то основной источник для информационного хранилища теряет большую часть своего значения.

1.1.1. Почему так важно проектирование базы данных

Хорошая база данных не получается случайно, структура ее содержимого должна тщательно прорабатываться. На самом деле, проектирование базы данных — настолько важная часть работы с БД, что основная часть этой книги посвящена именно разработке добротных методов проектирования баз данных. Даже хорошая СУБД будет плохо работать с неудачно спроектированной базой данных.

Правильно спроектированная БД облегчает управление данными и становится ценным поставщиком информации. Плохо спроектированная БД вероятнее всего станет накопителем избыточной информации, т. е. неоправданного дублирования данных. Избыточность, как правило, затрудняет выявление ошибок в данных.

База данных содержит избыточные данные, если одна и та же информация об одном и том же логическом объекте хранится в разных местах (логическим объектом, *сущностью* — *entity* — считается персона, местоположение или предмет, сведения о которых подлежат сбору и хранению). Например, избыточность данных имеет место, если номер телефона клиента хранится и в файле клиентов, и в файле коммивояжеров, и в файле счетов-фактур. Избыточность данных может стать источником несовместимости данных, когда трудно определить, какая информация является правильной, а какая нет (например, если телефон клиента был заменен новым только в файле счетов-фактур, а в других остался прежним). В отчетах, выполненных на основе такой информации, могут содержаться противоречивые сведения в зависимости от того, какая из версий данных была использована. Короче говоря, неконтролируемая избыточность данных — типичный признак плохо спроектированной БД.

Неудачно спроектированная база данных может служить источником ошибок, приводящим к неудачным решениям. Скверный проект базы данных, в конечном счете, может сам себя "исправить": организации, в которых используются плохо спроектированные БД, как правило, становятся банкротами (поскольку их менеджерам недоступна своевременная или, по крайней мере, достоверная информация), тем самым аннулируя и себя, и весь неудачный проект БД.

Поскольку база данных является источником информации, ее проектирование становится предметом детального изучения в современных информационных средах.

Проектирование базы данных — слишком важное дело, чтобы в данном случае полагаться "на авось". Вот почему в колледжах студенты изучают проектирование баз данных, вот почему малые и большие предприятия любых форм собственности посылают своих представителей на семинары, посвященные проектированию БД, и, наконец, именно поэтому специалисты по проектированию БД зачастую имеют столь солидный доход.

1.1.2. Практический подход к проектированию базы данных

В наш "век информации" разработка полезных БД имеет огромное практическое значение. По этой причине мы уделим самое пристальное внимание принципам и идеям, непосредственно влияющим на практические подходы к проектированию БД. Если вы хотите узнать, как спроектировать практически полезную базу данных, то эта книга станет для вас хорошим подспорьем. Если же вашей целью является постижение всех глубин теории баз данных, то эта книга не для вас.

Чтобы подключить материал, необходимый для разработки словаря базы данных и для лучшего понимания основ БД, были созданы два законченных приложения в процессе логического проектирования базы данных (*логическое проектирование* это проектирование структуры БД). Мы полагаем, что практический подход к процессу проектирования должным образом подготовит вас к разработке собственных проектов.

1.2. Предыстория баз данных: файлы и системы файлов

Исторически сложилось так, что первые компьютерные приложения предназначались для решения организационных задач: обработка заказов и поставок, составление платежных ведомостей, разработка графиков выполнения работ и т. д. Такие приложения получали информацию из файлов, хранящихся на компьютере. Запросы следовали один за другим (сколько было продано, кем и кому?), а отчеты создавались с целью преобразования хранимых в компьютере данных к виду, удобному для руководства. В этой главе мы рассмотрим историю развития компьютерных файловых структур, создаваемых с целью получения более подробной и своевременной информации.

Хотя файловые структуры в настоящее время постепенно уходят в прошлое, есть несколько веских причин для их детального изучения:

- файловые структуры представляют собой интересную ретроспективу способов обработки данных;
- философ Джордж Сантаяна (George Santayana) однажды заметил, что "те, кто не помнит прошлое, обречены его повторить". Некоторые из дефектов, свойственных файловым структурам, могут вновь появиться в программном обеспечении БД, если его пользователям не будут известны ошибки управления данными;
- осмысление основных свойств файловых структур упростит изучение более сложных методов проектирования БД;

- ☐ если вам придется преобразовать устаревшую файловую структуру в базу данных, то в этом случае знание основных ограничений файловой структуры будет весьма полезным.

В недавнем прошлом руководители практически любого малого предприятия обрабатывали (а некоторые обрабатывают и по сей день) важную информацию с помощью картотек. Такие картотеки традиционно представляли собой коллекцию папок с документами, каждая из которых помечалась и хранилась в картотечном ящике.

Упорядочивание данных внутри папок с документами определялось предполагаемым использованием данных. В идеальном случае содержимое каждой папки было логически взаимосвязанно. Например, папки в кабинете врача должны были содержать данные о пациентах — отдельная папка на каждого пациента. Все данные в такой папке описывали историю болезни отдельного пациента. Подобным образом служащий отдела кадров организовывал персональные данные по категориям служащих (клерки, технический персонал, коммерсанты, администрация и т. д.). При этом папка с пометкой "технический персонал" должна была содержать данные только о сотрудниках, относящихся к техническому персоналу.

До тех пор пока объем данных был относительно мал, и руководителю организации требовались всего лишь несколько отчетов, такие картотеки прекрасно исполняли роль хранилища данных. Однако по мере роста предприятия и повышения требований к отчетности обработка информации с помощью картотек становилась все более трудоемкой. Фактически поиск и использование данных среди возрастающего количества картотечных ящиков становились настолько долгими и обременительными, что возможность извлечения сколько-либо полезной информации постепенно сводилась к нулю. Вот лишь несколько вопросов, на которые владелец предприятия хотел бы получить ответ.

- ☐ Какой продукт продавался лучше всего на прошлой неделе?
- ☐ Каков объем сделок в долларах за текущий день, неделю, месяц, квартал, год?
- ☐ Каково положение дел с продажами в настоящий момент времени по сравнению с прошлой неделей, прошлым месяцем или прошлым годом?
- ☐ Какие статьи расходов увеличились, уменьшились или остались на прежнем уровне в течение прошлой недели, месяца или года?
- ☐ Не свидетельствует ли уровень продаж о необходимости пополнить складские запасы торговой точки?

И этот список пополняется с каждым днем!

Есть и более веские причины, побуждающие современного руководителя анализировать все больший объем информации. Деятельность некоторых предприятий — например чартерных авиационных компаний — строго регулируется правительством. Руководители таких компаний должны выпускать объемистые подробные квартальные отчеты, содержимое которых тщательно проверяется. Создание такой отчетности и ответы на различные вопросы во время периодических внеплановых проверок, выполняемых федеральными службами, требует весьма эффективного доступа к данным.

К сожалению, получить такой отчет с помощью старинной картотеки практически невозможно. Ведь фактически руководителям потребуются недели интенсивной ра-

боты для создания отчетности, которую необходимо ежеквартально представлять в правительственные органы, даже при том условии, что картотека содержится в идеальном состоянии. В результате руководитель предприятия приходит к мысли о необходимости создания компьютерной системы обработки данных и создания требуемых отчетов.

Перевод картотеки в соответствующую файловую структуру компьютера был достаточно сложной технической задачей (мы-то уже привыкли к современному дружественному пользовательскому интерфейсу компьютера и позабыли, какими враждебными и непонятными казались компьютеры пользователям в недалеком прошлом!). По этой причине возникла потребность в профессионалах особого рода — специалистах по обработке данных (Data Processing specialist, DP), которых необходимо было нанять или "взрастить" из имеющихся сотрудников. DP-специалист умел строить необходимые структуры файлов, зачастую разрабатывая и программное обеспечение, помогающее управлять данными в таких структурах, а также писал прикладные программы, которые автоматически создавали необходимые отчеты на основе данных файла. В то время было порождено огромное число компьютеризованных систем файлов.

Изначально компьютерные файлы в файловой структуре были очень похожи на документы картотеки. Простейший пример пользовательского файла данных для небольшой страховой компании, историю которой мы далее и рассмотрим, представлен на рис. 1.3.

Примечание

Далее вы узнаете, что хотя файловую структуру, представленную на рис. 1.3, можно обнаружить в ранних системах файлов, все же она является не совсем правильной. Поиск решения для структурно-зависимых задач способствовал развитию различных концепций и методов хранения данных, которые мы обсудим позже в этой главе.

	C_NAME	C_PHONE	C_ADDRESS	C_ZIP	A_NAME	A_PHONE	TP	AMT	REN
▶	Alfred A. Ramas	615-844-2573	218 Fork Rd., Babs, TN	36123	Leah F. Hahn	615-882-1244	T1	\$100.00	05-Apr-2002
	Lecna K. Dunne	713-894-1238	Box 12A, Fox, KY	25246	Alex B. Alby	713-228-1249	T1	\$250.00	16-Jun-2002
	Kathy W. Smith	615-894-2285	125 Oak Ln, Babs, TN	36123	Leah F. Hahn	615-882-2144	S2	\$150.00	29-Jan-2001
	Paul F. Olowski	615-894-2180	217 Lee Ln., Babs, TN	36123	Leah F. Hahn	615-882-1244	S1	\$300.00	14-Oct-2002
	Myron Orlando	615-222-1672	Box 111, New, TN	36155	Alex B. Alby	713-228-1249	T1	\$100.00	28-Dec-2002
	Amy B. O'Brien	713-442-3381	387 Troll Dr., Fox, KY	25246	John T. Okon	615-123-5589	T2	\$850.00	22-Sep-2002
	James G. Brown	615-297-1228	21 Tye Rd., Nash, TN	37118	Leah F. Hahn	615-882-1244	S1	\$120.00	25-Mar-2002
	George Williams	615-290-2556	155 Maple, Nash, TN	37119	John T. Okon	615-123-5589	S1	\$250.00	17-Jul-2002
	Anne G. Ferriss	713-382-7185	2119 Elm, Crew, KY	25432	Alex B. Alby	713-228-1249	T2	\$100.00	03-Dec-2002
	Olette K. Smith	615-297-3809	2782 Main, Nash, TN	37118	John T. Okon	615-123-5589	S2	\$500.00	14-Mar-2002

C_NAME	=	Имя клиента	A_NAME	=	Имя агента
C_PHONE	=	Телефон клиента	A_PHONE	=	Телефон агента
C_ADDRESS	=	Адрес клиента	TP	=	Тип страховки
C_ZIP	=	Почтовый индекс клиента	AMT	=	Сумма страхового полиса в тыс. долларов
			REN	=	Дата обновления страховки

Рис. 1.3. Содержимое пользовательского файла

Описание компьютерных файлов требует специального словаря. Каждая отрасль старается разработать свой жаргон, чтобы все исполнители хорошо понимали друг друга. Краткий основной словарь, которым оперируют пользователи систем файлов, приведенный в табл. 1.1, поможет и вам лучше понять дальнейшие рассуждения.

Таблица 1.1. Основная терминология файловых систем

Термин	Описание
Данные (Data)	"Сырые" необработанные сведения, "куча", такие, например, как номера телефонов, даты рождения, имена клиентов, дата, объем сделок и т. д. Данные не играют большой роли, если они не организованы некоторым логическим образом. Наименьший фрагмент данных, распознаваемый компьютером, это простой символ, например, буква "А", число "5" или какой-нибудь символ, например, "\". Для хранения простого символа в памяти компьютера требуется один байт
Поле (Field)	Символ или группа символов (алфавитных или цифровых), которые имеют определенное значение. Поля используются для определения и хранения данных
Запись (Record)	Логически связанный набор из одного или более полей, описывающих персону, местоположение или предмет. Например, поля, составляющие запись для пользователя с именем J. D. Rudd должны содержать имя J. D. Rudd, адрес, номер телефона, дату рождения, размер кредита и баланс неплатежа
Файл (File)	Совокупность связанных записей. Например, файл может содержать данные о поставщиках компании ROBCOR или записи о студентах, внесенных в список учащихся университета Gigantic University

Используя терминологию файловых систем, представленную в табл. 1.1, мы можем идентифицировать компоненты файла, показанные на рис. 1.3. Обратите внимание, что файл CUSTOMER (клиент), представленный на рис. 1.3, содержит 10 записей. Каждая запись состоит из девяти полей: C_NAME (имя), C_PHONE (телефон), C_ADDRESS (адрес), C_ZIP (почтовый индекс), A_NAME (имя), A_PHONE (телефон), TP (тип), AMT и REN. Все 10 записей хранятся в именованном файле. Поскольку файл, представленный на рис. 1.3, содержит информацию о клиенте, файлу присвоено имя CUSTOMER.

На основе содержимого файлов типа CUSTOMER DP-специалист писал программы, создававшие необходимые отчеты для коммерческого отдела:

- ☐ ежемесячные сводки, в которых отражены типы и размеры страховых полисов, проданных каждым агентом (такие отчеты используются при анализе деятельности каждого агента);
- ☐ ежемесячные сверки пользователей с целью выяснения, с кем из них нужно связаться для возобновления действия страхового полиса;
- ☐ отчеты, анализирующие соотношение типов страховых полисов, проданных каждым агентом;

- периодическая рассылка клиентам писем, информирующих клиента о результатах действия страхового полиса, а также для уведомления клиента при получении им каких-либо поощрений и премий.

Со временем писались дополнительные программы, которые создавали новые отчеты. Хотя для определения содержимого отчета и написания программы требовалось какое-то время, руководитель коммерческого отдела не сожалел о старой доброй картотеке — компьютер сэкономил массу времени и сил. Отчеты производили должное впечатление, а возможность выполнять сложный поиск помогала извлекать информацию, необходимую для принятия верных решений.

Затем в коммерческом отделе был создан файл с названием SALES (продажи), помогающий следить за ежедневными сделками. По мере необходимости получения новых отчетных форм были созданы дополнительные файлы. На самом деле успешная деятельность коммерческого отдела стала столь очевидной, что и руководитель отдела кадров попросил привлечь DP-специалиста для автоматизации обработки платежных ведомостей, а также других функций. В результате DP-специалиста попросили создать файл AGENT (агенты), представленный на рис. 1.4. Данные в файле AGENT использовались для выдачи чеков, отслеживания уплаты налогов, итоговых результатов страхования и т. д.

A_NAME	A_PHONE	A_ADDRESS	ZIP	HIRED	YTD_PAY	YTD_FIT	YTD_FICA	YTD_SLS	DEP
▶ Alan B. Abn	713-228-1249	123 Toll, Nash, TN	37119	01-Nov-1993	\$20,566.24	\$4,332.21	\$1,534.57	\$1,735.00	3
Leah F. Mohn	815-882-1244	334 Main, Fox, KY	25248	23-May-1984	\$25,213.76	\$5,934.75	\$1,788.38	\$4,967.00	0
John T. Okon	815-123-5589	452 Elm, New, TN	36155	15-Jun-1989	\$23,198.29	\$4,332.24	\$1,689.44	\$3,093.00	2

A_NAME	= Имя агента	YTD_PAY	= Оплата на сегодняшний день
A_PHONE	= Телефон агента	YTD_FIT	= Налоги, оплаченные на сегодняшний день
A_ADDRESS	= Адрес агента	YTD_FICA	= Оплата социального страхования
ZIP	= Почтовый индекс агента	YTD_SLS	= Объем продаж на сегодняшний день
HIRED	= Дата приема агента на работу	DEP	= Число иждивенцев

Рис. 1.4. Содержимое файла AGENT

По мере роста числа файлов небольшая файловая структура, наподобие представленной на рис. 1.5, все разрасталась и разрасталась. Каждый файл в этой системе принадлежал сотруднику или отделу, по заказу которого он был создан.

По мере роста системы файлов объем задач программирования рос еще быстрее, поэтому DP-специалисту разрешили принять на работу дополнительный штат программистов. Возросший размер системы файлов к тому же стал причиной приобретения нового компьютера с более мощными ресурсами. После приобретения компьютера и приема на работу программистов DP-специалист практически перестал заниматься программированием и стал уделять основное внимание управлению техническими ресурсами и кадрами. Поэтому DP-специалист получил должность руководителя отдела обработки данных — DP-менеджера. Несмотря на эти организационные изменения, все же основная деятельность нового DP-отдела сводилась к программированию, и неминуемо DP-менеджер большую часть своего рабочего времени исполнял роль старшего программиста-наблюдателя и занимался поиском ошибок в программах.

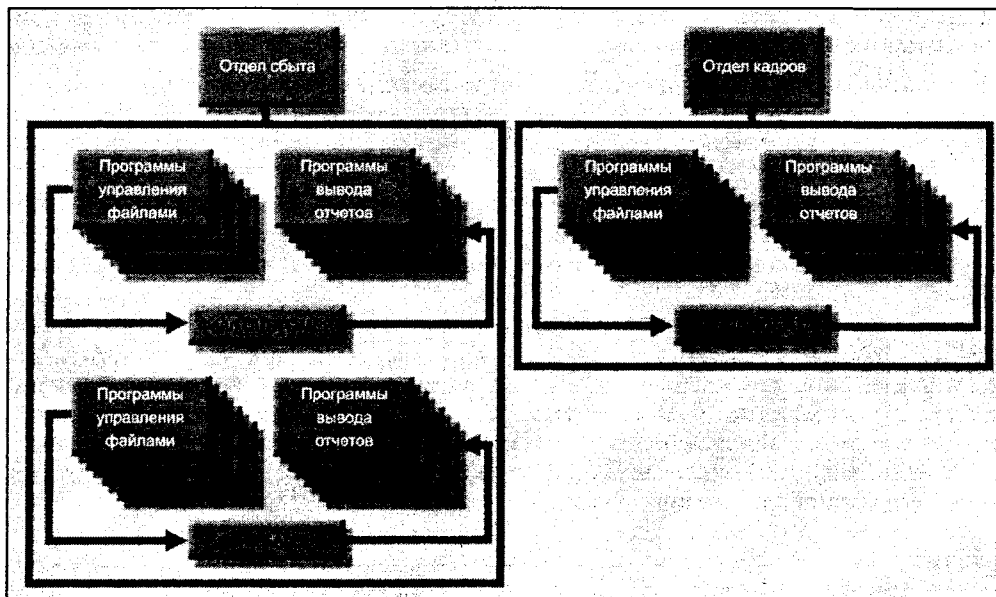


Рис. 1.5. Простейшая система файлов

Примечание

С точки зрения управления система файлов включает в себя следующие составные части.

1. *Аппаратные средства.* Компьютер.
2. *Программное обеспечение.* Операционная система, утилиты, файлы, программы управления файлами и прикладные программы для создания отчетов на основе данных из файлов.
3. *Сотрудники.* DP-менеджер, DP-специалисты, программисты и конечные пользователи.
4. *Процедуры.* Инструкции и правила, в соответствии с которыми разрабатываются и используются программные компоненты.
5. *Данные.* Совокупность "сырых" неупорядоченных сведений.

1.3. Оценка системы файлов

Несмотря на то, что в этом разделе критикуется производительность файловых структур, необходимо помнить, что они служили верой и правдой в течение двух десятков лет, а это очень большой отрезок времени по меркам стремительной компьютерной эры. Приводимая здесь критика преследует две основные цели:

- осмысление недостатков системы файлов позволит нам понять причины возникновения и развития баз данных;

- ☐ отнюдь не все перечисленные ниже недостатки свойственны именно системам файлов. Непонимание этих проблем может привести к их повторению в среде базы данных, невзирая на то, что в современных технологиях БД сделать такие просчеты достаточно трудно.

1.3.1. Управление системой файлов

Алгоритмизация даже простейшей задачи поиска требует достаточно интенсивного использования *языков программирования третьего поколения* (third-generation language — 3GL). Эти языки программирования предполагают, что программист должен определить как то, *что* необходимо сделать, так и то, *как* это необходимо выполнить. К языкам 3GL относятся, например, COBOL (COmmon Business-Oriented Language), BASIC (Beginner's All-purpose Symbolic Instruction Code) и FORTRAN (FORmula TRANslation).

Программирование на таких языках требует значительных затрат времени и высокого профессионализма. Поскольку простейший файл, представленный на рис. 1.5, достаточно сильно отличается от физического способа хранения представленной в нем информации на носителе, программист должен быть знаком с физической структурой дисковых файлов, т. е. знать, как и где файлы хранятся на носителях компьютера. Поэтому программист должен определить маршрут доступа к данным каждого файла, на который имеются ссылки в программе. Такой маршрут рассчитывается по сложному алгоритму и позволяет установить точное местоположение различных компонентов файла и системы, а также свойства связанных с ними данных. По мере того, как система файлов становится более сложной, маршруты доступа все труднее поддаются контролю, и это может стать причиной системных сбоев.

Необходимость написания программ на языках 3GL для создания даже самых простых отчетов делает практически невозможной обработку нерегламентированных запросов. Принятых на работу DP-специалистов или DP-менеджеров, работающих со сложившейся системой файлов, нередко просят создать новые отчеты. Чаше всего они вынуждены давать ответ вроде "зайдите на следующей неделе" или даже "через месяц". Однако если информация вам требуется немедленно, то вполне вероятно, что "на следующей неделе" или в "следующем месяце" она не будет иметь никакого смысла.

По мере расширения системы файлов усложняется и системное администрирование. Каждый файл должен иметь собственную систему управления, состоящую из программ, дающих клиентам возможность выполнять следующие действия:

- ☐ создание структуры файла;
- ☐ добавление данных в файл;
- ☐ удаление данных из файла;
- ☐ изменение данных в файле;
- ☐ вывод содержимого файла.

Даже простая система, состоящая из 20 файлов, потребует $5 \times 20 = 100$ управляющих программ. Если к каждому из этих файлов осуществляется доступ из 10 различ-

ных программ, генерирующих отчеты, то необходимо написать дополнительно $20 \times 10 = 200$ программ. Поскольку нерегламентированные (в данном случае незапрограммированные) запросы невозможны, число программ генерации отчетов быстро умножается. А если каждый отдел организации является единоличным хозяином своих данных и создает собственные файлы, то общее число таких файлов растёт очень быстро.

Тщательное планирование структур файлов является очень важной обязанностью DB-менеджеров, поскольку изменение существующей структуры файла — дело слишком трудоемкое. Например, чтобы изменить только одно поле в исходном файле CUSTOMER, потребуется написать программу, которая должна выполнять следующие действия.

1. Поместить новую файловую структуру в специальный раздел памяти компьютера, называемый буфером.
2. Открыть исходный файл, используя другой буфер.
3. Считать запись из исходного файла.
4. Преобразовать исходные данные в форму новой структуры хранения.
5. Записать преобразованные данные в новую файловую структуру.

Затем исходный файл удаляется. Наконец, все программы, использующие файл CUSTOMER, должны быть настроены для использования новой структуры файла. Фактически любые изменения в структуре файла потребуют, как минимум, изменить все программы, использующие информацию из этого файла. Любые изменения являются источником ошибок и требуют дополнительных временных затрат на отладку. Поговорка "семь раз отмерь, один раз отрежь" особенно справедлива при работе с системой файлов, поскольку даже при малейших изменениях в файловой структуре приходится менять все программы доступа к данным.

В этой среде безопасность, например, защита данных надежным паролем, блокировка фрагментов файла или разделов самой системы и другие средства обеспечения защиты информации, трудно программируется и поэтому обычно не соблюдается. Если и предпринимаются попытки обеспечить безопасность данных и системы, то, как правило, в очень ограниченных пределах.

Структура файлов и недостаточная безопасность создают определенные трудности при накоплении данных. Такая организация обработки данных способствует их монопольному использованию, побуждая таким образом хранить одну и ту же информацию в различных местах (для такой разбросанной информации профессиональные разработчики БД используют термин "информационные островки" — islands of information, изолированные участки информации). Поскольку маловероятно, что данные, хранящиеся в разных местах, будут всегда согласованно обновляться, "островки информации" часто содержат различные версии одних и тех же данных. Результат использования системы файлов легко предсказать: хотя такой подход легко реализуется на начальных стадиях компьютерной обработки данных, по мере роста системы она, скорее всего, выйдет из-под контроля. Очевидно, что системы файлов не могут удовлетворять современным требованиям, предъявляемым к информационным системам.

1.3.2. Структурная зависимость и зависимость по данным

В предыдущем разделе мы обсудили, как изменение в любой файловой структуре (например, добавление или удаление поля) влечет за собой переделку всех программ, использующих этот файл. Такая модификация необходима, потому что система файлов обладает *структурной зависимостью (structural dependence)*, т. е. доступ к файлу зависит от его структуры.

Даже изменение свойств данных файла, например, изменение типа поля с *integer* (целое) на *decimal* (десятичное) потребует изменений во всех программах, использующих этот файл. Поскольку необходимо поменять все программы доступа к данным при любых изменениях характеристик данных файла, говорят, что система файлов обладает *зависимостью по данным (data dependence)*.

Практический смысл зависимости по данным состоит в разнице между *логическим форматом данных* (data logical format — как видит данные человек) и *физическим форматом данных* (data physical format — как “видит” данные компьютер). Поэтому всякая программа, получающая доступ к файлу, должна сказать компьютеру не только, *что* надо делать, но и *как* делать. Именно поэтому каждая программа должна содержать строки кода, в которых задаются способ открытия файлов конкретного типа, спецификация его записей и определения полей. Зависимость по данным, таким образом, делает систему файлов чрезвычайно громоздкой и с точки зрения программирования, и с точки зрения управления.

1.3.3. Определение полей и соглашение об именах

На первый взгляд кажется, что файл CUSTOMER, приведенный на рис. 1.3, прекрасно справляется с поставленными задачами: с его помощью можно легко создать необходимые отчеты. Но что если мы захотим создать телефонный справочник клиентов, используя данные, хранящиеся в этом файле? Использовать имена клиентов, хранящиеся в отдельных полях, неудобно, поскольку в справочнике содержимое этого поля необходимо разделить на фамилию, имя и отчество, причем в алфавитном порядке. А если нам потребуется получить список клиентов, отсортированный по телефонным кодам города? Включать код города в поле номера телефона не очень-то удобно.

И получить список клиентов, отсортированный по городам, тоже не так просто, как это может показаться. С точки зрения пользователя лучше (и удобнее), чтобы определения полей в записях соответствовали бы компонентам отчетной формы. Тогда поля файла CUSTOMER должны будут выглядеть следующим образом.

Таблица 1.2. Поля файла CUSTOMER

Поле	Содержимое	Пример заполнения
CUS_NAME	Фамилия клиента	Ramas
CUS_FNAME	Имя	Alfred

Таблица 1.2 (окончание)

Поле	Содержимое	Пример заполнения
CUS_INITIAL	Отчество (первая буква)	A
CUS_AREACODE	Код города	615
CUS_PHONE	Телефон	123-4567
CUS_ADDRESS	Номер дома и улица	218 Fork Rd.
CUS_CITY	Город	Babs
CUS_STATE	Штат	TN
CUS_ZIP	Почтовый индекс	36123

Выбор подходящих имен для полей имеет большое значение. В частности, нужно стараться давать полям содержательные имена. Из структуры файла, представленной на рис. 1.3, не сразу можно понять, что имя поля REN связано с датой окончания действия страхового полиса клиента. Использовать для этого поля имя CUS_RENEW_DATE гораздо лучше сразу по двум причинам. Во-первых, префикс CUS указывает на источник данных этого поля, которым в данном случае является файл CUSTOMER. Следовательно, мы знаем, что это поле относится к свойствам клиента (customer). Во-вторых, часть RENEW_DATE имени поля наглядно отражает его содержимое. При правильном выборе соглашения об именах файловая структура как бы описывает сама себя. В этом случае, просто посмотрев на имена полей, можно определить, какому файлу принадлежат данные поля и какого рода информацию они содержат.

В некоторых приложениях длина имени поля ограничивается, но все равно нужно стараться давать как можно более содержательные имена полям даже в рамках этих ограничений. К тому же необходимо помнить, что очень длинные имена трудно размещать на странице (экране), что создает проблемы при выводе. Например, имя поля CUSTOMER_INSURANCE_RENEWAL_DATE, несмотря на его предельную содержательность, все же менее предпочтительно, чем CUS_RENEW_DATE.

Другая проблема, которую можно заметить в файле, представленном на рис. 1.3, — невозможность эффективного поиска данных. Обратите внимание, что текущая версия файла CUSTOMER не имеет уникальных идентификаторов записей. Например, вполне возможно существование нескольких клиентов с именем John B. Smith. Поэтому наличие дополнительного поля CUS_ACCOUNT, в котором содержится уникальный номер счета клиента, было бы весьма полезным.

Хотя мы критиковали определения полей и соглашения об именах файловой структуры, представленной на рис. 1.3, тем не менее, проблема, которую мы только что обсудили, присуща не только системам файлов. Эти соглашения будут для нас важными и в последующих главах книги, поэтому убедитесь, что вы хорошо понимаете их. Мы снова столкнемся с определениями полей и соглашениями об именах при изучении проектирования баз данных в гл. 3, при обсуждении реализации базы данных в гл. 6, а также при изучении практического использования методов проектирования баз данных в гл. 7 и 8. В любой информационной среде, будь то система фай-

лов или база данных, проектирование должно всегда соответствовать как потребностям разработчика, так и требованиям, предъявляемым к отчетности и обработке данных конечными пользователями. Во всех случаях необходимо придерживаться надлежащего определения полей и соглашений об именах.

Примечание

Помните, что далеко не все соглашения об именах смогут удовлетворять требованиям любой системы. Некоторые слова или фразы резервируются для внутреннего использования СУБД. Например, в некоторых СУБД зарезервированным является слово ORDER. Подобным образом СУБД может интерпретировать дефис (-) как операцию вычитания, поэтому имя поля CUS-NAME может восприниматься как операция вычитания поля NAME из поля CUS. Поскольку таких полей не существует, будет выдано сообщение об ошибке. С другой стороны, имя CUS_NAME вполне приемлемо, поскольку в нем использован символ подчеркивания.

1.3.4. Избыточность данных

Если в файловой системе возникли трудности с обработкой данных, то весьма вероятно, что в различных местах хранятся одинаковые данные. Например, на рис. 1.3 и 1.4 имена агентов и номера телефонов встречаются как в файле CUSTOMER, так и в файле AGENT, хотя на самом деле достаточно иметь всего лишь один набор имен агентов и список их телефонных номеров. Если же они встречаются в нескольких местах, то имеет место *избыточность данных*.

Неконтролируемая избыточность может стать причиной возникновения следующих проблем.

- ❑ *Противоречивость данных (data inconsistency)*. О противоречивости данных говорят, когда в нескольких местах имеются различные противоречащие друг другу варианты одних и тех же данных. Предположим, мы изменили номер телефона агента или его адрес в файле AGENT. Если при этом мы забудем внести соответствующие изменения в файл CUSTOMER, то эти файлы будут содержать противоречивые данные по этому агенту. Созданные отчеты также будут противоречивы в зависимости от того, какой вариант данных был использован при их подготовке. Противоречивость данных тесно связана с недостаточным обеспечением *целостности данных (data integrity)*.

Ошибки оператора наиболее вероятны при вводе сложных элементов (например, десятизначных телефонных номеров) в несколько разных файлов и/или при часто повторяющемся вводе в один или более файл. На самом деле, файл CUSTOMER, представленный на рис. 1.3, содержит такую ошибку ввода: в третьей записи файла CUSTOMER переставлены цифры в номере телефона агента (615-882-2144 вместо 615-882-1244).

В файл CUSTOMER легко могут быть внесены несуществующие имена агентов и номера телефонов, и клиентам вряд ли понравится, если страховое агентство предоставит в их распоряжение имя и номер телефона несуществующего агента. К тому же должен ли отдел кадров выплачивать премиальные и предоставлять льготы несуществующим агентам? Фактически любая ошибка ввода данных, на-

пример, связанная с неправильно введенным именем или некорректным номером телефона, является причиной нарушения целостности данных.

□ *Аномалия данных.* Словарь определяет термин "аномалия" как отклонение, ненормальность. В идеальном случае изменение значения поля должно выполняться только в одном месте. Тем не менее, избыточность данных приводит к тому, что возникает необходимость производить такие изменения в множестве различных мест. Взгляните на файл CUSTOMER (см. рис. 1.3). Если агент Leath F. Hahn решит выйти замуж и сменить место жительства, то имя агента придется изменить, по всей видимости, адрес и номер телефона тоже. Вместо того чтобы внести такие изменения в единственный файл AGENT, мы должны исправить эту информацию во всех записях файла CUSTOMER, где встречается этот агент. Нам предстоит также проделать сотни исправлений в записях каждого клиента, связанного с этим агентом. Такая же проблема возникает при увольнении агента. Каждому клиенту, обслуживаемому уволившимся агентом, необходимо назначить нового. Обратите внимание, что аномалии данных существуют именно потому, что в целях обеспечения целостности данных любые изменения в каждом поле необходимо правильно выполнить во многих местах. Аномалии данных, которые имеются на рис. 1.3, в общем случае определяются так:

- *аномалии модификации.* Если у агента Leath F. Hahn меняется номер телефона, то новый номер должен быть введен в каждую запись файла CUSTOMER, в которой упоминается номер телефона госпожи Hahn. В нашем случае необходимо сделать только три исправления. В больших системах файлов такие исправления, возможно, придется делать в сотнях, а может, и тысячах записей. Очевидно, что при этом вероятность возникновения несовместимости в данных очень высока;
- *аномалии включения.* Чтобы добавить нового клиента в файл CUSTOMER, также необходимо ввести данные по агенту. Если мы добавляем несколько сотен новых клиентов, то нам придется ввести и несколько сотен имен агентов и не меньшее количество телефонных номеров. Опять-таки в этом случае высока вероятность несовместимости данных;
- *аномалии удаления.* Если агент Alex B. Alby удаляется из списка AGENT, то все его клиенты в файле CUSTOMER будут связаны с несуществующим агентом. Для разрешения этой проблемы мы должны изменить все записи, в которых встречаются имя господина Alby и его телефон.

1.4. Системы баз данных

Использовать систему базы данных, конечно же, намного привлекательнее, чем работать с системами файлов, где имеется столько проблем. В отличие от систем файлов с большим количеством не связанных друг с другом файлов, база данных состоит из логически взаимосвязанных данных, размещенных в едином хранилище. Поэтому БД предоставляет конечным пользователям иной способ управления данными, доступа к ним и хранения. Система управления базой данных, упомянутая в *разд. 1.1* и представленная на рис. 1.6, обладает целым рядом преимуществ по сравнению с системой управления файлами, представленной на рис. 1.5, и обеспечивает решение проблемы несовместимости данных, аномалии данных, зависимости по

данным и структурной зависимости. Более того, текущее поколение программного обеспечения СУБД позволяет централизованно хранить не только структуры данных, но и взаимосвязи таких структур, а также маршруты доступа к этим структурам. Современные СУБД также берут на себя заботу об определении, хранении и управлении всеми необходимыми маршрутами доступа к этим компонентам.

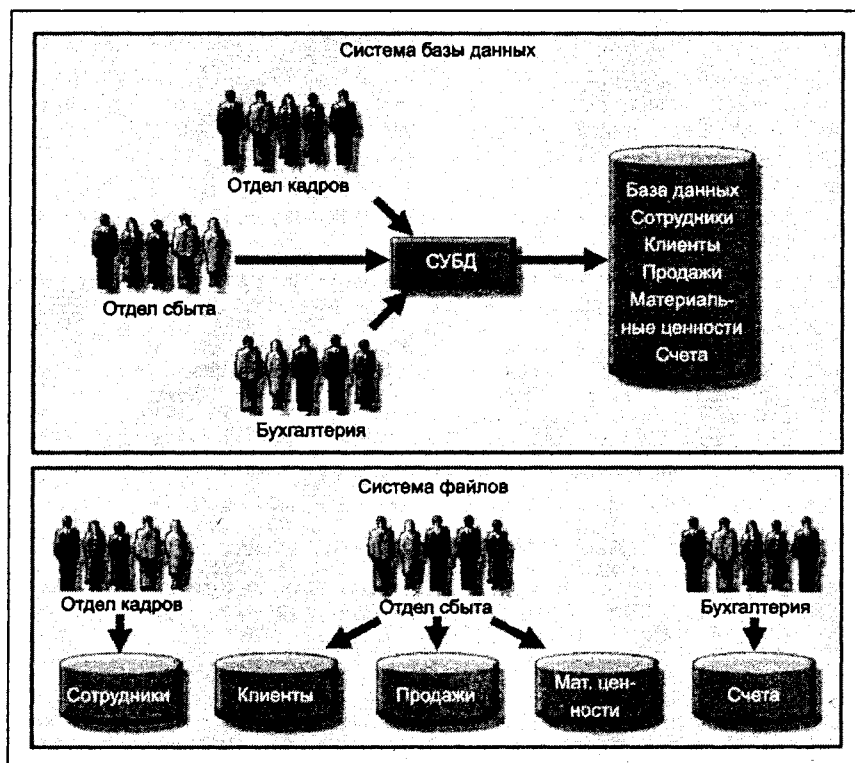


Рис. 1.6. Отличие базы данных от системы файлов

Необходимо помнить, что СУБД это всего лишь один из множества важнейших компонентов системы базы данных. Может быть, СУБД и можно считать "сердцем" системы базы данных, однако для обеспечения жизнедеятельности такого сложного организма одного сердца недостаточно. В разд. 1.4.1 мы рассмотрим, что представляет собой система базы данных, из каких компонентов она состоит и каким образом СУБД встраивается в общую картину системы базы данных.

1.4.1. Среда системы базы данных

Термин *система базы данных* относится к организации компонентов, определяющих и регулирующих сбор, хранение, управление и использование данных в среде базы данных. С точки зрения общего управления система базы данных состоит из пяти

основных частей, представленных на рис. 1.7: аппаратные средства, программное обеспечение, люди, процедуры и данные.

- **Оборудование.** Сюда относятся все системные аппаратные средства. Главный элемент оборудования системы базы данных, который проще всего идентифицировать, это компьютер, который может быть карманным компьютером, ноутбуком, персональным компьютером (ПК) или универсальным компьютером (mainframe). В состав оборудования также входит вся компьютерная периферия (периферийное оборудование), представляющая собой физические устройства, обслуживающие ввод/вывод. Другими словами, в состав периферийного оборудования входят клавиатура, мышь, модемы и принтеры. К периферии относятся и электронные устройства, используемые для подключения дополнительных компьютеров и организации сети. Сети компьютеров особенно важны для баз данных, хранящих информацию, доступ к которой необходимо обеспечивать с удаленных мест, например, в системах банкоматов и бронирования авиабилетов.

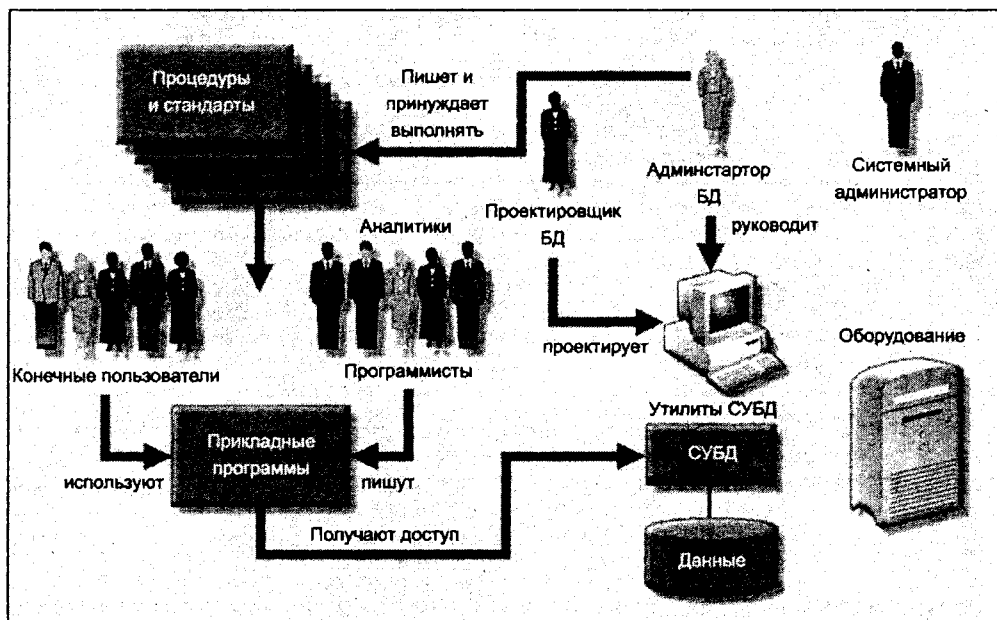


Рис. 1.7. Среда системы базы данных

- **Программное обеспечение.** Сюда входят комплекты компьютерных программ, используемых в системе базы данных. Хотя, как правило, под программным обеспечением понимают именно СУБД, тем не менее, для выполнения всех функций системы базы данных требуется программное обеспечение трех видов: системное программное обеспечение, программное обеспечение СУБД, а также прикладные программы и утилиты.
 - **Системное программное обеспечение** управляет всеми компонентами оборудования и обеспечивает доступ к нему всем другим приложениям, работающим

на компьютере. Примеры системного программного обеспечения: DOS, OS/2, операционные системы семейства Windows, устанавливаемые на персональных компьютерах, системы UNIX и VMS, используемые, как правило, на компьютерах средней мощности, и операционная система MVS, которая устанавливается на мэйнфреймах фирмы IBM.

- *Программное обеспечение СУБД* управляет базой данных. Примерами СУБД являются приложения Access и SQL Server корпорации Microsoft, Oracle корпорации Oracle и DB2 фирмы IBM.
- *Прикладные программы и утилиты* предназначены для получения доступа к данным и манипулирования ими в среде СУБД, а также в вычислительных средах, где необходимо выполнять такие действия. Прикладные программы (приложения) в большинстве случаев используются для получения доступа к данным, хранящимся в БД, для составления отчетов и таблиц, помогающих в принятии решений.

□ *Люди.* Сюда относятся все пользователи системы базы данных. Если взять за основу функциональные обязанности, то в системе базы данных можно выделить пять основных групп пользователей: системные администраторы, администраторы базы данных, проектировщики базы данных, системные аналитики/программисты и конечные пользователи. Члены каждой группы выполняют не только свои основные, но и различные дополнительные функции.

- *Системные администраторы* наблюдают за основными операциями системы.
- *Администраторы базы данных (DBA)* управляют работой СУБД и обеспечивают функционирование базы данных. Важнейшая роль, которую играют DBA, подробно рассматривается в гл. 16.
- *Проектировщики базы данных* проектируют структуру БД. Они, в сущности, являются архитекторами базы данных. Если БД спроектирована плохо, то даже лучшие прикладные программисты и самые просвещенные DBA не смогут обеспечить ее должное функционирование. Используя более древнюю аналогию, можно сказать, что даже лучшие каменщики и плотники не смогут построить хорошее здание по плохому проекту. Поскольку предприятия стремятся оптимизировать свои информационные ресурсы, должностные инструкции проектировщиков баз данных были (относительно недавно) в значительной степени переработаны с тем, чтобы расширить круг обязанностей проектировщиков и повысить их ответственность.
- *Системные аналитики и программисты* пишут и реализуют прикладные программы. Они проектируют и создают формы ввода данных, отчеты и процедуры, с помощью которых конечные пользователи получают доступ к данным и возможность манипулирования ими.
- *Конечные пользователи* это те, кто использует прикладные программы для выполнения ежедневных операций предприятия, например, продавцы, инспекторы, руководители и управляющие — все они считаются конечными пользователями. Конечные пользователи высшего уровня используют информацию, полученную из базы данных, для решения тактических и стратегических задач предприятия.

- ❑ *Процедуры.* Процедуры это инструкции и правила, которые управляют проектированием и использованием системы БД. Процедуры — очень важный, хотя зачастую незаслуженно забытый компонент системы. Они играют важнейшую роль на предприятии, поскольку устанавливают стандарты ведения коммерческой деятельности в рамках предприятия и с клиентами. Процедуры также используются для организации наблюдения и аудита как за вводимой в БД информацией, так и за информацией, порождаемой на основе этих данных.
- ❑ *Данные.* Под термином "данные", "информация" или "сведения" понимается весь фактический материал, хранящийся в базе данных. Данные (еще раз подчеркнем) являются необработанным "сырьем", которое подлежит соответствующему структурированию, а принятые решения о том, какую информацию необходимо поместить в БД и каким образом ее упорядочить, является важнейшей частью работы проектировщиков базы данных.

Очевидно, что база данных добавляет некое новое измерение в структуру управления предприятия. В целом эта административная структура зависит от размера предприятия, его предназначения и культуры его производства. Поэтому системы баз данных могут создаваться и функционировать на самых разных уровнях сложности и с весьма широким диапазоном правил и стандартов. Сравните, например, небольшую систему учета проката видеокассет с системой государственного страхования. Системой проката видеокассет могут управлять два человека; в состав ее оборудования входит, скорее всего, один персональный компьютер; процедуры, по всей вероятности, самые простые, а объем информации не так уж и велик. В системе государственного страхования, скорее всего, имеются, как минимум, один системный администратор, несколько штатных администраторов базы данных и несколько проектировщиков и программистов; в состав оборудования, по всей вероятности, входит несколько мэйнфреймов, расположенных в разных точках страны; процедур достаточно много, они сложны и тщательно проработаны, а объем обрабатываемой информации очень велик.

Кроме понимания того, что уровень сложности разрабатываемой системы базы данных определяется сферой деятельности предприятия и его масштабом, руководство предприятия должно принимать во внимание еще и тот факт, что разработка базы данных должна быть экономически выгодна и эффективна как с тактической, так и со стратегической точки зрения. Разработка системы стоимостью миллион долларов для решения копеечной проблемы едва ли может служить примером правильного выбора системы базы данных или верного проектного решения. Очевидно, что существующие технические решения на основе применения базы данных склоняют чашу весов в сторону использования таких систем. Учитывая высокий уровень требований, предъявляемых к базам данных и современным технологиям, нелишним будет обсудить несколько различных типов систем управления базами данных.

1.4.2. Типы систем управления базами данных

Системы управления базами данных (СУБД), составляющие основу систем баз данных, можно классифицировать по количеству пользователей, местоположению сайта БД, а также по ожидаемому способу и сфере использования.

По количеству пользователей СУБД можно подразделить на *однопользовательские* (single-user) и *многопользовательские* (multiuser). Однопользовательская СУБД обслуживает в данный момент времени только одного клиента. Другими словами, если пользователь А использует базу данных, то пользователи В и С должны подождать, пока пользователь А не закончит работу с базой. Если однопользовательская БД развернута на персональном компьютере, то ее называют *настольной базой данных* (*desktop database*). В противоположность этому, многопользовательская СУБД может обслуживать несколько пользователей одновременно. Если многопользовательская база данных обслуживает относительно небольшое число пользователей (менее 50) или, скажем, отдел предприятия, то она называется *базой данных рабочей группы* (*workgroup database*). Если же база данных используется в рамках всего предприятия и обслуживает большое число (более 50, как правило, сотни) пользователей нескольких подразделений и отделов, то такая БД называется *базой данных предприятия* (*enterprise database*).

Сайт, на котором размещена БД, тоже может служить признаком классификации СУБД. Например, СУБД, которая обслуживает базу данных, размещенную на одном сайте, называется *централизованной СУБД* (*centralized DBMS*). СУБД, обслуживающая базу данных, распределенную по нескольким различным сайтам, называется *распределенной СУБД* (*distributed DBMS*). Условия, при которых базу данных можно сделать распределенной, и способы управления такой базой будут детально обсуждаться в гл. 10.

Возможно, классификация СУБД по способу применения и сфере использования базы данных является самым распространенным и общепризнанным способом. Например, транзакции по продаже товаров или услуг, платежи и закупка товаров представляют собой важнейшие каждодневные операции. Время, которое отводится на выполнение таких транзакций, весьма ограничено, а результат их действий должен фиксироваться и вступать в силу немедленно. СУБД, которые управляют работой баз данных, спроектированных для транзакций "немедленного отклика", называются *транзакционными СУБД* (*transactional DBMS*) или *рабочими СУБД* (*production DBMS*)¹. В отличие от них *база данных поддержки решений* (decision support database) предназначена в основном для получения необходимой информации при выработке стратегических или тактических решений на уровне среднего и высшего руководства предприятия. Поддержка решений, обеспечиваемая *системой поддержки решений* (*decision support system, DSS*), как правило, требует широкомасштабной обработки данных (манипулирования данными) для извлечения полезной информации из данных, полученных за некоторый длительный промежуток времени, с тем, чтобы принять верное решение по ценовой политике, спрогнозировать сбыт, состояние рынка и т. д. Поскольку основная часть информации DSS извлекается из накопленных за некоторое время сведений, фактор времени, затраченного на получение данных в такой системе, не имеет столь существенного значения, как в транзакционных базах данных. К тому же информация систем DSS, как правило, базируется на большом объеме данных, полученных из различных источников. Чтобы облегчить поиск в этом море информации, структура базы данных DSS несколько отличается от структуры транзакционных БД. На самом деле, для обозначения баз данных,

¹ Сейчас такие СУБД чаще называют *операционными СУБД*. — Прим. пер.

предназначенных для систем DSS, используется термин *хранилище данных*, или *банк данных* (*Data Warehouse, DW*).

Совершенно ясно — хороший проект базы данных предполагает, что проектировщик БД точно знает, для чего предназначается проектируемая база. При проектировании транзакционных БД основное внимание уделяется целостности и непротиворечивости данных, а также скорости выполнения операций. При проектировании базы данных поддержки решений акцент делается на методах обработки данных, накопленных за длительное время. Проектирование баз данных, предназначенных для централизованного однопользовательского применения, требует иного подхода, нежели проектирование распределенных многопользовательских БД. В этой книге основное внимание будет уделено проектированию транзакционных централизованных однопользовательских и многопользовательских баз данных. Однако в *гл. 10* и *13* будут обсуждаться проблемы, стоящие перед проектировщиками распределенных баз данных и баз данных поддержки решений.

1.4.3. Предназначение СУБД

СУБД выполняет несколько очень важных функций, обеспечивающих целостность и непротиворечивость данных, хранящихся в БД. Большинство этих функций для конечного пользователя незаметны, и основную их часть выполняет СУБД. Сюда включаются управление словарем данных, управление хранением данных, преобразование данных и их представление, обеспечение безопасности, обслуживание многопользовательского доступа, резервное копирование и восстановление, целостность данных, языки доступа к данным и интерфейсы прикладного программирования, а также интерфейсы взаимодействия с базой данных.

- ❑ *Управление словарем данных.* Функционирование СУБД предусматривает, что определения элементов данных и их отношений (метаданные) хранятся в *словаре данных* (*data dictionary*). В свою очередь любые программы получают доступ к данным БД посредством СУБД. Для поиска необходимых структур данных и их отношений СУБД использует словарь данных, помогая избежать кодирования таких сложных взаимосвязей в каждой программе. Вдобавок любые изменения, которые делаются в структуре базы данных, автоматически регистрируются в словаре данных, что также освобождает нас от необходимости модифицировать программы доступа к изменившимся структурам данных. Другими словами, СУБД обеспечивает абстракцию данных, тем самым устраняя в системе структурную зависимость и зависимость по данным.
- ❑ *Управление хранением данных.* СУБД создает сложные структуры, необходимые для хранения данных, опять-таки освобождая нас от трудной задачи определения и программирования физических свойств данных. Современные СУБД обеспечивают хранение не только данных, но и связанных с данными экранных форм, схем отчетов, правил проверки данных, кода процедур, систем обработки видео, форматы изображений и т. д.
- ❑ *Преобразование и представление данных.* СУБД берет на себя задачу структурирования вводимых данных, преобразуя их в форму, удобную для хранения. Поэтому СУБД и здесь избавляет нас от рутинной работы преобразования логического формата данных в физический формат. Обеспечивая независимость данных,

СУБД преобразует логические запросы в команды, обеспечивающие определение физического местоположения необходимой информации и ее извлечение, т. е. СУБД форматирует физически полученные данные для придания им удобочитаемой формы. Иными словами, СУБД обеспечивает программную независимость и абстракцию данных.

- ❑ *Управление безопасностью.* СУБД создает систему безопасности, которая обеспечивает защиту пользователя и конфиденциальность данных внутри БД. Правила безопасности устанавливают, какие пользователи могут получить доступ к базе данных, к каким элементам данных пользователь может получить доступ, а также какие операции с данными (чтение, добавление, удаление или изменение) может выполнять пользователь. Это имеет особое значение в многопользовательских системах, где несколько пользователей могут одновременно получить доступ к данным. В гл. 16 будут подробно обсуждаться безопасность данных и проблемы конфиденциальности.
- ❑ *Управление многопользовательским доступом.* СУБД создает сложные структуры, обеспечивающие доступ к данным нескольким пользователям одновременно. Для того чтобы обеспечить целостность и непротиворечивость данных, в СУБД используются сложные алгоритмы, гарантирующие, что несколько пользователей могут получить одновременный доступ к базе данных без риска нарушить ее целостность. Проблемам, связанным с многопользовательским доступом, посвящена гл. 9.
- ❑ *Управление резервным копированием и восстановлением.* В СУБД имеются процедуры резервного копирования и восстановления данных, обеспечивающие их безопасность и целостность. Современные СУБД содержат специальные утилиты, с помощью которых администраторы базы данных могут выполнять регулярные и экстренные процедуры резервного копирования и восстановления данных. Восстановление данных производится после повреждения БД, например, в случае появления сбойного сектора на жестком диске или после аварийного отключения питания. Такая возможность необходима для обеспечения целостности данных. Проблемы резервного копирования и восстановления данных обсуждаются в гл. 16.
- ❑ *Управление целостностью данных.* В СУБД предусмотрены правила, обеспечивающие целостность данных, что позволяет минимизировать избыточность данных и гарантировать их непротиворечивость. Для обеспечения целостности данных используются их связи, которые хранятся в словаре данных. Целостность данных имеет особенно большое значение в транзакционных БД. Проблемы транзакций и целостности данных рассматриваются в гл. 9.
- ❑ *Языки доступа к данным и интерфейсы прикладного программирования.* СУБД обеспечивает доступ к данным с помощью языка запросов. *Язык запросов* это непроцедурный язык, т. е. он предоставляет пользователю возможность определять, что необходимо выполнить, без необходимости указывать, как это нужно выполнять. В состав языка запросов СУБД входят два основных компонента: *язык определения данных (data definition language, DDL)* и *язык манипулирования данными (data manipulation language, DML)*. DDL определяет структуры, в которых размещаются данные, а DML позволяет конечным пользователям извлекать данные из БД. СУБД также предоставляет программистам доступ к данным из процедурных

языков третьего поколения (3GL), таких как COBOL, C, PASCAL, Visual Basic и др. В составе СУБД имеются административные утилиты, предназначенные для администраторов базы данных (DBA) и проектировщиков БД, для внедрения, текущего контроля и обслуживания базы данных.

- **Интерфейсы взаимодействия с базой данных.** Текущее поколение СУБД обеспечивает специальные программы взаимодействия, разработанные для того, чтобы БД могла принимать запросы конечных пользователей в сетевом окружении. Фактически возможности взаимодействия с базой данных являются неотъемлемой составляющей современных СУБД. Например, СУБД предоставляет функции взаимодействия для получения доступа к базе данных, используя в качестве внешнего интерфейса интернет-браузер (такой как Netscape или Internet Explorer). В подобной среде взаимодействие может осуществляться несколькими способами:
- конечный пользователь может получать ответ на запросы, заполняя экранные формы с помощью выбранного им браузера;
 - с помощью СУБД можно автоматизировать публикацию форм отчетов в Интернете с помощью Web-форматирования, что позволяет просматривать их с помощью любого браузера;
 - с помощью СУБД можно подключаться к независимым системам для распространения информации по электронной почте (e-mail) или с помощью других эффективных приложений, например, Lotus Notes.

Интерфейсы взаимодействия с базой данных будут достаточно подробно исследованы в гл. 10 и 12, но главным образом — в гл. 14 и 15.

1.4.4. Управление системой базы данных: смещение акцентов

Как видите, СУБД выполняет основную часть рутинных процедур по обработке информации, которую когда-то в среде системы файлов должны были выполнять DP-специалисты, причем делает это куда искуснее и изощреннее. Более того, СУБД выполняет эту работу без утомительного и долгого программирования, которое было уделом систем файлов. Вдобавок, СУБД предоставляет рабочую среду, где используются строго определенные процедуры и стандарты. Поэтому теперь основой деятельности DP-специалистов или DP-менеджеров становится не утилитарное программирование, а решение проблем управления ресурсами данных предприятия и администрирование сложного программного обеспечения БД.

Поскольку бывшие DP-менеджеры систем файлов получили столь широкие функциональные обязанности в среде базы данных, их вполне можно выдвигать на роль системных администраторов. На первых порах, когда среда системы базы данных довольно проста, системные администраторы и выполняют задачи управления и получают практический опыт администрирования БД. По мере расширения базы данных системный администратор обычно назначает одного или двух администраторов базы данных, которые оценивают качество проекта БД, поддерживают программное обеспечение базы данных, присваивают полномочия отдельным пользователям на использование БД и координируют разработку приложений на основе имеющихся информационных ресурсов. Если информационные ресурсы требуют

использования более чем одной базы данных, системный администратор может назначить администраторов для каждой БД.

1.4.5. Проектирование базы данных и моделирование

Высокая продуктивность СУБД позволяет более изощренно эксплуатировать ресурсы данных в том случае, если мы хотим использовать все возможности разрабатываемой БД. Различные структуры данных, созданные внутри БД, и отношения между ними оказывают существенное влияние на эффективность СУБД. Поэтому проектирование играет ключевую роль в среде баз данных.

Использование моделей упрощает процесс проектирования баз данных. *Модели (models)* представляют собой упрощенные абстракции реальных событий и условий. Такие абстракции позволяют исследовать характеристики логических объектов (сущностей) и отношений, которые могут создаваться между такими объектами. Если модели не будут логически верными, то полученный на их основе проект БД не будет отвечать требованиям эффективности обработки информации. Поэтому в данной книге мы попытаемся разработать хорошую технику моделирования. Хорошее моделирование является залогом создания хорошо спроектированной БД, которая в свою очередь является основой для хороших приложений. И наоборот, вряд ли можно разработать хорошее приложение, если в нем используется неудачный проект БД, созданный на основе плохих моделей.

1.5. Модели баз данных

Поиск наилучшего управления данными выявил несколько способов разрешения ключевых проблем, имевшихся в системах файлов. Итогом исследований стала разработка различных моделей баз данных.

Модель базы данных это совокупность логических конструкций, используемых для представления структуры данных и отношений между ними внутри БД. Модели баз данных можно подразделить на две категории: *концептуальные модели* и *модели реализации*.

- ❑ *Концептуальная (понятийная) модель* нацелена на логическую природу представления данных. Поэтому в концептуальной модели основное внимание уделяется тому, *что* представлено в БД, а не *как* это представлено. К концептуальным моделям относятся модель "сущность-связь" (ER-модель), которая будет обсуждаться в гл. 3, и объектно-ориентированная модель, о которой мы расскажем в гл. 11.
- ❑ В отличие от концептуальной модели, *модель реализации* ставит во главу угла способ представления данных в БД или то, *как* реализовать структуры данных, чтобы получить представление о том, *что* мы моделируем. К моделям реализации БД относятся иерархическая модель, сетевая модель, реляционная модель и объектно-ориентированная модель.

Модели реализации мы кратко рассмотрим в *разд. 1.5.1 и 1.5.3*. Реляционная модель подробно рассматривается в гл. 2—8.

Разработчики баз данных берут концептуальную модель за основу для проектирования БД. В концептуальной модели используются три типа связей для описания отношений между данными: *один-ко-многим*, *многие-ко-многим* и *один-к-одному*. Проектировщики баз данных обычно используют для этого сокращенные обозначения — 1:M, M:N и 1:1 соответственно (хотя стандартным обозначением для связи "многие-ко-многим" является M:N, можно использовать и обозначение M:M). Следующие примеры показывают, в чем состоит разница между этими связями.

- ❑ *Один-ко-многим*. Художник может написать много различных картин, но каждая из них написана одним художником. Следовательно, художник ("один") связан с картинами ("многие"). Поэтому проектировщик баз данных обозначает связь "ХУДОЖНИК пишет КАРТИНУ" как 1:M. Точно так же в учетной записи клиента ("один") может содержаться множество счетов, но данные счета ("многие") относятся к единственному клиенту. Связь "КЛИЕНТУ выписан СЧЕТ" также относится к типу 1:M.
- ❑ *Многие-ко-многим*. Служащий может повышать квалификацию по нескольким направлениям, в то же время на занятиях по каждому направлению могут присутствовать многие служащие. Проектировщики баз данных обозначают связь "СЛУЖАЩИЕ повышают КВАЛИФИКАЦИЮ" как M:N. Так же студент может изучать несколько предметов, и на занятиях по каждому предмету может быть много студентов, поэтому связь "СУДЕНТЫ изучают ПРЕДМЕТ" следует обозначить как M:N.
- ❑ *Один-к-одному*. Структура отдела продаж предприятия может быть организована так, что каждым магазином заведует один сотрудник. В свою очередь каждый заведующий (естественно, являющийся сотрудником) управляет одним магазином. Поэтому связь "СОТРУДНИК заведует МАГАЗИНОМ" обозначается как 1:1.

Поскольку у каждой модели базы данных есть предшественники, в этой главе мы вкратце исследуем все модели. Опыт показывает, что изучение основных понятий структуры каждой модели данных позволит лучше понять методы проектирования и проблемы управления современных баз данных. На самом деле многие "новые" концепции баз данных имеют поразительное сходство с некоторыми "старыми" идеями и структурами. Например, можно обнаружить несомненное сходство между некоторыми концепциями иерархической модели базы данных и реализацией объектно-ориентированной структуры (см. гл. 11).

1.5.1. Иерархическая модель

Компания North American Rockwell была основным подрядчиком проекта Apollo, кульминацией которого стала высадка астронавтов на Луне в 1969 году. Для успешного завершения такого грандиозного проекта необходимо было обрабатывать информацию о миллионах деталей. Информация о деталях была организована как сложная компьютерная система файлов. Однако, как мы уже отмечали, использованные системы файлов ведут к избыточности хранимой информации, сопровождаемой соответствующими аномалиями данных.

Когда компания North American Rockwell приступила к разработке собственной системы базы данных, проверка компьютерных носителей информации показала, что избыточность данных достигала 60%. Проблемы, являвшиеся следствием избыточ-

ности данных, вынудили компанию выработать альтернативную стратегию управления таким огромным объемом информации. Взяв за основу уже имевшиеся в то время идеи баз данных, компания разработала программное обеспечение GUAM (Generalized Update Access Method — обобщенный метод доступа и модификации), заложив в его основу принцип агрегатности, когда множество небольших деталей составляют узел, который в свою очередь включается в узлы более высокого уровня (сборки), эти узлы входят в еще более сложные компоненты (агрегаты) и т. д., а все компоненты, собранные воедино, и образуют конструкцию в целом.

В середине 1960-х годов компания IBM объединилась с компанией North American Rockwell с целью дальнейшего развития системы GUAM и заменила носители информации на магнитных лентах более современными жесткими дисками, что позволило разработать сложную систему указателей.

Примечание

Указатель (pointer) это справочное устройство, которое точно "указывает" на местоположение определенных данных на устройстве хранения. Это напоминает глоссарий терминов в книге, который позволяет вам производить поиск среди терминов, расположенных в глоссарии по алфавиту, и затем по имеющемуся номеру страницы найти в книге необходимые сведения. Хотя компьютерная система указателей гораздо сложнее, принцип ее работы тот же.

Результат объединенных усилий Rockwell—IBM получил известность как Information Management System (IMS — информационно-управляющая система). С учетом растущего влияния компании IBM на рынок, IMS стала мировым лидером среди иерархических систем баз данных для универсальных машин в 70-х годах и в начале 80-х годов прошлого столетия.

Иерархическая модель базы данных основана на структуре, имеющей сходство с перевернутым деревом, где от ствола отходят ветви, от которых в свою очередь отходят другие ветви. Хотя иерархическая модель базы данных и не играет существенной роли на современном рынке БД, есть ряд причин, по которым стоит изучить ее основные свойства.

- ❑ Основные идеи этой модели БД лежат в основе разработки современных баз данных.
- ❑ Ограничения, имеющиеся в иерархической модели, позволяют по-новому взглянуть на проектирование БД.
- ❑ Некоторые важнейшие концепции этой модели применяются и в современных моделях БД.

Основные понятия

Базовая логическая структура иерархической модели станет понятнее, если провести аналогию с неким производственным процессом. Рассмотрим упрощенную процедуру создания картотеки.

1. Картотечный шкаф состоит из нескольких узлов (components): каркас, набор выдвижных ящиков и направляющие для этих ящиков.

2. Каждый узел может состоять из множества более мелких сборок (assembly). Например, у каждого ящика имеются ручка с защелкой, несколько роликов, с помощью которых он перемещается по направляющим, и разделительные пластины.
3. Каждая сборка может состоять из нескольких деталей (parts). К примеру, каждый ролик состоит из небольшого колесика, оси и крепежной скобы.
4. Процесс производства картотечного шкафа состоит в сборке всех частей, взаимосвязи между которыми постоянны во времени. Когда бы ни был собран данный картотечный шкаф, сегодня или завтра, одни и те же детали собираются вместе тем же способом, образуя сборку, сборки собираются в узел, а все узлы собираются воедино, образуя тем самым картотечный шкаф.

Описание формирования деталей, сборок и узлов, которое мы только что привели, помогает представить весь этот логический процесс в виде перевернутого "дерева", которое называют *иерархической структурой* (рис. 1.8). На рисунке каждый компонент структуры обозначен соответствующим образом, чтобы помочь разобраться в основной терминологии иерархической БД.

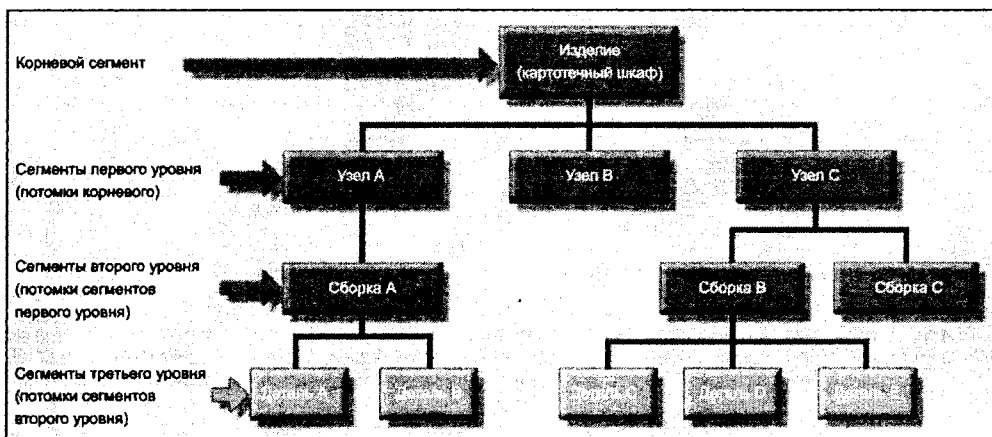


Рис. 1.8. Иерархическая структура

Если внимательно посмотреть на рис. 1.8, то можно заметить, что с точки зрения пользователя иерархическая база данных представляет собой иерархию сегментов. *Сегмент* (segment) эквивалентен записи в системе файлов. Другими словами, иерархическая база данных это совокупность записей, которые логически организованы с целью отображения структуры перевернутого дерева (иерархической структуры), представленной на рис. 1.8. Внутри этой иерархии верхний слой (*корень*, root) воспринимается как *предок* (parent) сегмента, находящегося непосредственно под ним. Например, на рис. 1.8 корневой сегмент является предком сегментов уровня 1, которые в свою очередь являются предками сегментов уровня 2 и т. д. Сегменты, расположенные под другим сегментом, являются *потомками* (дочерними сегментами, child) сегмента, находящегося над ними.

Вкратце можно сказать, что:

- ☐ каждый предок может иметь несколько потомков;
- ☐ каждый потомок имеет только одного предка.

В такой иерархической структуре достаточно просто проследить и компоненты базы данных, и существующие между ними связи типа 1:М.

Необходимо иметь в виду, что древовидная структура, представленная на рис. 1.8, не может быть непосредственно воспроизведена в компьютерном устройстве хранения. Компьютер не может, как человек, увидеть логическую структуру. Вместо этого дерево определяется цепочкой, отображающей маршрут от сегмента-предка к сегменту-потомку, начиная слева. Эта упорядоченная последовательность сегментов, отображающая иерархическую структуру, называется *иерархическим маршрутом (hierarchical path)*. Например, иерархический маршрут к сегменту с именем "Деталь D" на рис. 1.8 может быть представлен следующим образом:

Изделие → Узел A → Сборка A → Деталь A → Деталь B → Узел B
Узел C → Сборка B → Деталь C → Деталь D

Обратите внимание, что маршрут обходит все сегменты, включая корневой, начиная с самого левого. Этот "левосторонний" список называется *прямым порядком обхода (preorder traversal)* или *иерархической последовательностью (hierarchic sequence)*. Проектировщики баз данных должны убедиться, что сегменты и их элементы в этой цепочке, доступ к которым происходит чаще всего, расположены ближе к левому краю дерева с тем, чтобы обеспечить более эффективную обработку данных. На рис. 1.8, например, если Part D (Деталь D) используется чаще всего, то было бы разумнее изменить структуру базы данных так, чтобы поместить Component C (Узел C) в левую часть уровня 1. Затем, внутри этой перемещенной "ветви" нужно переставить Part D (Деталь D) на место, которое до этого занимала Part C (Деталь C). Эти изменения сделают иерархическую последовательность более короткой.

Изделие → Узел C → Сборка B → Деталь D

В этой связи у вас, тем не менее, не должно создаться впечатление, что иерархическая модель БД ограничивается рамками производственных процессов.

Иерархическая модель эффективна в независимости от количества транзакций, в которые вовлечены остающиеся постоянными во времени связи 1:М. Например, система учета банковских счетов хорошо укладывается в иерархическую модель:

- ☐ банковский счет каждого клиента может быть предметом множества операций (связь "один-ко-многим");
- ☐ операции над счетом могут совершаться или по дебиту, или по кредиту. Поэтому основные связи между банковским счетом клиента и операциями, которые по нему совершаются, фиксированны;
- ☐ банк, как правило, имеет множество счетов клиентов, и каждый такой счет может стать предметом множества операций (транзакций). Поэтому общее число транзакций может, по всей вероятности, быть очень большим.

Не стоит удивляться тому, что многие банки выбирают именно иерархическую модель БД.

Преимущества

У иерархической модели базы данных есть множество преимуществ по сравнению с системой файлов. На самом деле большинство преимуществ, которыми обладает иерархическая модель базы данных, помогли сформировать основу для создания современных моделей баз данных, куда многие из этих достоинств были скопированы, хотя и в несколько измененном виде. Наиболее важные достоинства иерархической модели перечислены в следующем списке.

- ❑ *Простота идеи.* Структура иерархической модели, отношения между различными слоями интуитивно понятны. Поэтому достаточно легко мысленно представить себе всю базу данных, что упрощает ее проектирование.
- ❑ *Безопасность базы данных.* Безопасность базы данных обеспечивает СУБД. Поэтому безопасность едина для всей системы и не требует никаких усилий от программистов, у которых могут быть различные взгляды на способы защиты.
- ❑ *Независимость данных.* СУБД создает среду, которая может обеспечить независимость данных, существенно упрощая этим работу программистов (независимость данных имеет место, если изменение типа данных вызывает его автоматическое изменение с помощью СУБД во всей базе данных, исключая таким образом, необходимость модифицировать участки программ, которые используют эти данные).
- ❑ *Целостность данных.* Взаимоотношение предок/потомок всегда предполагает наличие связи между сегментом-предком и его дочерними сегментами (потомками). Поскольку дочерний сегмент всегда автоматически связан со своим предком, иерархическая модель тем самым всегда обеспечивает целостность БД.
- ❑ *Эффективность.* Иерархическая модель базы данных очень эффективна, когда в БД содержится большой объем данных со связью 1:М и когда пользователи выполняют большое число транзакций, используя объекты, связи между которыми фиксированы во времени.

Из-за явного превосходства над системой файлов иерархические БД быстро стали доминировать на рынке в 1970-х годах. Это стало предпосылкой создания больших БД (на мэйнфреймах), что в свою очередь послужило причиной появления целого поколения программистов, знающих такие системы и разработавших большое число хорошо зарекомендовавших себя бизнес-приложений.

Недостатки

Хотя в настоящее время иерархические базы данных и их приложения еще существуют, все же иерархическая модель стала быстро терять популярность в конце 1970-х и начале 1980-х годов по нескольким важным причинам.

- ❑ *Сложность реализации.* Несмотря на то что СУБД иерархической модели упрощает работу проектировщиков и программистов при решении проблем обработки данных, от них все же требуется высокий профессионализм при организации физического хранения данных. Поэтому задача реализации проекта БД может оказаться достаточно сложной.
- ❑ *Сложность управления.* Любые изменения в структуре БД, например, перемещение сегментов, вызовут необходимость изменений во всех прикладных программах, получающих доступ к базе данных. Следовательно, управление базой дан-

ных может стать трудной задачей. И хотя иерархическая структура стимулирует целостность базы данных, в то же время она дает возможность удалить один сегмент, что приведет к невольному удалению всех сегментов под ним — ошибка, которая может дорого стоить!

- ❑ *Недостаток структурной независимости.* Структурная независимость имеет место, если изменения в структуре базы данных не влияют на возможность доступа СУБД к данным. Иерархическую базу данных еще называют *навигационной системой*, поскольку доступ к данным предполагает, что для "навигации" по заданным сегментам используется маршрут физического хранения. Внутри навигационной системы базы данных программист должен знать такой маршрут к соответствующим сегментам (для того, чтобы получить доступ к дочерней записи, прежде необходимо обеспечить доступ к предку), чтобы извлекать данные из БД. Изменения в структуре БД могут привести к проблемам с прикладными программами, которые до этого работали правильно; за преимущество независимости по данным приходится расплачиваться структурной зависимостью.
- ❑ *Сложность программирования и использования приложений.* В структуре навигационной базы данных прикладные программисты и конечные пользователи должны точно знать, каким образом физически данные размещены в БД для того, чтобы получить к ним доступ. Даже если им известен маршрут доступа к данным, получение данных требует знания сложной системы указателей. Поэтому говорят, что иерархические базы данных были созданы программистами для программистов.
- ❑ *Ограничение в реализации.* Многие связи не могут быть изображены схемой 1:M, на которой основана иерархическая БД. Например, рассмотрим список студентов университета. На каждую дисциплину может записаться много студентов, и каждый студент может проходить обучение по нескольким дисциплинам. Такую связь "многие-ко-многим" (M:N) трудно реализовать в иерархической модели. К тому же в действительности имеется достаточно много связей, основанных на потомке, имеющем несколько предков. Например, из рис. 1.9 следует, что сегмент Строка заказа имеет двух предков — Заказ и Деталь. Когда клиент делает заказ, строка заказов содержит данные как из сегмента Заказ, так и из сегмента Деталь:
 - Order: 10012 Customer 89221 Part number: M-2W113
 - Подобную схему, при которой потомок имеет двух предков, невозможно легко реализовать в иерархической модели.
- ❑ *Недостаток стандартизации.* Хотя основные концепции иерархической модели используются повсюду в программном обеспечении БД, все-таки не существует стандартизированного набора *ключевых понятий* и компонентов, на базе чего можно было создать модель, в которую было бы включено описание всех необходимых стандартов. Проблемы с реализацией БД вызывали особенное раздражение, поскольку компоненту администрирования базы данных явно недоставало и стандартного языка определения данных (data definition language, DDL) для определения компонентов БД и языка манипулирования данными (data manipulation language, DML) для работы с содержимым БД. Несмотря на то что программное обеспечение IMS, разработанное совместно компаниями IBM и North American Rockwell, фактически стало играть роль СУБД в конце 60-х и начале 70-х годов прошлого века, параллельно, вне рамок концепций и терминологии,

определенных в IMS, использовалось множество менее заметных программ. Поэтому переход с одной иерархической БД на другую был весьма трудной задачей: *переносимость* таких баз данных была ограничена.

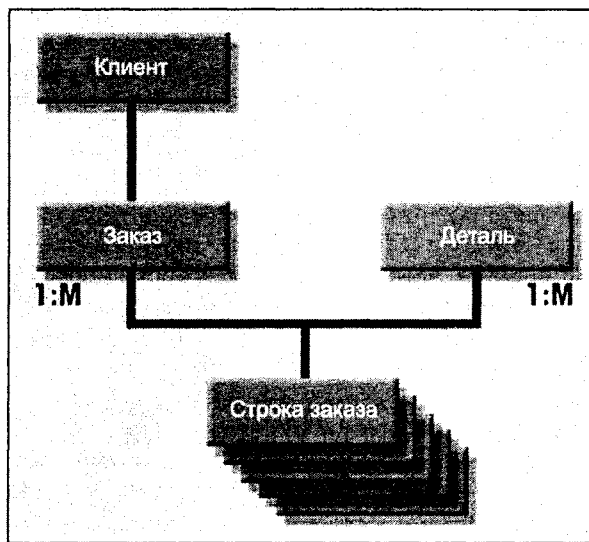


Рис. 1.9. Потомок с несколькими предками

В 1970 году профессиональные разработчики баз данных провели серию конференций, кульминацией которых стала публикация комплекта стандартов для баз данных, что, в конечном счете, привело к разработке альтернативных моделей БД. Наиболее заметной из этих моделей стала сетевая модель базы данных.

1.5.2. Сетевая модель

Сетевая модель базы данных была разработана для того, чтобы более эффективно представлять сложные отношения данных, чем это можно было сделать в иерархической модели, а также для улучшения производительности БД и для стандартизации баз данных. Недостаток стандартизации вызывал нарекания программистов и прикладных разработчиков, поскольку это создавало проблемы с переносимостью приложений. Более того, отсутствие стандартных понятий и компонентов мешало поиску наилучших моделей БД. Беспорядок редко способствует прогрессу.

Чтобы разработать стандарты для баз данных, в 1971 году была созвана конференция Conference on Data Systems Languages (CODASYL — постоянно действующая конференция по языкам обработки данных). Группа CODASYL изначально была сформирована пользователями и производителями компьютеров для разработки стандарта COBOL. Рекомендации COBOL CODASYL были впоследствии приняты институтом American National Standards Institute (ANSI, Национальный Институт Стандартиза-

ции США) в качестве стандарта на спецификацию COBOL, что привело к появлению стандарта ANSI COBOL. После успешной разработки стандарта COBOL группа CODASYL вознамерилась сделать то же самое в области баз данных, для чего была создана группа Database Task Group (DBTG, рабочая группа по базам данных). Группа DBTG должна была разработать стандартное описание среды, облегчающее создание БД и манипулирование данными.

В заключительном отчете DBTG описаны три ключевых компонента базы данных.

- *Схема*, абстрактная организация базы данных с точки зрения администратора БД. В эту схему были включены определения имени базы данных, типа каждой записи и компонентов, которые заполняли эти записи.
- *Подсхема*, определяющая фрагмент базы данных, "видимый" прикладными программами, которые фактически извлекают необходимую информацию из базы данных. Определения подсхемы позволяют всем прикладным программам упростить вызов подсхемы, необходимой для доступа к соответствующему файлу (файлам) БД.
- *Язык управления данными* (data management language), предназначенный для определения свойств данных и их структуры, чтобы можно было манипулировать данными.

Чтобы выработать необходимый стандарт для каждого из этих трех компонентов, группой DBTG были также определены три отдельных компонента языка управления данными.

- *Схема языка определения данных* (DDL, Data Definition Language), которая позволяет администратору определять компоненты схемы.
- *Подсхема DDL*, дающая возможность прикладным программистам определить компоненты БД, которые будут использоваться.
- *Язык манипулирования данными* (DML, data manipulation language) для работы с содержимым БД.

Комитет ANSI Standards Planning and Requirements Committee (SPARC) в 1975 году расширил стандарты БД. Ожидалось, что все программное обеспечение сетевой БД для мэйнфреймов будет приведено в соответствие со стандартами отчета DBTG. Поэтому, выполняя все требования стандартов, проектировщики и пользователи сталкивались бы со значительно меньшим количеством проблем при переходе от одного коммерческого приложения к другому, оперируя на абстрактном уровне или на уровне схемы. К сожалению, стандарты зачастую "растягивались" для того, чтобы подогнать их под возрастающие требования поставщиков программного обеспечения.

Основные понятия

Во многих отношениях сетевая модель базы данных сходна с иерархической моделью. Например, как и в случае иерархической модели, пользователь воспринимает БД как совокупность записей, между которыми существует связь 1:М. Однако в отличие от иерархической модели, записи сетевой модели могут иметь более чем одного предка. Поэтому такие часто встречающиеся связи, представленные на рис. 1.9, легко описываются средствами сетевой модели.

По терминологии сетевых БД связь называется *множеством* (set). Каждое множество состоит как минимум из двух типов записей: *запись-владелец* (owner), которая в иерархической модели соответствует предку, и *запись-член* (member), которая соответствует потомку в иерархической модели. Различие между иерархической и сетевой моделями состоит в том, что в последней, при некоторых условиях, запись (как запись-член) может появляться более чем в одном множестве. Другими словами, запись-член может иметь несколько записей-владельцев. Множество представляет связь 1:M между записью-владельцем и записью-членом. Пример такой связи представлен на рис. 1.10.

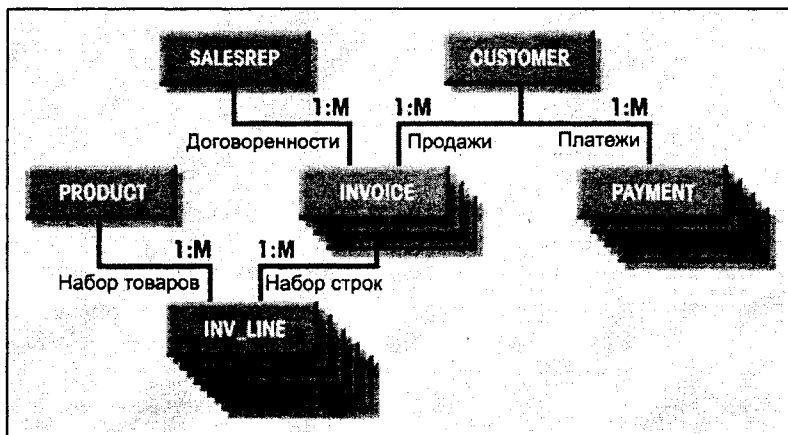


Рис. 1.10. Сетевая модель базы данных

Рис. 1.10 иллюстрирует пример сетевой модели базы данных типичного торгового предприятия. В этой модели CUSTOMER (клиент), SALESREP (SALES REPresentative, торговый агент), INVOICE (счет), INV_LINE (INvoice LINE, строка счета), PRODUCT (товар) и PAYMENT (оплата) представляют собой типы записей. Обратите внимание, что на рис. 1.10 записью INVOICE "владеют" и SALESREP, и CUSTOMER. Подобным образом запись INV_LINE имеет двух владельцев PRODUCT и INVOICE. Обратите внимание, что сетевая модель может содержать связи с одним владельцем (например, клиент — CUSTOMER — производит оплату — PAYMENT), характерные для иерархических моделей.

Сетевая модель базы данных, представленная на рис. 1.10, иллюстрирует транзакции, основанные на сериях связей "один-ко-многим".

- ❑ Агент может выписать множество бланков счетов, но каждый бланк выписан единственным агентом. Поэтому агенты SALESREP и счета INVOICE связаны как 1:M.
- ❑ Клиент может делать покупки в различных обстоятельствах. При каждой покупке выписывается счет (INVOICE), т. е. клиент (CUSTOMER) может создать за какое-то время множество счетов, но каждый счет (INVOICE) принадлежит единственному клиенту (CUSTOMER). Поэтому между клиентом (CUSTOMER) и счетом (INVOICE) существует связь 1:M.

- Каждый заказанный товар занимает соответствующую строку в счете. Поскольку клиент может за один раз купить несколько товаров (более одного) — например, он может во время посещения магазина приобрести отвертку, коробку шурупов и банку краски — в каждом счете может быть одна или более строк. Но каждая строка (INV_LINE) конкретного счета (INVOICE) "принадлежит" этому счету.
- Каждая строка счета ссылается на единственный товар. Но поскольку магазин может продать множество отверток за какой-то период времени, этот товар может появиться во многих разных счетах. Поэтому между товаром (PRODUCT) и строкой счета (INV_LINE) существует связь 1:M.
- Клиент может сделать несколько платежей за некоторый период времени. Но только один клиент делает данный платеж. Поэтому между клиентом (CUSTOMER) и оплатой (PAYMENT) имеется связь 1:M.

Преимущества

В сетевой модели сохранились основные преимущества иерархической модели, в то же время в ней были исправлены многие недостатки, присущие иерархической модели.

- *Концептуальная простота.* Как и в иерархической модели, абстрактное представление базы данных является достаточно простым, что упрощает проектирование.
- *Поддержка других типов связей.* Связь M:N проще реализуется в сетевой модели, чем в иерархической. К примеру, сетевая база данных обрабатывает связь "данный товар может быть продан множеству клиентов, а данный клиент может сделать множество покупок" проще, поскольку в ней допускается наличие нескольких "владельцев" записи (см. рис. 1.10).
- *Гибкий доступ к данным.* Гибкость доступа к данным в сетевой модели значительно выше, чем в иерархической модели и в системах файлов. Любое приложение может получить доступ к записи-владельцу и всем записям-членам внутри множества. Поэтому, если запись-член имеет двух или более владельцев, как это показано на рис. 1.10, можно напрямую перейти от одного владельца к другому (иначе говоря, в этом случае не потребуется долгий прямой обход).
- *Обеспечение целостности базы данных.* Сетевая модель способствует соблюдению целостности базы данных, поскольку пользователь должен, прежде всего, определить запись-владельца, а уже потом записи-члены (запись-член не может существовать без записи-владельца).
- *Независимость данных.* В сетевой модели обеспечена достаточная независимость данных, которая, по крайней мере, частично избавляет от программирования сложных деталей, связанных с методами физического хранения информации. Поэтому изменения в свойствах данных не потребуют переделки тех участков прикладных программ, где выполняется доступ к данным.
- *Соответствие стандартам.* Возникновение сетевой модели, по крайней мере, в некоторой части, связано со стандартами, разработанными и принятыми в семидесятых годах двадцатого века. Эти стандарты, включая DDL и DML, значительно улучшили возможности администрирования баз данных, а также их переносимость.

Недостатки

Несмотря на то что сетевая модель базы данных имеет значительные преимущества перед иерархической моделью, она все же обладает и целым рядом недостатков.

- ❑ *Сложность системы в целом.* Обеспечение целостности и эффективность, с которой сетевая БД управляет отношениями, иногда становятся причиной сложности всей системы. Так же как и иерархическая, сетевая модель предоставляет навигационные средства доступа к данным, когда доступ к записи осуществляется за один раз. Поэтому администраторы БД, программисты и конечные пользователи для получения доступа к данным должны хорошо знать внутреннюю структуру базы. Говоря кратко, сетевую модель базы данных, как и иерархическую, нельзя считать дружественной системой.
- ❑ *Недостаточная структурная независимость.* Поскольку сетевая модель базы данных также обеспечивает доступ к данным с помощью средств навигации, то в ней, как и в иерархической модели, трудно производить структурные изменения, а некоторые из них просто невозможны. Если в структуре БД делаются какие-то изменения, то придется проверить все прикладные программы, перед тем как разрешить им доступ к БД. В конечном счете, хотя в сетевой модели и достигается независимость по данным, все же она не обеспечивает структурной независимости.

Вследствие имеющихся серьезных недостатков в 1980-х годах сетевая модель была вытеснена реляционной моделью.

1.5.3. Реляционная модель

Поскольку разработчикам приходилось вникать в тонкости структуры базы данных, проектирование баз данных было очень трудным делом. Фактически, несмотря на свои сильные стороны, сложная структура сетевых БД позволяла лишь немногим пользователям и проектировщикам использовать их с максимальной эффективностью. По мере роста объема обрабатываемой информации и повышения требований к базам данных и приложениям, проектирование, управление и использование БД становились все более трудоемкими.

Отсутствие возможности обработки нерегламентированных запросов заставляло программистов писать код даже для выпуска простейших отчетов. И хотя существующие БД обеспечивали ограниченную независимость по данным, любые структурные изменения в базе данных заставляли полностью переделывать все прикладные программы, получающие информацию из БД. Ветераны баз данных помнят бесконечные задержки в получении информации, возникающие в иерархических и сетевых средах.

Реляционная модель, впервые предложенная Э. Коддом (E. F. Codd, компания IBM), стала настоящим прорывом как для пользователей, так и для проектировщиков. Образно говоря, реляционная модель это "база данных с автоматической коробкой передач", в отличие от "баз данных с ручной коробкой передач", где использовались иерархические и сетевые модели. Концептуальная простота реляционной модели подготовила почву для подлинной революции в сфере баз данных.

В 70-х годах прошлого века работа Э. Кодда считалась весьма остроумной, но абсолютно непрактичной. Внешняя концептуальная простота реляционной модели достигается за счет ресурсов компьютера, а компьютеры в то время не обладали достаточной мощностью для реализации реляционной модели. К счастью, возможности компьютеров стремительно росли вместе с ростом эффективности операционных систем. К тому же стоимость вычислительной техники снижалась даже быстрее, чем возрастали ее возможности. Сегодня даже настольные компьютеры, стоимость которых несоизмеримо ниже, чем у их предков — больших универсальных машин, могут работать с очень сложным программным обеспечением реляционных баз данных типа Informix, Oracle, Ingress, DB2 и пр.

Примечание

Сведения о реляционных базах данных, представленные в этой главе, лишь предваряют их очень подробное обсуждение в последующих главах. Фактически реляционная модель настолько важна, что она составляет основу всех наших рассуждений в большинстве остальных глав книги.

Основные понятия

Реляционная модель базы данных реализуется с помощью сложнейшей системы управления реляционной базой данных (реляционная СУБД — РСУБД, RDBMS — Relational Database Management System). Кроме того, что РСУБД выполняет те же основные действия, что и СУБД в иерархической и сетевой моделях, в ее состав входят и дополнительные функции, делающие реляционную модель простой для понимания и внедрения.

Возможно, наиболее важным преимуществом РСУБД является предоставленная пользователям и проектировщикам возможность оперировать обычными понятиями человеческой логики. РСУБД берет на себя управление всеми сложными деталями физической реализации БД. Поэтому реляционная база данных представляется пользователю в виде набора таблиц, в которых хранятся данные.

Каждая *таблица* (*table*) представляет собой матрицу, содержащую набор пересекающихся строк и столбцов (*row/columns*). Таблицы, которые также называются *отношениями* (*relation*), связаны друг с другом через общие свойства объекта. Например, таблица CUSTOMER (клиенты), представленная на рис. 1.11, может содержать номера торговых агентов, которые содержатся и в таблице AGENT (агенты).

Общая связь между таблицами CUSTOMER и AGENT дает возможность обеспечить соответствие клиента и его агента, несмотря на то, что данные по клиенту хранятся в одной таблице, а данные о торговом агенте — в другой. Например, мы легко можем определить, что агентом клиента Dunne является Alex Alby, потому что значение AGENT_CODE таблицы CUSTOMER для клиента Dunne равно 501, что соответствует значению AGENT_CODE для агента Alex Alby в таблице AGENT. Хотя таблицы полностью независимы одна от другой, мы можем очень просто связать данные между таблицами. Реляционная модель, таким образом, обеспечивает минимальный уровень контроля избыточности данных, устраняя избыточность, имеющую место в системах файлов.

Таблица AGENT

AGENT_CODE	AGENT_LNAME	AGENT_FNAME	AGENT_INITIAL	AGENT_AREACODE	AGENT_PHONE
501	Alby	Alex	B	713	228-1249
502	Hahn	Leah	F	615	882-1244
503	Okon	John	T	615	123-5589

Связь через AGENT_CODE

Таблица CUSTOMER

CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_RENEW_DATE	AGENT_CODE
10010	Ramas	Alfred	A	615	844-2573	05-Apr-2002	502
10011	Dunne	Leona	K	713	894-1238	16-Jun-2002	501
10012	Smith	Kathy	W	615	894-2285	29-Jan-2001	502
10013	Olowski	Paul	F	615	894-2180	14-Oct-2002	502
10014	Orlando	Myron		615	222-1672	28-Dec-2002	501
10015	O'Brian	Amy	B	713	442-3381	22-Sep-2002	503
10016	Brown	James	G	615	297-1228	25-Mar-2002	502
10017	Williams	George		615	290-2556	17-Jul-2002	503
10018	Farriss	Anne	G	713	382-7185	03-Dec-2002	501
10019	Smith	Olette	K	615	297-3809	14-Mar-2002	503

Рис. 1.11. Связь реляционных таблиц

Связывая таблицы AGENT и CUSTOMER через AGENT_CODE, мы можем определить, что агент Leah Hahn (AGENT_CODE = 502) работает с клиентами Ramas (10010), Smith (10012), Olowski (10013) и Brown (10016). Мы можем также определить, что агентом клиента Amy O'Brian является John Okon и т. д.

Тип связи (1:1, 1:M или M:N) часто отображается в *реляционной схеме* (схеме отношений), пример которой приведен на рис. 1.12. В реляционной схеме показываются связанные поля (в данном случае AGENT_CODE) и тип связи. При создании рис. 1.12 использовалась СУБД Microsoft Access, в которой символ "?" обозначает "много". В этом примере таблица CUSTOMER представляет собой "многих", поскольку у агента (AGENT) может быть несколько клиентов (CUSTOMER). Со стороны таблицы AGENT связь обозначена как "1", поскольку с каждым клиентом (CUSTOMER) работает только один агент (AGENT).

В реляционной таблице хранятся наборы связанных логических объектов (сущностей, entities). В этом отношении таблица реляционной базы данных имеет сходство с файлом. Но есть одно очень важное отличие таблицы от файла: таблица обеспечивает полную независимость по данным, а также структурную независимость, поскольку является исключительно логической структурой. Пользователя и проектировщика совершенно не касается, каким образом данные физически хранятся в базе данных; здесь принимается в расчет только их собственное представление о структуре данных. И это свойство реляционной модели, которое будет подробнее рассмотрено в следующей главе, стало источником настоящей революции в базах данных.

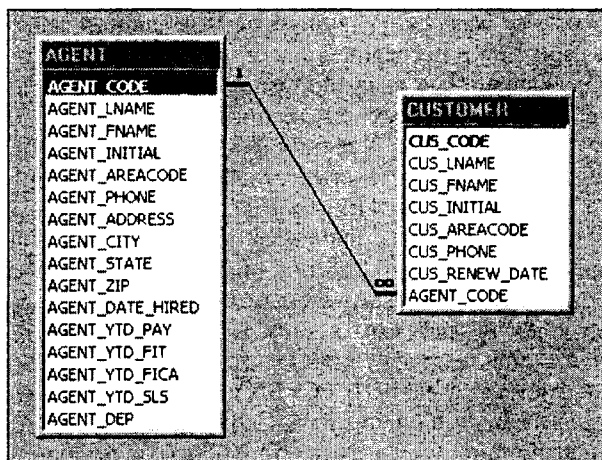


Рис. 1.12. Реляционная схема

Преимущества

Так же как иерархические и сетевые базы данных, реляционная БД является единым хранилищем данных, обеспечивающим их независимость. Однако реляционная база данных обладает рядом дополнительных преимуществ.

- **Структурная независимость.** Поскольку реляционная модель базы данных не использует навигационную схему доступа к данным, маршрут доступа к данным не имеет значения для проектировщиков, программистов и конечных пользователей реляционной базы данных. Изменения в структуре реляционной БД ни в коем случае не влияют на доступ к данным со стороны СУБД. Поэтому в реляционной модели базы данных достигается структурная независимость, не свойственная сетевым и иерархическим моделям. (*Структурная независимость* имеет место, когда изменения в структуре БД не влияют на возможность доступа к данным со стороны СУБД. В отличие от реляционной базы данных, любые изменения в древовидной структуре иерархической базы данных или во *множествах* сетевой БД будут оказывать влияние на маршруты доступа к данным, что потребует изменений во всех программах доступа к данным.)
- **Концептуальная простота.** Хотя иерархическая и сетевая модели понятнее системы файлов, на смену которой они пришли, на концептуальном уровне реляционная модель еще более проста для понимания. Поскольку реляционная модель дарует нам роскошь отрешения от подробностей физического хранения данных, мы можем целиком сосредоточиться на *логическом* представлении базы данных, т. е. можно уделить большее внимание нашему естественному представлению о хранении данных, а не популярному, но "трудно постижимому" методу "взгляда" на данные со стороны компьютера.
- **Простота проектирования, реализации, управления и использования.** Поскольку в реляционной модели достигаются и независимость по данным, и структурная независимость, становится проще проектировать базу и управлять ее содержимым:

- ❑ *Нерегламентированные запросы.* Реляционные базы данных заняли доминирующее положение на рынке не в последнюю очередь еще и потому, что обладают мощной и гибкой возможностью создания запросов. Для большей части программного обеспечения реляционных БД стандартным языком запросов является *Structured Query Language* (SQL — язык структурированных запросов). На самом деле, даже существует незамысловатый стишок "реляционная модель применяет SQL", который можно услышать на встречах профессионалов и увидеть в популярных компьютерных журналах, например, в InfoWeek. Сокращение "SQL" одни произносят как "сиквел" (sequel), другие как "эс-ку-эль", но как его ни назови, именно SQL сделал чистый язык нерегламентированных запросов реальностью. SQL относится к так называемым *языкам четвертого поколения* (4GL), которые дают возможность пользователям определить, *что* необходимо сделать, без указания того, *как* это надо сделать. В РСУБД язык SQL используется при трансляции запроса пользователя в специальный код, необходимый для извлечения запрошенной информации. Следовательно, запросы в реляционной базе данных требуют меньшего программирования, чем в любой другой базе или в среде системы файлов.

Имейте в виду, что любое SQL-приложение реляционной БД состоит из трех частей: интерфейс пользователя, набор таблиц внутри БД и SQL-машина (SQL "engine"). Интерфейс включает в себя систему меню, операции запросов и генераторы отчетов. В основном такой интерфейс дает возможность конечному пользователю взаимодействовать с данными. Любой интерфейс это реализация представлений поставщика программного обеспечения об удобном способе манипулирования данными. Вы можете также разработать интерфейс с собственными настройками с помощью генераторов приложений, которые в настоящее время являются стандартными средствами в сфере программного обеспечения БД.

В значительной степени скрытая от конечного пользователя SQL-машина выполняет трудную работу. Внутри РСУБД SQL используется для создания структуры таблиц, обслуживания словаря данных и системного каталога, обеспечения доступа к таблицам БД, а также для трансляции запросов пользователя в формат, пригодный для обработки компьютером. Обслуживание системы базы данных является важнейшей задачей, которую РСУБД выполняет в значительной степени скрытно от конечного пользователя.

- ❑ *Мощная система управления базой данных.* Хорошая РСУБД является более сложной частью программного обеспечения, нежели СУБД иерархических и сетевых баз данных. Это связано с тем, что она выполняет гораздо больше задач (и значительно более сложных) как для проектировщиков, так и для пользователей. В результате хорошая РСУБД скрывает физический уровень сложности системы как от проектировщика, так и от пользователя.

Учитывая тот факт, что РСУБД выполняет невидимую глазу работу управления оборудованием, нет необходимости уделять внимание физическим аспектам реляционных баз данных. Вместо этого в последующих главах мы сосредоточим внимание на логической составляющей реляционной модели и методах ее проектирования.

Недостатки

Несмотря на существенные преимущества реляционной модели перед иерархической и сетевой, она имеет и ряд недостатков.

- ❑ *Существенные требования к оборудованию и системному программному обеспечению.* Та же РСУБД, которая скрывает от глаз пользователя всю сложность системы, требует значительных системных и аппаратных ресурсов. Для решения задач РСУБД требуется достаточно мощный компьютер. Следовательно, система с реляционной базой данных при прочих равных условиях будет работать медленнее, чем какая-либо другая. Однако учитывая быстрый рост возможностей оборудования и непрерывное развитие операционных систем, определение "медленнее" постепенно теряет свой смысл.
- ❑ *Возможность "скороспелых" проектов и реализаций.* В известном смысле удобная и простая в использовании среда разработки реляционных систем становится источником неприятностей. Реляционное программное обеспечение, особенно на уровне настольных компьютеров, настолько просто в использовании, что относительно малоподготовленный человек может легко создать искусные отчеты и запросы, не утруждая себя тщательным проектированием БД. По мере роста БД пробелы в проектировании начинают замедлять работу системы и могут привести к аномалиям в данных, которые ранее встречались лишь в системах файлов.
- ❑ *Опасность возникновения проблемы "информационных островков".* Поскольку реляционная база данных проста в использовании, слишком многие находят нужным создавать собственные подмножества баз данных и приложений. Хотя автономия конечного пользователя желательна, когда он делает запрос к общей базе данных, она же может стать причиной разработки подмножеств базы данных, которые становятся собственностью подразделений или отдельных лиц. Разрастание таких подмножеств баз данных может вызвать проблему "информационных островков", свойственную системам файлов, с такой же тенденцией к несовместимости данных, вызывающей проблемы структурирования информации и ее верификации.

Поскольку недостатков у реляционной базы данных значительно меньше (и они не столь уж существенны), чем достоинств, реляционные модели стали быстро доминировать на рынке БД. Однако возрастание сложности информационных систем побуждает профессиональных разработчиков продолжать поиск альтернативных концепций моделирования данных. Здесь основное внимание уделяется развитию моделей с акцентом на *визуальном* компоненте.

1.5.4. Модель "сущность-связь"

Концептуальная простота реляционной модели позволила расширить возможности баз данных. Поэтому внедрение реляционной технологии послужило поводом к появлению все более сложных транзакций и хранению более сложной информации. В свою очередь быстрое увеличение требований к транзакциям и хранимой информации вело к созданию более сложных структур базы данных, таким образом стимулируя появление все более изощренных и эффективных инструментальных средств проектирования БД (строительство небоскреба требует более обстоятельного проектирования, чем сколачивание собачьей конуры).

Для получения требуемых результатов сложнейшее проектирование должно было хоть как-то компенсироваться простотой концепции. Хотя реляционная модель была с понятийной точки зрения гораздо проще иерархической и сетевой моделей, ее достоинства все же были не столь велики, чтобы позволить ей стать эффективным инструментом проектирования. Поскольку проще исследовать структуру графически, чем описывать ее в текстовой форме, проектировщики БД посчитали удобным использовать соответствующие изобразительные средства, в которых логические объекты (сущности, entity) и их связи (relationship) могли изображаться графическими средствами.

Несмотря на то что в последнее десятилетие было разработано великое множество инструментальных средств моделирования данных, модель "сущность-связь" (ER-модель) является, безусловно, самым широко распространенным и простым графическим инструментом моделирования данных для разработки реляционных БД.

Питер Чен (Peter Chen) впервые ввел понятие ER-модели в 1976 году в своей основополагающей статье "The Entity-Relationship Model: Toward a Unified View of Data" (AM Transaction on Database Systems 1:1, March 1976). Модель "сущность-связь" (ER-модель) дает графическое представление логических объектов (сущностей) и их отношений в структуре базы данных. Именно такое графическое представление данных сделало ER-диаграммы популярным средством на концептуальном уровне моделирования данных. Более того, ER-модель дополнила концепции реляционной модели, создав основы хорошо структурированной среды проектирования, гарантирующей надлежащую разработку реляционных баз данных.

Основные понятия

ER-модели обычно представляются в виде диаграмм "сущность-связь" (ER-диаграмма, ERD). В ER-диаграмме используется графическое представление модели компонентов базы данных.

Примечание

Поскольку цель данной главы — дать общие понятия о концепциях моделирования данных, в этом разделе приводится упрощенное представление об ER-диаграммах. Далее у вас будет возможность ознакомиться с ER-диаграммами во всех подробностях, а в гл. 3 будут рассмотрены способы проектирования реальных структур баз данных с помощью ER-диаграмм.

Основу ER-модели составляют следующие компоненты.

- Ранее в этой главе мы определили *сущность* (логический объект), как "персону, местоположение или предмет, сведения о которых подлежат сбору и хранению". В ER-модели сущность представлена в виде прямоугольника. Название сущности (имя существительное) записывается в центре прямоугольника, как правило, заглавными буквами и предпочтительнее в единственном числе: PAINTER лучше чем PAINTERS и EMPLOYEE лучше чем EMPLOYEES. Обычно если ER-диаграмма связана с реляционной моделью, сущность отображается на реляционную таблицу. Каждая строка реляционной таблицы соответствует *экземпляру сущности* (entity instance или entity occurrence) в терминах ER-модели.

Примечание

Совокупность подобных сущностей называется *набором сущностей* (entity set). Например, можно представить себе файл AGENT (см. рис. 1.4) как совокупность трех агентов (сущностей) в наборе сущностей AGENT. Формально говоря, ER-диаграмма описывает наборы сущностей. К сожалению, разработчики ER-диаграмм под "сущностью" понимают "набор сущностей", и мы будем следовать этой устоявшейся практике при обсуждении ER-диаграмм и их компонентов.

Сущность описывается набором *атрибутов*. Каждый атрибут описывает отдельное свойство сущности. Например, сущность EMPLOYEE (сотрудник) имеет такие атрибуты, как номер социального страхования, фамилию и имя.

- *Связь* описывает соединение между данными. Большинство связей описывает соединение между двумя сущностями. При обсуждении моделей баз данных мы указали три возможных типа связей между данными: "один-ко-многим" (1:M), "многие-ко-многим" (M:N) и "один-к-одному" (1:1). Разработчики ER-диаграмм для обозначения типа связи используют термин *связность* (connectivity) (на ER-диаграммах связность записывается рядом с прямоугольником, соответствующим сущности). Связь изображается на ER-диаграмме ромбом, соединенным с соответствующей сущностью. Название связи (в глагольной форме) записывается внутри ромба. Например, в каждом из подразделений (DEPARTMENT) компании *работает* множество сотрудников (EMPLOYEE). А художник (PAINTER) *пишет* много картин (PAINTING).

На рис. 1.13 проиллюстрированы ER-диаграммы с использованием связей, которые мы обсуждали при знакомстве с моделями баз данных (см. разд. 1.5).



Рис. 1.13. Представление отношений на ER-диаграмме (модель Чена)

ER-диаграмма, представленная на рис. 1.13, построена на основе так называемой *модели Чена*, описанной Питером Ченом в его вышеупомянутой работе. На ER-диаграммах, созданных на базе модели Чена, связи обозначаются ромбами, а сущности — прямоугольниками.

Другой широко известный способ построения ER-диаграмм, называемый "птичья лапка", представлен на рис. 1.14. Название "птичья лапка" произошло от специфичного вида символа, которым обозначается связь в направлении "многие" (хотя мы будем пользоваться ER-диаграммами на основе модели Чена, все же необходимо уметь применять и другие модели).



Рис. 1.14. Представление отношений на ER-диаграмме (модель "птичья лапка")

Преимущества

Человеку удобнее работать с графическим изображением (любое самое подробное и полное описание дома не может сравниться с его изображением). Поэтому не удивительно, что ER-модель стало трудно отделить от ER-диаграмм, с помощью которых она изображается. Фактически проектировщики баз данных считают ER-модель и ER-диаграмму двумя сторонами одной медали. В любом случае мы можем указать их достоинства.

- ❑ **Исключительная концептуальная простота.** Все модели баз данных обеспечивают лучшее логическое представление данных, чем система файлов. Однако ER-модель дает очень простое и наглядное представление об основных логических объектах БД и существующих между ними связях. Поэтому использование такой модели значительно упрощает разработку и организацию сложнейших баз данных.

- ❑ *Наглядное представление.* ER-модель дает проектировщикам баз данных, программистам и конечным пользователям простое наглядное представление о данных и связях между ними. Поэтому ER-модель является чрезвычайно эффективным средством, интегрирующим различные представления о данных в единую рабочую среду.
- ❑ *Интеграция с реляционной моделью баз данных.* ER-модель прекрасно увязывается с реляционной моделью БД. Такая интеграция помогает хорошо структурировать процесс проектирования реляционных БД.

Недостатки

Несмотря на то что ER-моделирование и ER-диаграммы стали популярным средством в арсенале методов разработки баз данных, у проектировщиков все же имеется к ним ряд претензий.

- ❑ *Недостаточные возможности представления ограничений.* С помощью этой модели легко изобразить ограничения, имеющие непосредственное отношение к связности. Например, ограничение "преподаватель может вести как несколько групп, так и ни одной группы (если он научный работник), но не более четырех групп" легко изображается средствами ER-модели. К сожалению, есть множество важнейших ограничений, которые невозможно смоделировать методами ER-модели, например: "оценки студентов варьируются в диапазоне от 0.0 до 4.0" и "пилот не может работать более 10 часов подряд". Такие ограничения должны обрабатываться на уровне приложений.
- ❑ *Ограниченные возможности представления отношений.* Связи представляются как нечто происходящее *между* сущностями. Поэтому связи между атрибутами *внутри* сущностей не могут быть представлены средствами ER-модели. Например, нет способа отобразить связь между оценкой качества подготовки студента и количеством прослушанных им часов. К тому же когда сущности связаны множеством связей с другими сущностями, значение таких связей будет трудно оценить.
- ❑ *Отсутствие языка манипулирования данными.* Сторонники реляционной модели обычно указывают на отсутствие команд манипулирования данными в ER-модели. Отсутствие таких команд делает ER-модель "неполной".
- ❑ *Утеря информационного наполнения.* Эта модель сильно "переполняется", если в ней отобразить все ее атрибуты. Поэтому проектировщики баз данных обычно избегают полного отображения атрибутов, таким образом уменьшая информационное наполнение ER-модели.

Недостатки ER-модели подчас затрудняют моделирование сложных типов данных и связей, которые становятся обычным делом в современных базах данных. Все же, несмотря на эти недостатки, ER-модель и ER-диаграммы живут и процветают. На самом деле, поставщики выпустили так много расширений для базовых ER-диаграмм, что они продолжают оставаться важнейшим инструментом проектирования баз данных. Тем не менее, поиск наилучших инструментальных средств моделирования данных продолжается, поскольку сфера обработки данных продолжает развиваться.

1.5.5. Объектно-ориентированная модель

Все более усложняющиеся практические задачи стимулируют появление иных моделей данных, точнее отображающих реальный мир. Первой из таких моделей стала семантическая модель данных (semantic database model — SDM), разработанная М. Хаммером (M. Hammer) и Д. Маклеодом (D. McLeod) в 1981 году в работе "Database Description with SDM: A Semantic Database Model" (ACM Transactions on Database Systems 6:3).

SDM позволяет моделировать как данные, так и их отношения в единой структуре, называемой *объектом*. Поскольку основной структурой модели является объект, модель SDM получила название *объектно-ориентированной модели базы данных (object-oriented database model, OODM)*. В свою очередь OODM стала основой создания объектно-ориентированной модели базы данных (OODBM), управление которой осуществляется с помощью *системы управления объектно-ориентированной базой данных (ООСУБД, OODBMS)*.

Примечание

В этом разделе обсуждаются лишь основные понятия объектно-ориентированной модели, используемые при моделировании и управлении данными. Подробно OODM, ее принципы и понятия рассматриваются в гл. 11.

Объектно-ориентированная модель базы данных отражает совершенно иной способ определения и использования сущностей. Как и сущность в реляционной модели базы данных, объект определяется своим фактическим содержанием. Но в отличие от сущности, в объект включается информация о связях, существующих внутри объекта, а также информация о его связях с другими объектами. Поэтому фактические сведения внутри объекта более значимы.

Примечание

Слово "семантика" означает *смысл, значение*. Поэтому в терминах объектно-ориентированной БД говорят, что объект имеет семантическое наполнение. Вот почему Хаммер и Маклеод назвали свое детище семантической моделью базы данных.

Последующее развитие OODM дало возможность включить в содержимое объекта все операции, которые могут над ним выполняться, например, изменение значения, поиск величины, распечатка параметров. Поскольку объект включает в себя и данные, и различные типы связей, и процедуры, его можно рассматривать как самостоятельный модуль, который может (способен) стать основным функциональным блоком автономных структур.

Хотя идеи OODM использовались уже начиная с 70-х годов двадцатого века, по крайней мере четыре фактора оказали влияние на значительное укрепление позиций этого подхода в 1990-х годах.

- Растущая стоимость производства и необходимость поддержки все более сложного программного обеспечения привлекли основное внимание к возможности повторного использования кода. Разработка программ в виде самостоятельных модулей, выполняющих специфичные функции, оказала самое сильное влияние

на возможность повторного использования кода. Такие модули могут затем "компоноваться" различными способами для создания широкого спектра приложений. Свойства объектно-ориентированной модели базы данных позволяли считать ее очень перспективной в сфере модульных концепций.

- ❑ В современные базы данных зачастую включают графику, видео и звук в дополнение к текстовой и цифровой информации. Усложнение типов данных и повышение требований к системе перестало укладываться в рамки реляционного подхода. Поскольку OODM ставит своей целью как раз обработку сложных типов данных, она и стала естественной основой для создания объектно-ориентированной модели базы данных.
- ❑ По мере того как информационная среда становилась все более сложной, возникла необходимость поддержки все более сложных транзакций, также повысились требования к качеству информации. Например, компания автомобильных грузоперевозок может счесть необходимой разработку специальных подробных компьютерных карт, на которых будут точно указаны местоположение трейлеров, пункт назначения каждого трейлера, его груз и примерное время доставки. Некой машиностроительной компании может потребоваться покомпонентный вывод изображения изделия для разрешения по телефону проблем, возникших у клиента.
- ❑ Возросшая мощность компьютеров позволяет выполнять на них огромную вычислительную работу при обработке сложных объектно-ориентированных данных, выполнении соответствующих процессов и процедур.

Примечание

Основы объектно-ориентированной модели базы данных (OODM) были разработаны в 70-х годах двадцатого века в исследовательском центре компании Xerox в Palo Alto. Исследования компании Xerox привели к крупным достижениям в компьютерной сфере, которые мы считаем сегодня само собой разумеющимися: графический интерфейс пользователя, компьютерная мышь, электронная почта и объектно-ориентированные языки программирования.

Основные понятия

Основу объектно-ориентированной модели данных составляют следующие компоненты.

- ❑ Объекты модели данных являются абстракциями сущностей и событий реального мира. В общих чертах любой объект может рассматриваться как эквивалент сущности ER-модели. Точнее, любой объект представляет только один экземпляр сущности (семантическое наполнение объекта определяется через несколько элементов этого списка).
- ❑ Атрибуты описывают свойства объекта. Например, в объект PERSON (персона) включены атрибуты Name (имя), Social Security Number (номер социального страхования) и Date of Birth (дата рождения).
- ❑ Объекты, которые совместно используют одни и те же характеристики, группируются в классы. *Класс* представляет собой совокупность подобных объектов со структурой совместного доступа (атрибуты) и поведением (методы). В общем случае класс напоминает набор сущностей ER-модели. Однако класс отличается

от набора сущностей тем, что он содержит набор процедур, называемых *методами*. Метод класса представляет собой реальное действие, например, поиск заданного имени в объекте PERSON, изменение имени или распечатку адреса. Иначе говоря, методы эквивалентны процедурам в традиционных языках программирования. В терминах объектно-ориентированного подхода методы определяют *поведение* объекта (некоторые варианты OODM, например, семантическая модель объекта, не включают в себя понятие метода).

- Классы организованы в *иерархию классов*. Иерархия классов похожа на перевернутое дерево, в котором каждый класс имеет только одного предка. Например, классы CUSTOMER и EMPLOYEE имеют родительский класс PERSON (обратите внимание на явное сходство с иерархической моделью!).
- *Наследование* — это возможность объекта внутри иерархии классов наследовать атрибуты и методы классов, структурно расположенные выше его. Например, мы можем создать два класса CUSTOMER и EMPLOYEE как подклассы класса PERSON. В этом случае классы CUSTOMER и EMPLOYEE будут наследовать все атрибуты и методы от класса PERSON.

Чтобы проиллюстрировать различие между OODM и ER-моделью, рассмотрим их графическое представление для простой задачи выписки счета (рис. 1.15).

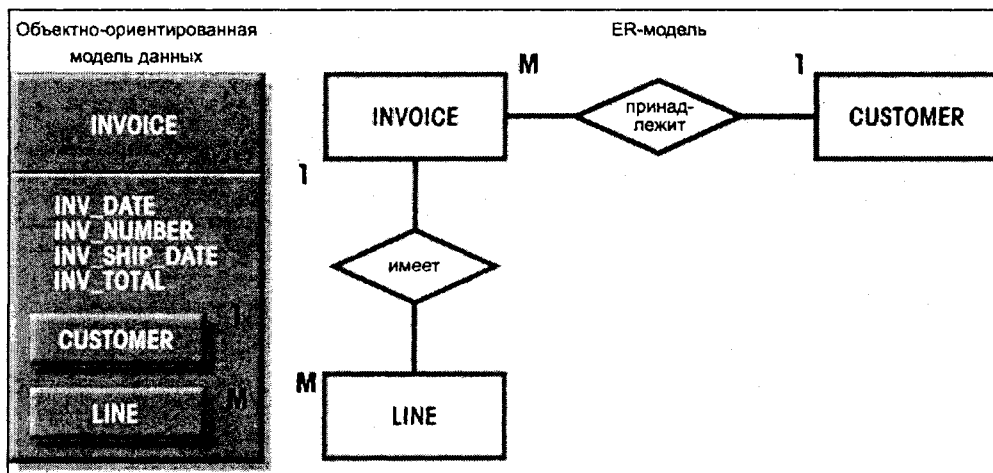


Рис. 1.15. Сравнение объектно-ориентированной модели и ER-модели

На этом рисунке можно заметить следующее.

- Объектно-ориентированная модель данных представляет объект в виде вертикального прямоугольника, все атрибуты объекта и связи с другими объектами включены в этот прямоугольник. Объектное представление INVOICE включает все связанные с ним объекты в *той же* прямоугольнике.
- Для представления транзакции выписки счета в ER-модели используются три различных сущности и две связи. Поскольку клиенты могут купить более чем

один предмет за один раз, каждый счет может содержать одну или более строк, каждый предмет в отдельной строке! И поскольку счета создаются клиентами, требования к моделированию данных предполагают наличие сущности "клиент" и связи между клиентом и счетом.

Преимущества

Объектно-ориентированная модель данных имеет несколько важнейших преимуществ перед ER-моделью.

- ❑ *Добавление семантического наполнения.* Добавление семантического наполнения делает модель данных более значимой. Например, объектно-ориентированная модель, представленная на рис. 1.15, демонстрирует отношение между клиентом и счетом, а также между счетом и строками счета внутри объекта "счет". Эта информация дополняет атрибуты объекта "счет".
- ❑ *Во внешнее представление включено семантическое наполнение.* Как и ER-диаграммы, объектно-ориентированная модель представляет отношения в наглядной форме. Однако в наглядное представление объектно-ориентированной модели включается семантическое наполнение, что упрощает визуализацию сложных отношений внутри и между объектами.
- ❑ *Целостность базы данных.* Так же как и иерархическая, объектно-ориентированная модель использует наследование для защиты целостности базы данных. Однако объекты OODM содержат большее число типов связей и более сложные связи.
- ❑ *Структурная независимость и независимость по данным.* Автономия объекта объектно-ориентированной модели гарантирует структурную независимость и независимость по данным.

Недостатки

Несмотря на впечатляющие достоинства объектно-ориентированной модели, она все же не смогла выиграть сражение с реляционными базами данных. Причин у этого поражения много, и недостатки OODM сыграли здесь не последнюю роль.

- ❑ *Отсутствие должной стандартизации.* Пока не существует стандартов для объектно-ориентированной модели. Хуже всего то, что нет стандартного метода доступа к данным. Этот недочет создает проблемы при доступе к данным от различных источников (различные поставщики поддерживают различные методы доступа к данным, как правило, несовместимые).
- ❑ *Сложная навигация доступа к данным.* Метод доступа к данным похож на стиль иерархической и сетевой модели.
- ❑ *Трудность изучения.* Недостаток стандартизации и трудности, вызванные навигационным стилем доступа к данным, приводят к затруднениям в изучении объектно-ориентированной модели, даже большим, чем при изучении реляционной модели. Несмотря на то что мы с легкостью используем объекты — перетаскиваем объекты Windows на экране дисплея, не заботясь и не задумываясь о тех процессах, которые обеспечивают это действие, — моделирование данных и реализация объектно-ориентированных баз данных — это совершенно другая история.

Каждая новая модель данных "обучалась" на недостатках предыдущих моделей. Сетевая модель сменила иерархическую, поскольку ее разработчики упростили представление сложных ("многие-ко-многим") связей. В свою очередь реляционная модель имеет преимущество перед иерархической и сетевой моделями, поскольку упрощает представление данных, обеспечивая высокую независимость и предоставляет в распоряжение пользователей относительно простой язык запросов. Хотя обсуждение достоинств и недостатков более старых иерархических и сетевых моделей и новейших реляционных моделей продолжается уже несколько лет, реляционная модель, тем не менее, сейчас является основной моделью баз данных для бизнес-приложений.

В связи со значительным усложнением приложений появились две новые модели данных: объектно-ориентированная модель (OODM) и расширенная реляционная модель данных (Extended Relational Data Model — ERDM). Модель ERDM вышла победителем в соревновании этих двух моделей по признанию многих исследователей баз данных и стала ответом на вызов, сделанный приверженцами объектно-ориентированной модели. Эта модель включила в себя основные достоинства объектно-ориентированной модели, унаследовав простоту структуры реляционных баз данных. Вот почему СУБД, основанные на ERDM, часто называют *объектно-реляционными системами управления базой данных (ОРСУБД)*.

Принимая во внимание огромное количество установленных реляционных баз и появление ERDM, объектно-ориентированная модель встретила сильное сопротивление. Однако идеи объектно-ориентированной модели стали основой многих систем анализа баз данных и процедур проектирования. Вдобавок объектно-ориентированный подход используется во многих генераторах приложений и других инструментальных средствах.

Сегодняшняя битва за лидерство между OODM и ERDM на рынке баз данных очень напоминает борьбу между сетевой и иерархической моделями и реляционной моделью десятилетие назад. Такое чувство, что OODM и ERDM соревнуются в том, кто из них сможет разместить большее количество семантической информации в рамках своей модели. Однако OODM и ERDM различаются в своей основе и по стратегии, и по природе подхода к проблеме. Если ERDM основана на концепциях реляционной модели, то OODM базируется на концепциях объектно-ориентированной модели и семантической модели данных. Модель ERDM в основном приспособлена для бизнес-приложений, в то время как OODM используется в весьма специальных инженерных и научных приложениях. Наиболее вероятным сценарием завершения борьбы двух подходов представляется вариант постепенного слияния объектно-ориентированных и реляционных концепций, а также связанных с ними процедур.

Хотя название "ERDM" часто используется в литературе по базам данных в качестве удачного ответа приверженцев реляционной модели на вызов OODM, К. Дейт² возражает против названия "ERDM" по следующим причинам:

- полезный вклад "объектной модели" состоит в том, что она позволяет пользователям определять собственные (зачастую очень сложные) типы данных. Однако в реляционной модели есть математические структуры, именуемые "доменами", также предоставляющие эту возможность. При надлежащей поддержке доменов реляционные модели баз данных будут иметь достаточные ресурсы для обработки

² C. J. Date, "Back to the Relational Future" ("Назад в реляционное будущее").

сложных данных, с которыми в ряде случаев сталкиваются в инженерном проектировании, автоматизации работы офиса, финансовом моделировании и т. д. Поскольку реляционная модель может обеспечить поддержку сложных типов данных, понятие "расширенная модель реляционных баз данных", или ERDM, "чрезвычайно неточно и неуместно" и "его нужно без колебаний отвергнуть" (возможность, которую пытаются внедрить "расширением", и так существует!);

- ❑ точно так же название "объектно-реляционная модель базы данных" (object/relational database model — O/RDM) не очень точно, поскольку домен реляционной модели базы данных не является объектом структуры модели. Однако на рынке уже имеется достаточно много продуктов с названием O/R (которые известны как *универсальные серверы баз данных*). Поэтому Дейт признает, что с названием O/R придется смириться. На самом деле Дейт уверен, что "O/R-системы — это будущее для всех". Точнее, Дейт приводит доводы, что настоящие OR-системы будут "не более чем настоящими реляционными системами — т. е. системами, поддерживающими реляционную модель со всеми вытекающими отсюда последствиями".

В заключение своих рассуждений К. Дейт заметил, что "нам не нужно ничего делать с реляционной моделью для достижения объектных функций (ничего, за исключением ее внедрения, т. е. именно того, что пока так и не сделано в мире коммерции)".

Поскольку К. Дейт считается одним из признанных лидеров среди философов и новаторов, его мнением невозможно пренебречь. В любом случае независимо от названия, используемого для возросших возможностей реляционной модели базы данных, очевидно, что реляционная модель БД по всей видимости в ближайшем будущем сохранит ведущую роль на рынке. Вот откуда исходит наша уверенность в том, что именно реляционной модели данных необходимо уделять наибольшее внимание.

Исследуя эволюцию моделей данных, можно определить некоторые общие свойства, которыми должны обладать эти модели для того, чтобы стать общепризнанными.

- ❑ Модель данных должна иметь достаточный уровень концептуальной простоты без ущерба для семантической полноты базы данных. Бессмысленно иметь модель данных, понять которую труднее, чем окружающую действительность.
- ❑ Модель данных должна как можно точнее соответствовать реальному миру. Этой цели проще достичь, если вкладывать больше смысла (семантики) в представление данных модели (семантика имеет отношение к динамическому поведению данных, в то время как представление данных составляет статический аспект сценария реальной действительности).
- ❑ Устойчивость и целостность любых моделей данных должны быть согласованы с представлением о преобразованиях (поведении) в реальном мире.

1.6.1. Модели баз данных и Интернет

Если что-то и есть постоянное в мире баз данных, так это то, что стратегии баз данных всегда находятся в движении. Использование Интернета как важнейшего инструмента для бизнеса радикально изменило роль и масштабы рынка баз данных. Фактически влияние, которое Интернет оказал на рынок баз данных, послужило поводом создания новых стратегий БД, в которых OODM и ERDM-O/RDM зани-

мают скромное положение при разработке доступа к базам данных через Интернет. Поэтому теперь вместо продолжения дуэли баз данных "OODM против ERDM-O/RDM" мы видим, что поставщики ведут основные разработки в направлении создания систем баз данных, обеспечивающих простой и удобный интерфейс с Интернетом. Говоря кратко, популярная база данных в "эпоху Интрнета" должна иметь следующие свойства:

- гибкий эффективный и безопасный доступ в Интернет, простой в использовании, разработке и обслуживании;
- поддержка сложных типов данных и связей (сложные типы данных должны включать видео и звук);
- интерфейс с множеством источников данных и структур;
- относительная простота концептуальной модели БД, что позволит сделать проектирование базы данных не столь обременительным в очень сложных информационных средах;
- обилие доступных инструментальных средств проектирования и реализации баз данных, а также средств разработки приложений;
- мощный графический интерфейс пользователя СУБД, упрощающий работу администратора БД.

Очевидно, что если база данных вписывается в общую структуру Интернета, то ее происхождение не имеет существенного значения. Это одна из причин преуспевания реляционной модели, которая заимствовала компоненты из других моделей баз данных. Например, новейшая база данных корпорации Oracle — Oracle 9 — содержит объектно-ориентированный компонент внутри структуры реляционной базы данных, точно так же, как текущая версия базы данных DB2 компании IBM. Может быть, самым главным препятствием господству OODM является то, что преимущества использования Интернета — а, следовательно, повышение значимости бизнеса — имеют большее значение, чем обеспечение чистых объектно-ориентированных инструментальных средств. Поэтому объектно-ориентированное моделирование нацелено сейчас в основном на поддержку работы в Интернете, вместо разработки баз данных, основанных на концепциях объектно-ориентированного моделирования.

Все же несмотря на то, что большинство разработок современных приложений БД предназначаются для работы в Интернете, необходимо помнить, что все бизнес-транзакции Интернета в конечном счете взаимодействуют с какой-нибудь базой данных. Если такая БД плохо спроектирована, самый лучший интерфейс с Интернетом не сможет исправить структурные проблемы БД и интернет-приложение не будет работать. Вот почему в этой книге уделяется основное внимание проектированию базы данных и проблемам, оказывающим влияние на процесс проектирования. Интернет, которому посвящены *гл. 14 и 15*, просто добавляет новый уровень доступа к основным базам данных.

Резюме

Необходимые сведения извлекаются из данных, хранящихся в базе данных. Для реализации базы данных и управления ее содержимым требуется коммерческое программное обеспечение, называемое системой управления базами данных (СУБД).

Проектирование базы данных определяет структуру БД, СУБД хранит сведения о структуре внутри самой базы данных. Таким образом, база данных содержит данные, которые вам необходимо накапливать, и "данные о данных" (метаданные). Хороший проект базы данных имеет очень большое значение, поскольку даже хорошая СУБД будет плохо работать с плохо спроектированной БД.

Базами данных предшествовали системы файлов. Так как системы файлов не имели в своем распоряжении СУБД, управлять файлами с ростом системы файлов становилось все труднее. Для каждого файла требовался собственный набор программ управления данными, и поскольку файлы обычно считались собственностью тех, кто их создал, количество их постоянно росло. Многие файлы зачастую содержали избыточные данные, что приводило к несовместимости данных, аномалиям в данных и недостаточной целостности данных. Поскольку каждый файл мог быть использован несколькими прикладными программами, сформировавшаяся файловая система требовала создания сотен, а иногда тысяч программ. Серьезные проблемы управления системой файлов часто являлись результатом зависимости по данным — доступ к любому файлу зависел от свойств данных и формата их хранения, поэтому даже небольшие изменения в структуре внутри файла требовали изменения всех программ, получающих доступ к информации этого файла.

Для устранения недостатков, присущих системам файлов, были разработаны модели баз данных. Вместо того чтобы размещать данные в различных файлах, базы данных хранили данные в едином хранилище, давая таким образом возможность СУБД осуществлять надежный контроль действий БД. Широкое распространение получили три основные модели баз данных — иерархическая, сетевая и реляционная. Во всех трех моделях используется СУБД для того, чтобы специалисты по обработке данных могли лучше выполнять свою работу, чем это было возможно в системах файлов. Однако сфера действия СУБД и ее дружелюбность пользователю заметно отличаются в различных системах базы данных.

Иерархическая модель базы данных основана на древовидной структуре, включающей в себя корневой сегмент, сегменты-предки и дочерние сегменты. Сегмент эквивалентен типу записи файла. Иерархическая модель БД отображает набор связей "один-ко-многим" (1:M) между предком и потомком. Для навигации по структуре иерархической модели используется иерархическая последовательность или прямой обход, который всегда начинается с самой левой части дерева.

Структура "предок-потомок" иерархической модели обеспечивает целостность и непротиворечивость базы данных: невозможно существование записи-потомка без записи-предка. Вдобавок, хорошо спроектированная иерархическая БД очень эффективна при обработке данных большого объема или при наличии множества транзакций.

К сожалению, иерархическая модель дает лишь некоторые преимущества в независимости данных. Хотя изменения в свойствах данных не требуют изменений в программах, использующих эти данные, любые изменения в структуре БД потребуют большой работы. Например, изменение расположения сегмента потребует изменений во всех программах, использующих этот сегмент внутри иерархического маршрута. Более того, изменения в сегментах или в их местоположении требуют от DB-менеджеров выполнения сложной работы. Несмотря на предпринимаемые усилия, удаление сегмента может привести к неумышленному удалению всех сегментов, расположенных ниже удаленного.

Поскольку иерархическая модель не имеет возможности обслуживать нерегламентированные запросы, прикладное программирование требует значительных усилий и ресурсов. Проектирование иерархических БД может также вызвать затруднения, поэтому эффективность СУБД трудно оценить. К тому же в иерархических БД трудно реализовать отношения со многими предками, как и связи типа "многие-ко-многим" (M:N). Наконец, из-за отсутствия стандартов иерархическая модель недостаточно мобильна (преимущества и недостатки различных моделей сведены в табл. 1.3, которая находится ниже в этом разделе).

В сетевой модели была сделана попытка устранить многие ограничения, свойственные иерархическим БД. Группа DBTG (Database Task Group) постоянно действующей конференции CODASYL разработала стандарты спецификаций сетевых БД для сетевой схемы, подсхем и языка управления данными. Язык управления данными содержал три основных компонента: язык определения данных (DDL), который использовался для определения компонентов схемы, подсхему DDL и язык манипулирования данными (DML), разработанный для манипулирования содержимым БД.

Хотя сетевая модель напоминает иерархическую модель, ее структура легко настраивается на обработку связей со многими предками. Даже основные компоненты модели похожи: сетевая запись-член эквивалентна иерархическому потомку, а сетевая запись-владелец эквивалентна иерархическому предку.

Сетевая модель использует большинство достоинств иерархической модели и улучшает многие из них. Например, целостность данных сетевой модели обеспечивается фактически тем, что запись-член не может существовать без записи-владельца. Сетевое множество определяет связь между владельцем и членом; ее существование дает возможность программе получать доступ к записи-владельцу и ко всем записям-членам внутри множества, обеспечивая таким образом большую гибкость доступа к данным, чем та, которая была возможна в иерархической модели.

Несмотря на всю результативность сетевой БД, ее структурная сложность зачастую ограничивает ее эффективность: проектировщики должны быть хорошо знакомы со структурой БД, чтобы добиться высокой производительности. Как и в случае с иерархической моделью, независимость по данным не сопровождается структурной независимостью. Любое изменение в структуре базы данных требует, чтобы все определения подсхемы проверялись перед тем, как какая-нибудь прикладная программа сможет получить доступ к базе данных. Другими словами, сетевой базой данных иногда очень трудно управлять, и ее сложность может вызывать проблемы при проектировании БД.

Реляционная модель базы данных в настоящее время является стандартом для разработки баз данных. Система управления реляционной базой данных (РСУБД) настолько сложна, что пользователи и проектировщики оперируют лишь с логическим представлением базы данных, а подробности физической реализации хранения, маршрутов доступа и структур данных берет на себя РСУБД. Поэтому проектировать реляционные БД проще, чем разрабатывать иерархические и сетевые БД.

Поскольку физический уровень реализации системы с помощью РСУБД скрыт от пользователей и проектировщиков, в реляционной БД автоматически обеспечивается как независимость по данным, так и структурная независимость. Поэтому управление данными стало проще, чем в прежних моделях.

В реляционной конфигурации требуется гораздо меньше программирования, потому что в состав реляционной БД включен мощный язык запросов SQL (Structural Query Language — язык структурированных запросов), позволяющий обрабатывать нерегламентированные запросы. Кроме того, в РСУБД имеется множество утилит, упрощающих проектирование и разработку отчетов и экранных форм ввода/вывода, тем самым еще более снижая потребность в программировании.

Возросшая сложность баз данных стала причиной появления простой интерпретируемой графической среды проектирования БД. Таким инструментом стали ER-диаграммы (диаграммы "сущность-связь"), в основе которых лежит модель "сущность-связь" (ER-модель). ER-модель имеет большое практическое значение, потому что она основана на наглядном представлении данных и их связей. Кроме того, ER-модель позволяет проектировщикам баз данных получать различное представление данных с точки зрения программистов, конечных пользователей и самих проектировщиков. С помощью ER-модели проектировщики могут интегрировать различные представления данных в единую рабочую среду. ER-модель породила новое поколение инструментальных средств проектирования БД, причем самое широкое распространение получили ER-диаграммы.

По сравнению с другими моделями ER-модель обеспечивает простой концептуальный взгляд на сущности и связи базы данных. Поэтому процесс проектирования сложных БД упрощается и им легче управлять. Так как ER-модель хорошо интегрируется с реляционной моделью, которая все еще доминирует на рынке моделей БД, ER-модель и ER-диаграммы совместно с реляционной моделью БД стали мощным инструментом, помогающим структурировать процесс проектирования БД.

Мир реальной информации становится все сложнее. Требования к информационному обеспечению ужесточаются и включают в себя необходимость обработки таких сложных типов данных, как изображения, звук и видео. Усложнение информационной среды стимулировало поиск других моделей баз данных для более удобного проектирования, управления и реализации. Первой такой моделью стала семантическая модель данных (semantic data model, SDM). SDM является объектно-ориентированной моделью данных (OODM), ставшей основой объектно-ориентированной модели баз данных (OODBM), которая управляется системой управления объектно-ориентированными базами данных (OODBMS, ООСУБД). Основной моделирующей структурой SDM является объект. Объект напоминает сущность, в которую включены сведения, определяющие объект. Но в отличие от сущности, в объект также включается информация о связях между сведениями, а также связи с другими объектами, что делает эти данные более значимыми. Следовательно, объект имеет большее семантическое (смысловое) наполнение, чем сущность, которая использовалась в ранних моделях данных.

Объекты, у которых имеются одинаковые свойства, объединяются в классы. Класс — это совокупность подобных объектов с совместной структурой (атрибуты) и поведением (методы). В этом смысле класс напоминает набор сущностей (entity set). Однако в отличие от набора сущностей в класс включен набор процедур, которые называются методами. Классы организуются в иерархии классов, в которых у каждого объекта есть свой предок. Структура иерархии классов дает возможность каждому объекту в иерархии наследовать атрибуты, связи и методы классов, находящихся выше уровнем, что обеспечивает целостность данных. Учитывая их улучшенное се-

матическое наполнение и наличие методов, объекты можно считать (потенциально) основными функциональными блоками для создания автономных структур, позволяя реализовать модульный принцип проектирования и реализации.

Несмотря на все преимущества объектно-ориентированной модели данных, она все же испытывает недостаток стандартизации, имеет достаточно трудную навигационную систему доступа к данным и относительно медленное выполнение транзакций, что требует увеличения расходов на приобретение более мощного оборудования. Все это препятствует широкому распространению объектно-ориентированной модели в качестве стандарта разработки баз данных. Напротив, сторонники реляционной модели, восприняв все самое лучшее из объектно-ориентированного подхода, разработали расширенную реляционную модель (ERDM — Extended Relational Data Model). В настоящее время OODM широко используется в специальных инженерных и научных приложениях, в то время как ERDM в основном применяется при разработке бизнес-приложений. Хотя наиболее вероятным сценарием можно было считать слияние технологий OODM и ERDM, вся эта борьба отошла на второй план в связи с возникшей потребностью разработки стратегии доступа к базам данных из Интернета.

Название "объектно-реляционная" (object/relational) часто используется для указания на то, что данная реляционная модель данных включает в себя объектно-ориентированные возможности. Хотя объектно-ориентированные возможности реляционной модели часто описываются как расширения реляционной модели, К. Дейт показал, что домены реляционной модели при их правильном применении также имеют достаточные возможности выполнения задач, которые считаются "объектно-ориентированными". Поскольку реляционная модель баз данных уже содержит в себе "объектные" возможности, реализуемые с помощью доменов, Дейт полагает, что название "объектно-реляционная" является избыточным, но все же в какой-то мере допустимым, а вот название "расширенная реляционная" (extended relational, ER) — совершенно неуместным. В сводной табл. 1.3 приведены свойства моделей баз данных, которые обсуждались в этой главе.

Таблица 1.3. Достоинства и недостатки различных моделей баз данных

Модель базы данных	Неза- виси- мость по данным	Структур- ная неза- висимость	Преимущества	Недостатки
Иерархи- ческая	Да	Нет	<ul style="list-style-type: none"> • Обеспечивает совместное использование данных • Концептуальная простота обеспечивается отношениями предок/потомок • Целостность базы данных обеспечивается отношениями предок/потомок 	<ul style="list-style-type: none"> • Навигационная система усложняет проектирование, внедрение, разработку приложений, использование и управление • Ограничения в реализации (отсутствие связей M:N или связей с несколькими предками)

Таблица 1.3 (продолжение)

Модель базы данных	Неза- виси- мость по данным	Структур- ная неза- висимость	Преимущества	Недостатки
			<ul style="list-style-type: none"> Эффективная работа с постоянными связями 1:M 	<ul style="list-style-type: none"> Отсутствие в СУБД языка определения данных или языка манипулирования данными Отсутствие стандартизации
Сетевая	Да	Нет	<ul style="list-style-type: none"> Концептуальная простота, по крайней мере, сравнимая с иерархической моделью Обработка большого числа типов связей, таких, например, как M:N и связей со многими предками Отношения владелец/член обеспечивают целостность базы данных Стандартизация В СУБД включены язык определения данных и язык манипулирования данными 	<ul style="list-style-type: none"> Сложность системы снижает ее эффективность (навигационный метод доступа к данным) Навигационная система доступа к данным усложняет проектирование, реализацию, разработку приложений, использование и управление
Реляционная	Да	Да	<ul style="list-style-type: none"> Табличное представление существенно улучшает концептуальную простоту, таким образом, упрощая проектирование, внедрение, управление и использование БД Возможность обработки нерегламентированных запросов на основе SQL Мощная РСУБД упрощает реализацию и внедрение 	<ul style="list-style-type: none"> Мощная РСУБД предъявляет повышенные требования к оборудованию и системному программному обеспечению Концептуальная простота системы допускает возможность плохого использования хорошей системы слабо подготовленным пользователем Может возникнуть проблема "информационных островков", поскольку отдельные пользователи и подразделения считают возможным разрабатывать свои собственные приложения

Таблица 1.3 (окончание)

Модель базы данных	Неза- виси- мость по данным	Структур- ная неза- висимость	Преимущества	Недостатки
Сущность- связь (ER-модель)	Да	Да	<ul style="list-style-type: none"> Визуальное моделирование обеспечивает исключительную концептуальную простоту Визуальное представление делает ее эффективным инструментом Интегрируется с наиболее влиятельной реляционной моделью базы данных 	<ul style="list-style-type: none"> Недостаточные возможности представления ограничений Недостаточные возможности в представлении связей Отсутствие языка манипулирования данными Недостаток семантического наполнения, поскольку атрибуты обычно удаляются, чтобы избежать избыточной информации на ER-диаграмме
Объектно-ориентированная	Да	Да	<ul style="list-style-type: none"> Добавляет семантическое наполнение В визуальное представление включается семантическое наполнение Наследование обеспечивает целостность базы данных 	<ul style="list-style-type: none"> Недостаточная стандартизация Сложная навигационная система доступа к данным Сложность обучения Повышенные требования к системе приводят в общем случае к замедлению выполнения транзакций

Примечание

Все модели предполагают использование единого хранилища данных внутри базы данных. Поэтому все модели базы данных обеспечивают совместное использование информации, потенциально устраняя таким образом, по крайней мере, проблему "информационных островков".

Основные термины

Администратор базы данных, DB-администратор — database administrator (DBA)

Американский национальный институт стандартизации — American National Standards Institute (ANSI)

Аномалии данных — data anomaly

Атрибут — attribute

База данных поддержки решений — decision support database

База данных предприятия — enterprise database

База данных рабочей группы — workgroup database

База данных, БД — database (DB)

Данные — data

Диаграмма "сущность-связь", ER-диаграмма — entity relationship diagram (ERD)

Зависимость по данным — data dependence

Запись — record

Запись-владелец в сетевой БД — owner, network database

Запись-член в сетевой БД — member, network database

Запрос — query

Иерархическая модель базы данных — hierarchical database model

Иерархическая последовательность — hierarchic sequence

Иерархическая структура — hierarchical structure

Иерархический маршрут — hierarchical path

Иерархия классов — class hierarchy

Избыточность данных — data redundancy

Избыточные данные — redundant data

Информационная управляющая система фирмы IBM — Information Management System (IMS)

Информационное хранилище — data warehouse

Информационные "островки" — islands of information

Класс — class

Комитет по планированию выпуска стандартов и технических условий — Standards Planning and Requirements Committee (SPARC)

Конференция по языкам обработки данных — Conference On Data Systems Language (CODASYL)

Концептуальная модель — conceptual model

Корень, корневой каталог — root

Логический формат данных — data logical format

Логическое проектирование — logical design

Метаданные — metadata

Метод — method

Многопользовательская СУБД — multi-user DBMS

Множество владделец-член — set owner/member

Модель "птичья лапка" — Crow's Foot model

Модель "сущность-связь", ER-модель — entity relationship(E-R model)

Модель — model

Модель базы данных — database model

Модель реализации — implementation model

Набор сущностей — entity set

Наследование — inheritance

Настольная база данных — desktop database

Независимость данных — data independence

Нерегламентированный запрос — ad hoc query

Обобщенный метод доступа и модификации — Generalized Update Access Method (GUAM)

Объект — object

Объектно-ориентированная модель базы данных — Object-Oriented DataBase Model (OODBM)

Объектно-ориентированная модель данных — Object-Oriented Data Model (OODM)

Объектно-реляционная модель базы данных — Object/Relational DataBase Model (O/RDM)

Однопользовательская СУБД — single-user DBMS

Операционная база данных — operational database

Отношение — relation

Подсхема — subschema

Поле — field

Проектирование базы данных — database design

Противоречивость данных — data inconsistency

Прямой обход — preorder traversal

Рабочая база данных — production database

Рабочая группа по базам данных — Data Base Task Group (DBTG)

Распределенная СУБД — distributed DBMS

Расширенная реляционная модель данных — Extended Relational Data Model (ERDM)

Реляционная схема — relational schema

Руководитель отдела обработки данных, DP-менеджер — data processing manager

Связность — connectivity

Связь — relationship

Сегмент (предок, потомок) — segment (parent, child)

- Семантическая модель данных — semantic data model (SDM)
- Система базы данных — database system
- Система поддержки решений — decision support system (DSS)
- Система управления базой данных, СУБД — database management system (DBMS)
- Система управления объектно-ориентированной базой данных — Object-Oriented DataBase Management System (OODBMS)
- Система управления объектно-реляционной базой данных — Object/Relational DataBase Management System (O/RDBMS)
- Система управления реляционной базой данных, реляционная система управления базой данных, РСУБД — Relational DataBase Management System (RDBMS)
- Система файлов, структура файлов — file system
- Системный администратор — systems administrator
- Словарь данных — data dictionary
- Специалист по обработке данных, DP-специалист — data processing specialist
- Структурированные данные — information
- Структурная зависимость — structural dependence
- Структурная независимость — structural independence
- Сущность — entity
- Схема — schema
- Таблица — table
- Транзакционная СУБД — transactional DBMS
- Указатель — pointer
- Универсальный сервер баз данных — universal database server
- Управление данными — data management
- Файл — file
- Физический формат данных — data physical format
- Целостность данных — data integrity
- Централизованная СУБД — centralized DBMS
- Экземпляр сущности — entity instance
- Язык запросов — query language
- Язык манипулирования данными — data manipulation language (DML)
- Язык определения данных — data definition language (DDL)
- Язык структурированных запросов — Structural Query Language (SQL)
- Язык третьего поколения — third-generation language (3GL)
- Язык четвертого поколения — fourth-generation language (4GL)

Вопросы

1. Определите каждый из этих терминов:
 - данные
 - поле
 - запись
 - файл
2. Что такое избыточность данных и какие свойства системы файлов являются ее причиной?
3. Говорят, что система файлов не обеспечивает достаточной независимости по данным. Поясните.
4. Что такое СУБД и каковы ее функции?
5. Как в иерархической базе данных решается проблема избыточности данных?
6. Что означает каждая из перечисленных ниже аббревиатур, и какое отношение они имеют к появлению сетевой модели базы данных?
 - CODASYL
 - SPARC
 - ANSI
 - DBTG
7. Какие три языка были приняты DBTG для стандартизации основной сетевой модели базы данных и почему стандартизация так важна для проектировщиков и пользователей?
8. Что такое независимость по данным и почему так важно ее обеспечить?
9. Опишите основные возможности реляционной модели и поясните их значение для конечных пользователей и проектировщиков.
10. Объясните, каким образом модель "сущность-связь" (ER-модель) помогает создать более структурированную конфигурацию для проектирования реляционных баз данных.
11. Используйте сценарий "Клиент может сделать несколько покупок, но каждый платеж делается только одним клиентом" в качестве основы для построения диаграммы "сущность-связь" (ER-диаграммы).
12. Что такое связность и какую роль она играет в проектировании БД?
13. Почему считается, что объект имеет большее семантическое наполнение, чем сущность?
14. В чем отличие между объектом и классом в объектно-ориентированной модели данных?
15. Как можно смоделировать вопрос № 11 с помощью OODM? (Используйте рис. 1.15.)
16. Что такое ERDM и какова ее роль в современных (рабочих или операционных) базах данных?

17. Сравните систему файлов с пятью системами баз данных, рассмотренными в этой главе, в смысле независимости по данным и структурной независимости.
18. Объясните разницу между неструктурированной и структурированной информацией.

Задачи

При решении задач 1—6 используйте файловую структуру, представленную на рис. 1.17.

	PROJECT_CODE	PROJECT_MANAGER	MANAGER_PHONE	MANAGER_ADDRESS	PROJECT_BID_PRICE
▶	1-52	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	\$16,833,460.00
	25-2D	Jane O. Grant	615-698-9909	218 Clark Blvd., Nashville, TN 36362	\$12,500,000.00
	25-5A	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	\$32,512,420.00
	25-9T	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	\$21,563,234.00
	27-4Q	George F. Dorts	615-227-1245	124 River Dr., Franklin, TN 29185	\$10,314,545.00
	29-2D	Holly B. Parker	904-338-3416	3334 Lee Rd., Gainesville, FL 37123	\$25,559,999.00
	31-7P	William K. Moor	904-445-2719	216 Morton Rd., Stetson, FL 30155	\$56,850,000.00

Рис. 1.17. Файловая структура для задач 1—6

1. Сколько записей содержит этот файл и сколько полей содержит запись?
2. С какими проблемами можно столкнуться, если потребуется напечатать список городов? Как бы вы решили эти проблемы, изменив структуру файла?
3. Если необходимо получить распечатку содержимого файла по фамилии, коду, городу, штату или почтовому индексу, то как стоило бы изменить структуру файла?
4. Какую избыточность по данным вы можете обнаружить, и к каким аномалиям в данных может привести эта избыточность?
5. Используя две таблицы реляционной базы PROJECT и MANAGER, устраните избыточность, обнаруженную в задаче № 4. Убедитесь, что вы используете соглашение об именах, обсуждавшееся в разд. 1.3.3, и соедините две таблицы соответствующими связями. (Подсказка: используйте рис. 1.11 в качестве примера.)
6. Создайте реляционную схему для демонстрации связи двух таблиц базы данных в задаче № 5.

При решении задач 7—13 используйте файловую структуру, представленную на рис. 1.18.

7. Выявите и объясните серьезную проблему избыточности данных в структуре, представленной на рис. 1.18.
8. Сколько различных источников данных по всей вероятности необходимо использовать для файла при решении задачи № 7?
9. Как на основе результатов задач № 7 и № 8 с помощью конфигурации реляционной базы данных устранить проблему избыточности данных?
10. На основе решения задачи № 9, определите, сколько таблиц необходимо использовать для окончательного устранения проблемы избыточности данных. Какую бы структуру таблиц вы рекомендовали?

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CODE	JOB_CHG_HOUR	PROJ_HOURS	EMP_PHONE
1	Hurricane	101	John D. Newson	EE	\$85.00	13.3	653-234-3245
1	Hurricane	105	David F. Schwann	CT	\$60.00	16.2	653-234-1123
1	Hurricane	110	Anne R. Ramoras	CT	\$60.00	14.3	615-233-5568
2	Coast	101	John D. Newson	EE	\$85.00	19.8	653-234-3254
2	Coast	108	June H. Sattlemir	EE	\$85.00	17.5	905-554-7812
3	Satellite	110	Anne R. Ramoras	CT	\$62.00	11.6	615-233-5568
3	Satellite	105	David F. Schwann	CT	\$26.00	23.4	653-234-1123
3	Satellite	123	Mary D. Chen	EE	\$85.00	19.1	615-233-5432
3	Satellite	112	Allecia R. Smith	BE	\$85.00	20.7	615-678-6879

Рис. 1.18. Файловая структура для задач 7—13

- Используя решение задачи № 10, продемонстрируйте содержимое каждой таблицы.
- Определите типы связей (1:1, 1:M, M:N) между таблицами, определенными в задачах № 10 и № 11. (Примечание: Представьте информацию о связях в форме диаграмм, как показано в разд. 1.5.)
- Создайте реляционную схему для задачи № 12, как показано на рис. 1.11.
- Исследуйте структуру таблиц, представленную на рис. 1.19, и определите, какие проблемы могут возникнуть при удалении поля KOM.

BUILDING_CODE	ROOM_CODE	TEACHER_LNAME	TEACHER_FNAME	TEACHER_INITIAL
KOM	204E	Williston	Horace	G
KOM	123	Cordoza	Maria	L
LDB	504	Patroski	Donald	J
KOM	34	Hawkins	Anne	W
JKP	225B	Risell	James	
LDB	301	Robertson	Jeanette	P

Рис. 1.19. Структура таблицы для задачи № 14

- Используя иерархическое представление, приведенное на рис. 1.20, выполните следующие задания:
 - определите типы сегментов;
 - определите компоненты, эквивалентные полям в системе файлов;
 - опишите иерархический маршрут к третьему сегменту PAINTING.
- Сетевая диаграмма, представленная на рис. 1.21, отображает единственную запись для пациента Judy D. Johanssen. Обычно пациент, находясь в больнице, получает лекарственные препараты, которые ему назначает лечащий врач. Поскольку пациент часто получает несколько препаратов в день, между PATIENT (Пациент) и ORDER (Заказ) существует связь 1:M. Точно так же каждое назначение может включать несколько препаратов, т. е. между ORDER (Заказ) и MEDICATION (Препарат) существует связь 1:M.

Используя структуру, представленную на рис. 1.21:

- определите тип сегментов;
- определите компоненты, эквивалентные полям в системе файлов;
- опишите иерархический маршрут для второго сегмента MEDICATION.

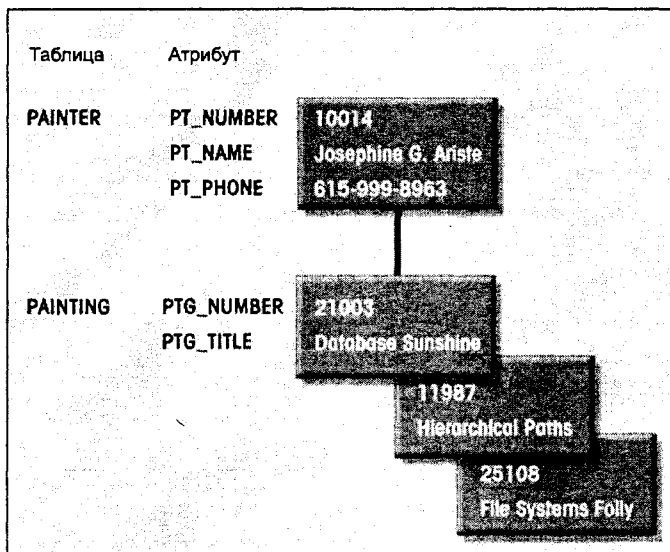


Рис. 1.20. Иерархическая структура для задачи № 15

- Расширьте модель, приведенную в задаче № 16, включив в нее сегмент DOCTOR (Врач), затем опишите ее иерархическую структуру (определите все сегменты). (Подсказка: у пациента может быть несколько лечащих врачей, но пациент с именем Judy B. Johanssen встречается только один раз в записях каждого врача).
- Предположим вам необходимо создать отчет, который показывает:
 - пациентов, которых лечит каждый доктор;
 - всех докторов, которые лечат каждого из пациентов.

Оцените иерархическую структуру, которую вы создали в задаче № 17, с точки зрения эффективности создания такого отчета.

- Компания PYRAID желает проследить каждую деталь (PART), используемую в каждом отдельном блоке оборудования (EQUIPMENT); каждая деталь покупается у конкретного поставщика (SUPPLIER). Используя это описание, нарисуйте сетевую структуру и определите множества для базы данных компании PYRAID. (Подсказка: блок оборудования состоит из многих деталей, но каждая деталь используется только в одном конкретном узле оборудования. Поставщик может поставлять несколько деталей, но каждая деталь поставляется только одним поставщиком.)

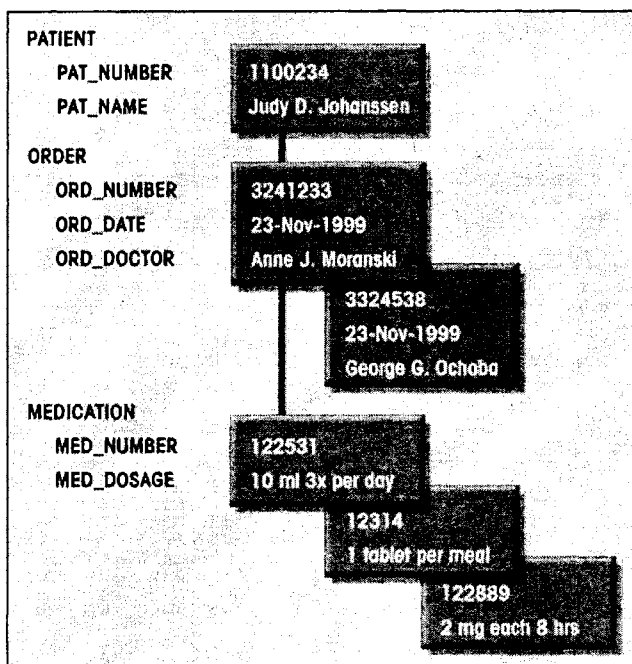


Рис. 1.21. Сетевая структура для задачи № 16

20. Компания United Broke Artists (UBA — объединение нищих художников) торгует не-очень-выдающимися картинами. UBA поддерживает небольшую сетевую базу данных для отслеживания художников, картин и выставок. Используя компоненты PAINTER (Художник), PAINTING (Картина) и GALLERY (Выставка), опишите сетевую структуру и определите подходящие множества для базы данных UBA. (*Подсказка 1:* картину пишет один художник, и данная картина представлена на конкретной выставке. *Подсказка 2:* на выставке может быть представлено много картин, но каждая картина может быть представлена только на одной выставке; картина написана одним художником, но каждый художник может написать несколько картин.)
21. Если вы решили конвертировать сетевую базу данных задачи № 20 в реляционную БД:
 - какие таблицы необходимо создать и какие компоненты будут содержать эти таблицы?
 - каким образом таблицы (независимые) будут связаны друг с другом?
22. Конвертируйте сетевую модель базы данных, представленную на рис. 1.10, в проект реляционной модели с помощью ER-диаграммы. Покажите все сущности, связи и связности.
23. Используя ER-диаграмму из задачи № 22, создайте реляционную схему. (Создайте подходящую коллекцию атрибутов для каждой сущности. Убедитесь, что вы используете подходящее соглашение об именовании атрибутов!)

24. Конвертируйте ER-диаграмму из задачи № 22 в базовую модель OODM.
25. Конвертируйте ER-диаграмму из задачи № 22 в модель "птичья лапка".
26. Опишите отношения, представленные на ER-диаграмме (рис. 1.22).

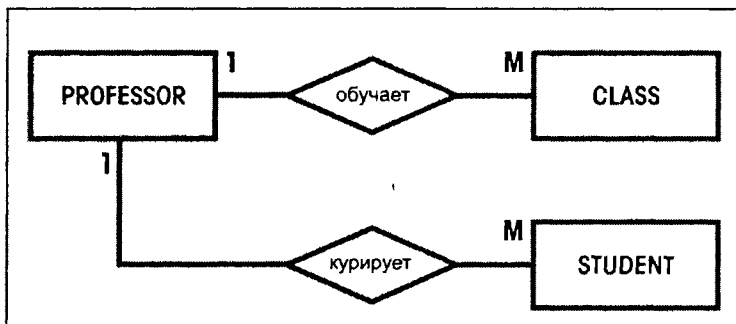


Рис. 1.22. ER-диаграмма для задачи № 26

27. Конвертируйте ER-диаграмму из задачи № 26 в модель "птичья лапка".
28. Опишите связи, представленные на ER-диаграмме (рис. 1.23).

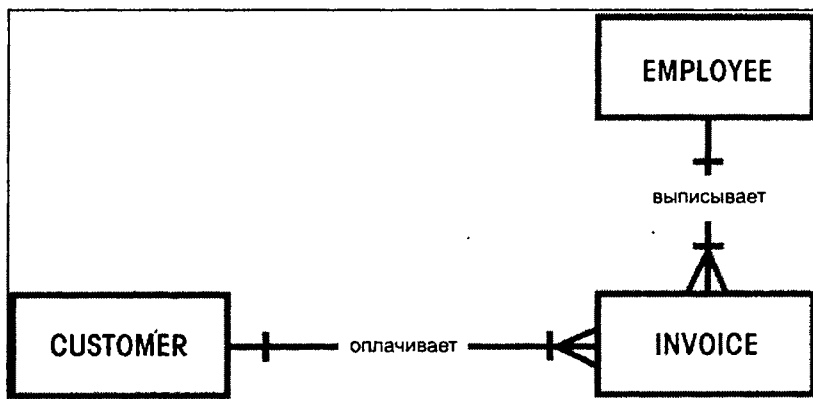


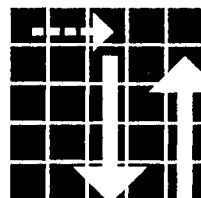
Рис. 1.23. ER-диаграмма для задачи № 28

29. Конвертируйте ER-диаграмму из задачи № 28 в модель Чена.
30. Создайте ER-диаграмму Чена и модель "птичья лапка" для каждого из следующих описаний. (Примечание: в среде баз данных слово "много" означает "больше чем один").)
 - Каждое подразделение корпорации MegaCo состоит из нескольких отделов. В каждом отделе работает несколько сотрудников, но каждый сотрудник ра-

ботает только в одном отделе. Каждым отделом руководит один сотрудник, и каждый из этих руководителей в настоящий момент времени может управлять только одним отделом.

- В течение некоторого периода времени клиент может взять напрокат несколько видеокассет в магазине BigVid. Каждую из кассет магазин BigVid может давать напрокат многим клиентам в течение какого-то периода времени.
- Авиалайнер может выполнять несколько рейсов, но каждый рейс выполняется только одним самолетом.
- Корпорация KwikTite владеет несколькими заводами. Рабочие каждого завода проживают в некоторой области. Каждая область может быть "домом" для многих рабочих заводов KwikTite. Всякую работу на данном заводе выполняет много рабочих, но каждый рабочий работает только на одной фабрике.
- Рабочий может иметь несколько разных квалификаций, и по каждой квалификации может иметься несколько рабочих.

Глава 2



Реляционная модель базы данных

В этой главе мы обсудим:

- ☐ отображение логического представления данных в реляционной модели;
- ☐ основные компоненты реляционной модели базы данных: сущности, атрибуты и связи между сущностями;
- ☐ каким образом сущности и их атрибуты организуются в таблицы;
- ☐ операторы реляционной базы данных, словарь данных и системный каталог;
- ☐ каким образом решается проблема избыточности данных в реляционной модели баз данных;
- ☐ почему важно индексирование.

Обзор

В гл. 1 было показано, что структурная независимость реляционной базы данных и ее независимость по данным позволяют исследовать логическую структуру модели без обращения к физическим аспектам хранения и извлечения данных. В этой главе мы рассмотрим некоторые важные детали логической структуры реляционной модели.

Мы покажем, что основными компонентами данных в реляционной модели БД являются сущности (логические объекты) и их атрибуты, и вы увидите, каким образом они встраиваются в логическую конструкцию, называемую таблицей. Вы убедитесь, что одна из самых важных причин простоты реляционной модели базы данных состоит в том, что ее таблицы могут рассматриваться как логические, а не как физические объекты. Мы расскажем, как независимые таблицы внутри базы данных можно связывать друг с другом.

После изучения таблиц, их компонентов и связей, вам будут представлены основные концепции проектирования таблиц. Поскольку таблица является неотъемлемой частью проекта реляционной базы данных, мы покажем, по каким признакам можно отличить хорошо спроектированные таблицы от плохо спроектированных.

Наконец, вам будут представлены некоторые базовые понятия, которые являются ключевыми для изучения последующих глав. Например, мы исследуем различные виды связей и то, каким образом можно управлять такими связями в среде реляционных баз данных.

2.1. Логическое представление данных

В гл. 1 было показано, что в базе данных хранятся и обрабатываются как данные, так и метаданные. Вы также убедились в том, что доступом к данным и к структуре базы данных управляет СУБД. Такая конструкция — размещение СУБД между приложением и базой данных — устраняет большую часть проблем, унаследованных от систем файлов. За такую гибкую схему приходится платить усложнением физической организации. Фактически структура базы данных, которая требуется для иерархических и сетевых моделей, зачастую настолько сложна, что уменьшает эффективность проекта БД в целом. *Реляционная модель базы данных* изменила такое положение дел, дав возможность проектировщику сосредоточиться на логическом представлении данных и их связей, а не на физических особенностях хранения данных. В реляционной базе данных как бы используется автоматическая коробка передач, избавляющая нас от необходимости пользоваться педалью сцепления и рычагом переключения передач. Говоря кратко, реляционная модель позволяет рассматривать данные с логической точки зрения, а не с физической.

На практике логическое представление данных удобно тем, что оно очень напоминает простую концепцию хранения данных в файле. Несмотря на то что таблица, в отличие от файла, имеет преимущества независимости по данным и структурной независимости, действительно, с концептуальной точки зрения таблица напоминает файл. Поскольку мы можем полагать, что связанные записи хранятся в независимых таблицах, реляционная модель проще для понимания, чем предшествующие ей иерархическая и сетевая модели. Логическая простота позволяет обеспечить простую и эффективную методологию проектирования.

Поскольку таблица играет такую выдающуюся роль в реляционной модели, она заслуживает самого тщательного изучения. Поэтому мы начнем наше обсуждение с детального исследования структуры таблицы и ее содержимого.

2.1.1. Сущности и атрибуты

Реляционное проектирование начинается с определения необходимых сущностей. Вспомните гл. 1, где определялось, что сущность это персона, местоположение или предмет, сведения о которых подлежат сбору и хранению. Например, в университете интересующими нас сущностями могут быть студенты, преподаватели, курсы и т. д. Если вы работаете в авиакомпании, сущностями здесь могут считаться пилоты, маршруты, оборудование и любое число дополнительных объектов, о которых необходимо собирать информацию.

Сущности группируются по их общим свойствам. Например, все студенты колледжа группируются вместе, формируя набор сущностей. *Набор сущностей* (entity set) это именованная совокупность сущностей, объединенных общими свойствами. В идеальном случае имя набора сущностей отражает его содержимое, чтобы напоминать проектировщику базы данных о своем функциональном предназначении внутри БД. Например, сведения о студентах университета Gigantic State University могут храниться в наборе сущностей с именем STUDENT. Другими словами, набор сущностей STUDENT (Студент) содержит множество сущностей "студент". Подобным образом сведения о преподавателях могут храниться в наборе сущностей FACULTY

(Профессорско-преподавательский состав), а сведения об авиалайнерах могут храниться в наборе сущностей с именем AIRCRAFT (авиалайнеры).

Каждая сущность имеет некоторые свойства, называемые *атрибутами*. Например, сущность STUDENT может иметь следующие атрибуты: идентификационный номер, средний балл (GPA, Grade Point Average), дата поступления, дата рождения, домашний адрес, номер телефона и профилирующая дисциплина. Подобным образом авиакомпания может определить такие атрибуты для сущности AIRCRAFT: бортовой номер, дата последнего технического осмотра, общий налет в часах и налет в часах со времени последнего технического осмотра.

Каждый атрибут нужно именовать надлежащим образом, чтобы его имя напоминало пользователю о содержимом атрибута. Например, в сущности STUDENT атрибут студента "дата рождения" (date of birth) может храниться под именем STU_DOB, а атрибут студента "домашний телефон" (home phone number) может храниться под именем STU_HOME_PHONE. Точно так же для сущности AIRCRAFT (авиалайнер) атрибут авиалайнера "бортовой номер" (aircraft number) может храниться под именем AC_NUMBER, а атрибут "общий налет в часах" (hours flown) может храниться под именем AC_HRS_FLOWN.

2.1.2. Таблицы и их свойства

Логическое представление реляционных баз данных упрощается созданием связей между данными на основе (логической) конструкции, называемой таблицей. Под *таблицей* понимается двумерная структура, состоящая из строк и столбцов. Пользователь должен понимать, что *таблица содержит группу связанных сущностей*, т. е. набор сущностей; по этой причине термины *набор сущностей* и *таблица* чаще всего означают одно и то же. Таблица также называется *отношением (relation)*, поскольку создатель реляционной модели Э. Ф. Кодд (E. F. Codd) использовал термин "отношение" как синоним слова "таблица".

Примечание

Термин "отношение", также известный в Microsoft Access как *dataset* (набор данных), пришел из математической теории множеств, которую Кодд использовал для создания своей модели. Поскольку реляционная модель основана на соединениях (links), позволяющих использовать связи (relationships) сущностей, многие пользователи баз данных неправильно думают, что термин "отношение" (relation) применяется к таким связям. Многие несправедливо полагают, что только реляционная модель допускает использование связей.

Оказывается, что табличное представление данных помогает определить связи сущностей, значительно упрощая задачу проектирования БД.

Свойства реляционной таблицы сведены в табл. 2.1

Таблица 2.1. Свойства реляционной таблицы

№	Свойство
1	Таблица представляет собой двумерную структуру, состоящую из строк и столбцов
2	Каждая строка таблицы (<i>кортеж</i> , tuple) представляет собой отдельную сущность внутри набора сущностей

Таблица 2.1 (окончание)

№	Свойство
3	Каждый столбец таблицы представляет собой атрибут, и у каждого столбца есть свое имя
4	На каждом пересечении строки и столбца имеется единственное значение
5	Каждая таблица должна иметь атрибут или несколько атрибутов, уникально идентифицирующих каждую строку
6	Все значения в столбце должны отображаться в одинаковом формате. Например, если атрибуту присваивается формат целого, то все значения в столбце, представляющем данный атрибут, должны быть целыми
7	Каждый столбец имеет определенный диапазон значений, называемый <i>доменом атрибута</i> (attribute domain)
8	Порядок следования строк и столбцов для СУБД не существует

Примечание

В реляционных базах данных используется строгая терминология. К сожалению, в среду баз данных иногда проникает терминология систем файлов. Таким образом, мы можем обнаружить, что строки иногда называют *записями*, а столбцы *полями*. Подчас таблицы называют файлами. С технической точки зрения, подмена терминов не всегда уместна: таблица базы данных представляет собой логическое, а не физическое понятие, а термины "файл", "запись" и "поле" описывают физические понятия. Тем не менее, учитывая, что таблица это логическое, а не физическое понятие, вы можете (на концептуальном уровне) считать строки таблицы записями, а столбцы полями. На самом деле, многие поставщики программного обеспечения до сих пор используют привычную терминологию систем файлов.

Для иллюстрации свойств, перечисленных в табл. 2.1, мы будем использовать таблицы, представленные на рис. 2.1.

С помощью таблицы STUDENT, представленной на рис. 2.1, мы можем сделать следующие выводы в соответствии со свойствами таблиц, представленными в табл. 2.1.

- Таблица STUDENT (см. рис. 2.1) представляет собой двумерную структуру, составленную из восьми строк (кортежей) и двенадцати столбцов. Можно также считать, что таблица состоит из восьми записей и двенадцати атрибутов (полей).
- Каждая строка в таблице STUDENT описывает отдельную сущность из имеющегося набора сущностей (набор сущностей представлен таблицей STUDENT). Обратите внимание, что строка (запись, или сущность), определенная как STU_NUM=321452, задает свойства (атрибуты, или поля) студента по имени William C. Bowser. Например, строка 4 на рис. 2.1 описывает студента по имени Walter H. Oblonski. Точно так же строка 3 описывает студентку по имени Juliette Brewer. Данная таблица содержит набор сущностей STUDENT, включающий восемь отдельных сущностей (строк).
- Каждый столбец представляет атрибут, и каждый столбец имеет свое имя.

Название базы данных: CH2_COLLEGE

Таблица: STUDENT

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS
321452	Bowser	William	C	Saturday, February 12, 1972	42	So
324257	Smithson	Anne	K	Tuesday, November 15, 1977	81	Jr
324258	Brewer	Juliette		Tuesday, August 23, 1966	36	So
324269	Oblonski	Walter	H	Sunday, September 16, 1973	66	Jr
324273	Smith	John	D	Friday, December 30, 1955	102	Sr
324274	Katinga	Raphael	P	Thursday, October 21, 1976	114	Sr
324291	Robertson	Gerald	T	Wednesday, April 08, 1970	120	Sr
324299	Smith	John	B	Wednesday, November 30, 1983	15	Fr

Таблица STUDENT
(продолжение)

STU_GPA	STU_TRANSFER	DEPT_CODE	STU_PHONE	PROF_NUM
2.84	No	BIOL	2134	205
3.27	Yes	CIS	2256	222
2.26	Yes	ACCT	2256	228
3.09	No	CIS	2114	222
2.11	Yes	ENGL	2231	199
3.15	No	ACCT	2267	228
3.87	No	EDU	2267	311
2.92	No	ACCT	2315	230

STU_HRS = Прослушанные часы
 STU_CLASS = Специализация
 STU_DOB = Дата рождения

STU_GPA = Средний балл
 STU_PHONE = 4-значный номер телефона в кампусе
 PROF_NUM = Номер преподавателя-куратора

Рис. 2.1. Список значений атрибутов таблицы STUDENT

Ограничения на имена таблиц и столбцов

В основном программное обеспечение современных баз данных может обрабатывать длинные информативные имена. Однако поскольку иногда необходимо "смешивать и подгонять" программное обеспечение в сложных конфигурациях баз данных, некоторые старые СУБД могут накладывать некоторые ограничения на имена:

- ☐ длина имени таблицы ограничивается восемью символами;
- ☐ длина имени столбца (атрибута) ограничивается десятью символами;
- ☐ имена столбцов не могут начинаться с цифры.

Вдобавок, иногда программное обеспечение баз данных накладывает ограничение на использование пробелов или специальных символов вроде дефиса (-), звездочки (*), точки (.) или слэша (/). Например, если Microsoft Access посчитает имя PRICE-DISCOUNT приемлемым, то в другом программном обеспечении БД такое имя может стать причиной сообщения об ошибке, а вот имя PRICE_DISCOUNT будет считаться корректным (обратите внимание на использование знака подчеркивания вместо дефиса). На самом деле некоторые СУБД интерпретируют PRICE-DISCOUNT как "вычитание значения атрибута DISCOUNT из значения атрибута PRICE". А имя атрибута FIRST.NAME может интерпретироваться как "атрибут NAME, расположенный в таблице с именем FIRST". Поэтому вам необходимо внимательно прочитать справочное руководство по СУБД для того, чтобы ознакомиться с имеющимися в ней ограничениями на имена.

□ На пересечении строк и столбцов содержится единственное значение. Данные должны классифицироваться в соответствии с их форматом и функциями. Хотя разные СУБД могут поддерживать различные типы данных, большинство из них поддерживает следующие типы.

- **Числовой.** К числовому типу данных относятся данные, над которыми вы можете выполнять различные арифметические действия. Например, на рис. 2.1 STU_HRS и STU_GPA являются атрибутами числового типа. С другой стороны, STU_PHONE не является числовым атрибутом, поскольку сложение или вычитание телефонных номеров не определено.
- **Символьный.** Данные символьного типа, также называемые текстовыми или строковыми данными, могут содержать любые буквы и символы и не предназначены для выполнения над ними математических операций. На рис. 2.1, например, STU_LNAME, STU_FNAME, STU_INIT, STU_CLASS и STU_PHONE являются атрибутами символьного типа (текстовыми или строковыми атрибутами).
- **Дата.** Атрибуты типа "дата" содержат календарные даты, хранящиеся в специальном формате, называемом юлианским форматом дат (число дней от начала года). Хотя физический способ хранения юлианской даты не имеет для пользователя никакого значения, тем не менее, он позволяет нам выполнять специальную разновидность арифметических операций (арифметика юлианского представления дат). Используя такую арифметику, мы можем определить число дней, прошедших между 5/12/1993¹ и 03/20/2002, просто вычтя 5/12/1993 из 03/20/2002. На рис. 2.1 STU_DOB классифицируется как атрибут типа "дата". Программное обеспечение большинства реляционных баз данных поддерживает юлианский формат даты. В то время как *внутренний* формат даты преимущественно является юлианским, допускаются различные формы *внешнего* представления даты. На рис. 2.1 даты STU_DB представлены в формате long date (длинном формате) MS Access. Если вы используете MS Access, вам может потребоваться представить дату рождения господина Bowser в более распространенном "среднем" формате (12-Feb-72) или в "коротком" формате (2/12/72). В большинстве реляционных СУБД, включая MS Access, можно даже определить собственный формат представления даты. Например, в Access и Oracle пользователи могут использовать формат даты DD-MMM-YYYY, чтобы первое значение атрибута STU_DUB выглядело как 12-FEB-1972.

Примечание

Атрибут типа "дата", как здесь описано, является примером абстрактного типа данных. Пока абстрактный тип данных может быть определен как набор схожих объектов (в нашем случае дат), которые имеют:

- внутреннее (encapsulated) представление данных. Специфика хранения скрыта от пользователя. Пользователю нет необходимости знать, хранятся ли такие данные

¹ В США принято обозначать дату в виде ММ/ЧЧ/ГГГГ в отличие от принятого в России обозначения ЧЧ.ММ.ГГ (где ММ — цифровое обозначение месяца, ЧЧ — цифровое обозначение числа месяца, а ГГ — последние две цифры года). — *Прим. пер.*

в виде строки из десяти символов (DD/MM/YYYY) или как целое число, определяющее количество дней, начиная с нулевого года;

- ❑ набор методов, т. е. допустимых операций. Для дат допустимыми операциями могут быть:

- сравнение двух дат;
- нахождение разности в днях между двумя датами;
- создание даты путем добавления (вычитания) заданного количества дней к данной дате;
- конвертирование даты из ее внутреннего представления во внешний формат, например, 20-Feb-2002 или 02/20/2002.

- *Логический*. Логический тип данных может принимать только два значения "истина" или "ложь" (true или false, yes или no, да или нет, 0 или 1). Например: закончил ли студент двухгодичный колледж? На рис. 2.1 STU_TRANSFER это атрибут логического типа. Большая часть (но не все) программного обеспечения реляционных баз данных поддерживает логический формат (в MS Access для логического типа данных используется обозначение "тип данных Yes/No").

- ❑ Каждая таблица должна иметь первичный ключ. В общих чертах *первичный ключ* (primary key) — это атрибут (или несколько атрибутов), уникально идентифицирующий данную сущность (строку). В нашем примере STU_NUM (идентификационный номер студента) выбран в качестве первичного ключа. Если внимательно присмотреться к данным, представленным на рис. 2.1, то можно увидеть, что фамилию (STU_LNAME) нельзя использовать в качестве первичного ключа, поскольку может встретиться несколько студентов с фамилией Smith. Даже комбинация фамилии (STU_LNAME) и имени (STU_FNAME) не подходит для первичного ключа, поскольку, как видно из рис. 2.1, вполне вероятно существование нескольких студентов с именем и фамилией John Smith.
- ❑ Все значения в столбце соответствуют свойствам атрибута сущности. Например, столбец средних оценок студента (STU_GPA) содержит только элементы STU_GPA для каждой из строк таблицы.
- ❑ Диапазон допустимых значений столбца называется *доменом* (domain). Поскольку значения STU_GPA ограничены диапазоном 0—4², то доменом этого столбца будет интервал значений [0,4].
- ❑ Порядок следования строк и столбцов не имеет значения для пользователя.

2.2. Ключи

Ключ состоит из одного или более атрибутов, определяющих другие атрибуты (например, номер счета идентифицирует все атрибуты счета, такие как дата выписки, имя клиента и т. д.). Один тип ключа, называемый *первичным ключом*, уже был определен. Структура таблицы STUDENT, представленная на рис. 2.1, в которой определен и описан первичный ключ, выглядит достаточно простой. Однако поскольку роль пер-

² Такова система оценок в высших учебных заведениях США. — Прим. пер.

вичного ключа в реляционной конфигурации столь значительна, мы подробно исследуем его свойства. Существует еще несколько типов ключей, заслуживающих внимания. В этом разделе мы узнаем, что представляют собой суперключ (superkey), потенциальный ключ (candidate key) и вторичный ключ (secondary key).

Роль ключа основана на концепции *определяемости* (determination). В контексте таблиц базы данных выражение "А определяет В" означает, что зная величину А, вы можете найти (определить) значение атрибута В. Например, если вы знаете STU_NUM в таблице STUDENT (см. рис. 2.1), то вы можете найти (определить) фамилию студента, средний балл, номер его телефона и т. д. Выражение "А определяет В" в сокращенном виде записывается как $A \rightarrow B$. Если А определяет В, С и D, то нужно писать $A \rightarrow B, C, D$.

Поэтому, используя атрибуты таблицы STUDENT (см. рис. 2.1), мы можем записать выражение "STU_NUM определяет STU_LNAME" в виде:

STU_NUM \rightarrow STU_LNAME

На самом деле, значение STU_NUM в таблице STUDENT определяет значения всех атрибутов студента. Например, можно записать:

STU_NUM \rightarrow STU_LNAME, STU_FNAME, STU_INIT

и

STU_NUM \rightarrow STU_LNAME, STU_FNAME, STU_INIT, STU_DOB,
STU_TRANSFER

И наоборот, атрибут STU_NUM не определяется атрибутом STU_LNAME, поскольку вполне вероятно, что имеется несколько студентов с фамилией Smith.

Принцип определяемости очень важен, поскольку он используется при формулировании центральной концепции реляционных баз данных — функциональной зависимости. *Функциональную зависимость* нестрого можно сформулировать так: *Атрибут В функционально зависит от атрибута А, если А определяет В*. Более точно:

Атрибут В функционально зависит от атрибута А, если каждое значение в столбце А определяет одно и только одно значение в столбце В.

Используя содержимое таблицы STUDENT (см. рис. 2.1), можно сказать, что STU_PHONE функционально зависит от STU_NUM. Например, значение STU_NUM, равное 321 452, определяет значение STU_PHONE, равное 2134. С другой стороны, STU_NUM не зависит функционально от STU_PHONE, поскольку значение STU_PHONE, равное 2267, ассоциируется с двумя значениями STU_NUM: 324 274 и 324 291 (очевидно, некоторые студенты пользуются одним телефоном). Подобным образом значение STU_NUM, равное 324 273, определяет значение STU_LNAME Smith. Но значение STU_NUM не зависит функционально от STU_LNAME, поскольку имя Smith могут иметь несколько студентов.

Определение функциональной зависимости должно быть обобщено таким образом, чтобы охватить те случаи, когда значение определяющего атрибута встречается в таблице более одного раза. Тогда функциональную зависимость можно определить так:

Атрибут А определяет атрибут В (т. е. В функционально зависит от А), если все строки в таблице, совпадающие по значению атрибута А, также совпадают по значению атрибута В.

Будьте осторожны при определении направления зависимости. Например, университет Gigantic State University классифицирует своих студентов так, как это показано в табл. 2.2.

Таблица 2.2. Классификация студентов

Количество прослушанных часов	Классификация
Менее 30	Fr (freshman — первокурсник)
30–59	So (sophomore, soph — второкурсник)
60–89	Jr (junior — предпоследний курс)
90 и более	Sr (senior student — старшекурсник)

Исходя из содержимого табл. 2.2 классификация студентов зависит от количества прослушанных ими часов. Поэтому мы можем написать:

STU_HRS → STU_CLASS

Но заданное количество часов не зависит от классификации. Вполне возможно найти студента предпоследнего курса (Jr), который прослушал 62 часа, и в то же время такого, у которого позади 84 часа. Другими словами, классификация (STU_CLASS) не определяет одно и только одно количество прослушанных часов (STU_HRS).

Следует иметь в виду, что может потребоваться более чем один атрибут для определения функциональной зависимости, т. е. ключ может состоять из более чем одного атрибута. Такие мультиатрибутные ключи называются *составными ключами* (*composite key*). Любой атрибут, который является частью составного ключа, называется *ключевым атрибутом* (*key attribute*). Например, в таблице STUDENT фамилия студента не может служить ключом. С другой стороны, комбинация фамилии, имени, инициалов и домашнего телефона, по всей вероятности, уникально определяет оставшиеся атрибуты. Например, мы можем написать:

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS

или

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS,
STU_GPA

или

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE → STU_HRS, STU_CLASS,
STU_GPA, STU_DOB

Учитывая возможность существования составного ключа, мы можем усовершенствовать определение функциональной зависимости и определить *полную функциональную зависимость*:

Если атрибут (B) функционально зависит от составного ключа (A), но не зависит ни от какого-либо подмножества этого составного ключа, то говорят, что атрибут (B) полностью функционально зависит от (A).

В рамках расширенной классификации ключей можно определить некоторые специальные ключи. Например, *суперключ* (*superkey*) — это ключ, уникально идентифици-

рующий каждую сущность. Иными словами, суперключ функционально определяет все атрибуты сущности. В структуре таблицы STUDENT суперключ может выглядеть так:

STU_NUM

STU_NUM, STU_LNAME

STU_NUM, STU_LNAME, STU_INIT

На самом деле, STU_NUM с дополнительными атрибутами или без них может считаться суперключом, даже если дополнительные атрибуты избыточны.

Потенциальный ключ (candidate key) можно определить как суперключ без избыточности. Используя этот признак, заметим, что составной ключ

STU_NUM, STU_LNAME

является суперключом, но не потенциальным ключом, поскольку STU_NUM сам по себе есть суперключ! Комбинация

STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE

может считаться потенциальным ключом, если вы не учитываете случай, когда два студента имеют одинаковые фамилии, имена и телефонные номера.

Если мы включим номер социального страхования студента в качестве одного из атрибутов таблицы STUDENT (см. рис. 2.1) — например, с именем STU_SOCSECNUM — как он, так и STU_NUM могут быть потенциальными ключами, поскольку и тот и другой однозначно идентифицируют студента. В этом случае выбор STU_NUM в качестве первичного ключа можно было оставить на усмотрение проектировщика или согласовать этот вопрос с заказчиком.

В пределах таблицы первичный ключ должен быть уникальным, чтобы однозначно идентифицировать каждую строку. Когда дело обстоит так, то говорят, что таблица проявляет *целостность на уровне сущности (entity integrity)*. Для обеспечения целостности на уровне сущности в первичном ключе недопустимы пустые значения, null (т. е. полное отсутствие данных).

Примечание

Пустое значение, null не есть ноль (zero) или пробел. Нажатие на клавиатуре клавиши <пробел> создает пропуск (пробел). Значение null создается в том случае, если вы нажимаете на клавиатуре клавишу <Enter> (Ввод) без предварительного ввода данных любого сорта.

Хотя пустых значений, кроме тех, что обнаружены в первичном ключе, не всегда можно избежать в остальных атрибутах, их надо использовать с осторожностью. Если значения null применяются некорректно, они могут создавать проблемы, поскольку в них может вкладываться различный смысл. Например, отсутствие данных (null) может означать:

- ☐ неизвестное значение атрибута;
- ☐ известное, но неверное значение атрибута;
- ☐ некорректное условие.

указывает на некую другую таблицу в базе данных. В нашем случае префикс VEND используется для указания на таблицу VENDOR в базе данных.

Связь между таблицами PRODUCT и VENDOR на рис. 2.2 может быть представлена реляционной схемой, приведенной на рис. 2.3.

Мы можем описать связь, исходя из того, что она создается, когда две таблицы совместно используют один атрибут с общим значением. Конкретнее, первичный ключ одной таблицы вновь появляется в качестве внешнего (foreign) ключа в связанной таблице. *Внешний ключ (foreign key)* это атрибут, значение которого совпадает со значением первичного ключа в связанной таблице. Например, на рис. 2.2 VEND_CODE — первичный ключ таблицы VENDOR, и он же выступает в роли внешнего ключа в таблице PRODUCT. Поскольку таблица VENDOR не связана с третьей таблицей, в ней отсутствует внешний ключ.

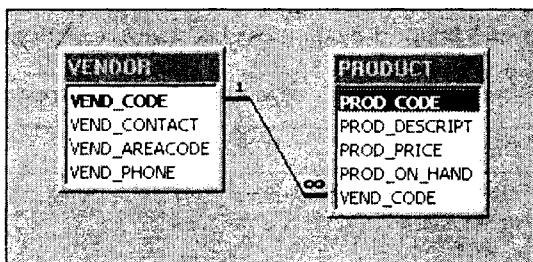


Рис. 2.3. Реляционная схема базы данных CH2_SALE_CO

Если внешний ключ содержит значения, совпадающие с первичным ключом или пустые значения (null), говорят, что таблица (или таблицы), использующая такой ключ, проявляет *целостность на уровне ссылки* (referential integrity). Другими словами, *целостность на уровне ссылки* означает, что в том случае, если внешний ключ содержит некое значение, то это значение ссылается на существующий действительный кортеж (строку) в другом отношении. Обратите внимание, что между таблицами PRODUCT и VENDOR, представленными на рис. 2.2, целостность на уровне ссылки обеспечивается.

Примечание

Помните, что пустое значение (null) не является значением в полном смысле. Например, на рис. 2.1 таблица STUDENT показывает, что значение STU_INIT для Juliette Brewer пусто (null), т. е. студентка не имеет инициала. Другими словами, пустое значение в этом случае означает отсутствие какого-либо значения. Необходимо помнить, что пустое значение (null) это не то же самое, что пробел.

Наконец, *вторичный ключ (secondary key)* определяется как ключ, который используется только для поиска данных. Предположим, данные клиента хранятся в таблице CUSTOMER, где в качестве первичного ключа используется идентификационный номер клиента. Вряд ли стоит рассчитывать, что клиенты помнят свой идентификационный номер. Однако необходимую информацию можно найти по фамилии кли-

ента и номеру его телефона. В данном случае в качестве первичного ключа используется идентификационный номер клиента, вторичный ключ — это комбинация фамилии клиента и его номера телефона. Нужно иметь в виду, что вторичный ключ не требует уникальности. Например, фамилия клиента и домашний телефон могут встретиться не один раз, если несколько членов семьи Smith живут в одном месте с одним телефонным номером. Точно так же комбинация фамилии и почтового индекса может привести к нескольким совпадениям, которые затем придется "разгрести" для поиска необходимой информации.

Эффективность вторичного ключа в сужении области поиска зависит от заданных вами ограничений. Например, хотя вторичный ключ CUS_CITY и можно использовать при поиске информации по клиенту, все же значения атрибута "New York" или "Sydney" вряд ли можно считать подходящим результатом, если только вы не хотите проверять миллионы возможных совпадений. (Конечно, в качестве вторичного ключа все же лучше использовать CUS_CITY, чем CUS_COUNTRY.)

Различные типы ключей таблиц реляционных баз данных приведены в табл. 2.3.

Таблица 2.3. Ключи реляционных баз данных

Тип ключа	Определение
Суперключ (superkey)	Атрибут (или комбинация атрибутов), уникально идентифицирующий каждую сущность в таблице
Потенциальный ключ (candidate key)	Минимальный суперключ. Суперключ, который не содержит подмножества атрибутов, которое само по себе является суперключом
Первичный ключ (primary key)	Потенциальный ключ, выбранный для уникальной идентификации всех остальных значений атрибутов в любой строке. НЕ может содержать пустых значений (null)
Вторичный ключ (secondary key)	Атрибут (или комбинация атрибутов), используемый исключительно в целях поиска данных
Внешний ключ (foreign key)	Атрибут (или комбинация атрибутов) в одной таблице, значения которого должны или совпадать со значениями первичного ключа в другой таблице, или быть пустыми (null)

2.3. Еще раз о правилах целостности

Впоследствии будет показано, что правила целостности реляционной базы данных очень важны для хорошего проекта БД. Многие (но далеко не все) системы управления реляционными базами данных (РСУБД) автоматически обеспечивают целостность данных. Однако куда проще и безопаснее убедиться самому, что проект приложения соответствует правилам целостностей на уровнях сущности и ссылки, о которых упоминалось ранее в этой главе. Эти правила представлены в табл. 2.4.

Таблица 2.4. Правила целостности

ЦЕЛОСТНОСТЬ НА УРОВНЕ СУЩНОСТИ	
Требование	Все элементы первичного ключа уникальны и никакая часть первичного ключа не может быть пустой (null)
Назначение	Гарантирует, что каждая сущность (логический объект) будет иметь уникальную идентификацию, а значения внешнего ключа могут должным образом ссылаться на значения первичного ключа
Пример	Счет не может иметь несколько дублирующих значений и не может иметь пустое значение (null). Короче говоря, все счета уникальны и идентифицируются своим номером
ЦЕЛОСТНОСТЬ НА УРОВНЕ ССЫЛКИ	
Требования	Внешний ключ может иметь или пустое значение (если только он не является частью первичного ключа данной таблицы), или значение, совпадающее со значением первичного ключа в связанной таблице. (Каждое непустое значение внешнего ключа должно ссылаться на существующее значение первичного ключа.)
Назначение	Допускается, что атрибут не имеет соответствующего значения, но атрибут не может принимать недопустимые значения. Выполнение правила целостности на уровне ссылки делает невозможным удаление строки в одной таблице, где первичный ключ имеет обязательное соответствие со значением внешнего ключа в другой таблице
Пример	Клиенту может быть не назначен (еще) торговый агент, но невозможно назначить клиенту несуществующего агента

Правила целостности, представленные в табл. 2.4, проиллюстрированы на рис. 2.4.

На рис. 2.4 обратите внимание на следующие особенности.

- ❑ *Целостность на уровне сущности.* Первичный ключ таблицы CUSTOMER — CUS_CODE. В столбце первичного ключа нет пустых значений и все его элементы уникальны. Точно так же первичным ключом таблицы AGENT служит атрибут AGENT_CODE, и в столбце первичного ключа также отсутствуют пустые элементы (null).
- ❑ *Целостность на уровне ссылки.* Таблица CUSTOMER содержит внешний ключ AGENT_CODE, который связывает элементы таблицы CUSTOMER с таблицей AGENT. Строка CUS_CODE, идентифицированная номером 10013 (первичный ключ), содержит пустой элемент во внешнем ключе AGENT_CODE, поскольку господину Paul F. Olowski еще не назначен торговый агент. Остальные элементы AGENT_CODE в таблице CUSTOMER соответствуют элементам AGENT_CODE в таблице AGENT.

Чтобы избежать пустых значений (null), некоторые проектировщики используют специальные коды, которые называются *флагами* (flags) для обозначения отсутствия значения. Используя в качестве примера рис. 2.4, можно использовать код -99 в качестве значения атрибута AGENT_CODE в четвертой строке таблицы CUSTOMER для указания на то, что покупателю Paul Olowski еще не назначен торговый агент.

Если используется такой флаг, таблица AGENT должна содержать фиктивную строку (фиктивного агента) со значением AGENT_CODE, равным -99. Таким образом, первая запись таблицы AGENT будет содержать значения, представленные в табл. 2.5.

Таблица: CUSTOMER								База данных: CH2_INSURE_CO	
Первичный ключ: CUST_CODE									
Внешний ключ: AGENT_CODE									
CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_RENEW_DATE	AGENT_CODE		
10010	Ramas	Alfred	A	615	844-2573	12-Mar-02	502		
10011	Dunne	Leona	K	713	894-1238	23-May-02	501		
10012	Smith	Kathy	W	615	894-2285	05-Jan-03	502		
10013	Olowski	Paul	F	615	894-2180	20-Sep-02			
10014	Orlando	Myron		615	222-1672	04-Dec-02	501		
10015	O'Brian	Amy	B	713	442-3381	29-Aug-02	503		
10016	Brown	James	G	615	297-1228	01-Mar-03	502		
10017	Williams	George		615	290-2556	23-Jun-02	503		
10018	Farriss	Anne	G	713	382-7185	09-Nov-02	501		
10019	Smith	Olette	K	615	297-3809	18-Feb-03	503		

Таблица: AGENT					
Первичный ключ: AGENT_CODE					
Внешний ключ: не задан					
AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SLS	
-99	713	226-1249	Alby	\$1,735,453.75	
502	615	892-1244	Hahn	\$4,967,003.28	
503	615	123-5569	Okon	\$3,093,980.41	

Рис. 2.4. Правила целостности

Таблица 2.5. Фиктивное значение, используемое в качестве флага

AGENT_CODE	AGENT_AREACODE	AGENT_PHONE	AGENT_LNAME	AGENT_YTD_SALES
-99	000	000-0000	None	\$0.00

В гл. 3 мы обсудим некоторые способы обработки пустых значений.

2.4. Реляционные операторы

Уровень соответствия требованиям к реляционным базам данных может определяться степенью поддержки реляционной алгебры. *Реляционная алгебра* определяет теоретические способы манипулирования содержимым таблиц с помощью восьми реляционных операторов: SELECT (выбор), PROJECT (проекция), JOIN (соединение), INTERSECT (пересечение), UNION (объединение), DIFFERENCE (разность), PRODUCT (произведение) и DIVIDE (деление). Впоследствии, в гл. 5 мы рассмотрим, как можно использовать команды языка SQL для выполнения реляционных процедур.

Примечание

Чтобы базу данных можно было считать реляционной хотя бы в самой малой степени, ее СУБД должна поддерживать основные реляционные операторы SELECT, PROJECT и JOIN. Очень немногие СУБД поддерживают все восемь реляционных операторов.

Хотя при использовании операторов реляционной алгебры над имеющимися таблицами теоретически создаются новые таблицы, в коммерческом программном обеспечении, как правило, при выполнении таких операторов создается только внешнее представление новых таблиц. Нет необходимости исследовать математические определения, свойства и характеристики операторов реляционной алгебры. Однако их использование можно проиллюстрировать примерами.

- **UNION** (объединение). Объединяет все строки из двух таблиц. Чтобы оператор UNION можно было использовать, таблицы должны иметь одинаковые свойства по атрибутам (столбцы и домены должны быть идентичны). Когда одна или более таблиц совместно используют одни и те же столбцы и домены, говорят, что они *совместимы по объединению* (*union-compatible*). Эффект выполнения оператора UNION показан на рис. 2.5

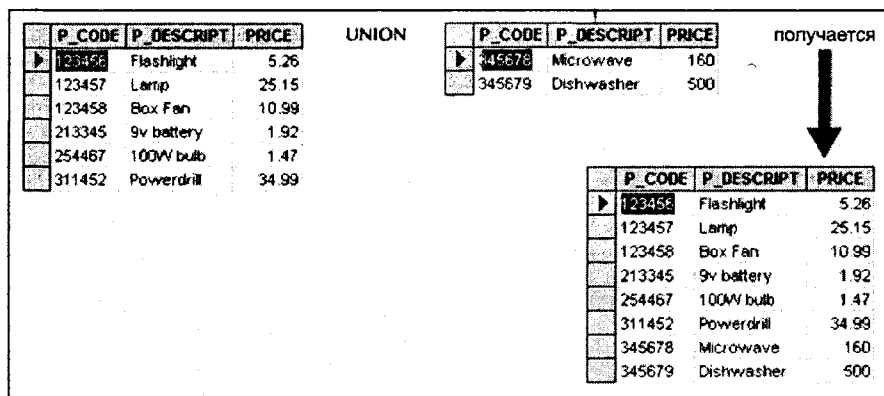


Рис. 2.5. UNION (объединение)

- **INTERSECT** (пересечение). Выводит только строки, которые встречаются в обеих таблицах. Таблицы должны быть совместимы по объединению. Например, вы не можете применить оператор INTERSECT, если один из атрибутов имеет числовой тип, а другой — символьный тип. Эффект применения оператора INTERSECT показан на рис. 2.6.
- **DIFFERENCE** (разность). Выводит все строки первой таблицы, которые отсутствуют в другой, т. е. производит вычитание одной таблицы из другой. Таблицы должны быть совместимы по объединению. Эффект применения оператора DIFFERENCE представлен на рис. 2.7.
- **PRODUCT** (произведение). Выводит все возможные пары строк из двух таблиц — этот оператор называется также *декартовым произведением*. Следовательно,

если в одной таблице имеется 6 строк, а в другой 3 строки, то оператор **PRODUCT** выдаст список, состоящий из $6 \times 3 = 18$ строк. Эффект применения оператора **PRODUCT** показан на рис. 2.8.

F_NAME	INTERSECT	F_NAME	получается.	F_NAME
George		Jane		Jane
Jane		William		Jorge
Elaine		Jorge		
Wilfred		Dennis		
Jorge				

Рис. 2.6. INTERSECT (пересечение)

F_NAME	DIFFERENCE	F_NAME	получается	F_NAME
George		Jane		George
Jane		William		Elaine
Elaine		Jorge		Wilfred
Wilfred		Dennis		
Jorge				

Рис. 2.7. DIFFERENCE (разность)

P_CODE	P_DESCRIPT	PRICE	PRODUCT			получается		
123456	Flashlight	5.26	STORE	aisle	shelf	↓		
123457	Lamp	25.15	23	W	5			
123458	Box Fan	10.99	24	K	9			
213345	9v battery	1.92	25	Z	6			
254467	100W bulb	1.47						
311452	Powerdrill	34.99						
P_CODE	P_DESCRIPT	PRICE	STORE	aisle	shelf			
123456	Flashlight	5.26	23	W	5			
123456	Flashlight	5.26	24	K	9			
123456	Flashlight	5.26	25	Z	6			
123457	Lamp	25.15	23	W	5			
123457	Lamp	25.15	24	K	9			
123457	Lamp	25.15	25	Z	6			
123458	Box Fan	10.99	23	W	5			
123458	Box Fan	10.99	24	K	9			
123458	Box Fan	10.99	25	Z	6			
213345	9v battery	1.92	23	W	5			
213345	9v battery	1.92	24	K	9			
213345	9v battery	1.92	25	Z	6			
311452	Powerdrill	34.99	23	W	5			
311452	Powerdrill	34.99	24	K	9			
311452	Powerdrill	34.99	25	Z	6			
254467	100W bulb	1.47	23	W	5			
254467	100W bulb	1.47	24	K	9			
254467	100W bulb	1.47	25	Z	6			

Рис. 2.8. PRODUCT (произведение)

- **SELECT** (выбор). Выводит значения всех строк, найденных в таблице. Оператор **SELECT** можно использовать для получения списка всех строк, либо для вывода только строк, соответствующих определенному критерию. Другими словами, оператор **SELECT** выводит горизонтальное подмножество таблицы. Эффект применения оператора **SELECT** представлен на рис. 2.9.

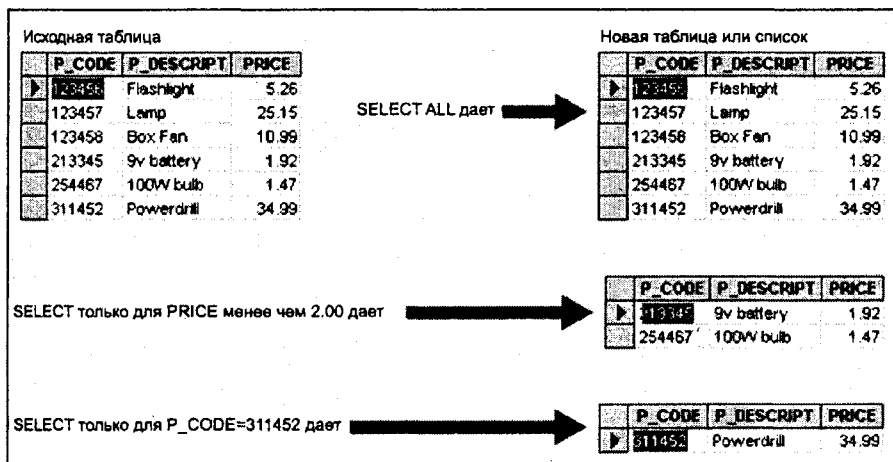


Рис. 2.9. SELECT (выбор)

- **PROJECT** (проекция). Выводит все значения выбранных атрибутов. Иначе говоря, оператор **PROJECT** выводит вертикальное подмножество таблицы. Эффект применения оператора **PROJECT** представлен на рис. 2.10.
- **JOIN** (соединение). Позволяет компоновать информацию из двух или более таблиц. Оператор **JOIN** обладает большими возможностями в среде реляционных баз данных, позволяя использовать независимые таблицы, связанные общими атрибутами. Для иллюстрации применения некоторых вариантов оператора **JOIN** мы будем использовать таблицы **CUSTOMER** и **AGENT**, представленные на рис. 2.11.

Естественное соединение (natural JOIN) связывает таблицы, выбирая только строки с общими значениями их общих атрибутов. Естественное соединение это результат трехступенчатой процедуры.

- Во-первых, применяется оператор **PRODUCT**, при этом получается результат, представленный на рис. 2.12.
- К полученному результату применяется оператор **SELECT**, который выводит только строки, где значения **AGENT_CODE** совпадают. Общие столбцы называются *столбцами соединения (join column)*. Результат этого шага представлен на рис. 2.13.

- К результату второго шага применяется оператор PROJECT для вывода единственной копии каждого атрибута, таким образом, устраняются дублирующиеся столбцы. Результат выполнения третьего шага представлен на рис. 2.14.

Окончательным результатом действия оператора естественного соединения является таблица, в которую не включены несовпадающие пары и где представлены только совпадающие копии.

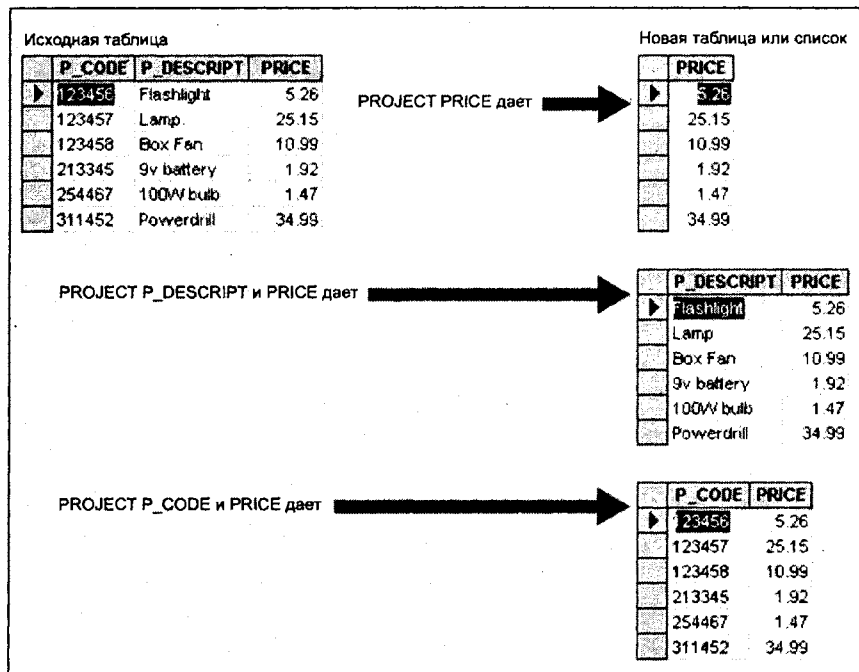


Рис. 2.10. PROJECT (проекция)

Таблица: CUSTOMER					Таблица: AGENT	
CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE		AGENT_CODE	AGENT_PHONE
1132445	Walker	32145	231		25	6152439887
1217782	Adares	32145	125		167	6153426778
1312243	Rakowski	34129	167		231	6152431124
1321242	Rodriguez	37134	125		333	9041234445
1542311	Smithson	37134	421			
1657399	Vanloo	32145	231			

Рис. 2.11. Таблицы для демонстрации применения оператора JOIN

	CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
▶	1132445	Walker	32145	231	125	6152439887
	1132445	Walker	32145	231	167	6153426778
	1132445	Walker	32145	231	231	6152431124
	1132445	Walker	32145	231	333	9041234445
	1217782	Adares	32145	125	125	6152439887
	1217782	Adares	32145	125	167	6153426778
	1217782	Adares	32145	125	231	6152431124
	1217782	Adares	32145	125	333	9041234445
	1312243	Rakowski	34129	167	125	6152439887
	1312243	Rakowski	34129	167	167	6153426778
	1312243	Rakowski	34129	167	231	6152431124
	1312243	Rakowski	34129	167	333	9041234445
	1321242	Rodriguez	37134	125	125	6152439887
	1321242	Rodriguez	37134	125	167	6153426778
	1321242	Rodriguez	37134	125	231	6152431124
	1321242	Rodriguez	37134	125	333	9041234445
	1542311	Smithson	37134	421	125	6152439887
	1542311	Smithson	37134	421	167	6153426778
	1542311	Smithson	37134	421	231	6152431124
	1542311	Smithson	37134	421	333	9041234445
	1657399	Vanloo	32145	231	125	6152439887
	1657399	Vanloo	32145	231	167	6153426778
	1657399	Vanloo	32145	231	231	6152431124
	1657399	Vanloo	32145	231	333	9041234445

Рис. 2.12. Естественное соединение,
шаг 1: PRODUCT

	CUS_CODE	CUS_LNAME	CUS_ZIP	CUSTOMER.AGENT_CODE	AGENT.AGENT_CODE	AGENT_PHONE
▶	1217782	Adares	32145	125	125	6152439887
	1321242	Rodriguez	37134	125	125	6152439887
	1312243	Rakowski	34129	167	167	6153426778
	1132445	Walker	32145	231	231	6152431124
	1657399	Vanloo	32145	231	231	6152431124

Рис. 2.13. Естественное соединение,
шаг 2: SELECT

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
▶	1217782	Adares	32145	125	6152439887
	1321242	Rodriguez	37134	125	6152439887
	1312243	Rakowski	34129	167	6153426778
	1132445	Walker	32145	231	6152431124
	1657399	Vanloo	32145	231	6152431124

Рис. 2.14. Естественное соединение,
шаг 3: PROJECT

Обратите внимание на некоторые важнейшие особенности естественного соединения:

- если не существует совпадений между строками таблицы, то в новую таблицу несовпадающие строки не включаются. В этом случае ни AGENT_CODE со значением 421, ни клиент по фамилии Smithson не войдут в результирующую таблицу. AGENT_CODE 421 для клиента Smithson не соответствует ни одному из элементов таблицы AGENT;
- столбец, над которым выполнялась операция JOIN, — т. е. AGENT_CODE, — в новой таблице встречается только один раз;
- если одно и то же значение AGENT_CODE в таблице AGENT встречается несколько раз, клиент будет выведен для каждого совпадения. Например, если значение AGENT_CODE, равное 167, встретилось в таблице AGENT три раза, то клиент по фамилии Rakowski, связанный с агентом, имеющим номер 167, тоже появится три раза в результирующей таблице (правильно организованная таблица AGENT не может, конечно, привести к такому результату, поскольку она должна содержать уникальные значения первичного ключа).

Другая форма оператора JOIN, которую называют *эквисоединением* (*equiJOIN*, *соединение по эквивалентности*), связывает таблицы по условию равенства, при котором сравниваются указанные столбцы каждой таблицы. В результирующем выводе оператора equiJOIN не удаляются дублирующиеся столбцы, а условие (критерий) соединения таблиц должно быть явно определено. Название оператора equiJOIN происходит от операции эквивалентного сравнения (=), используемой в условных операторах. Если используются другие операции сравнения, то соединение называется *тета-соединением* (*theta JOIN*).

Во *внешнем соединении* (*outer JOIN*) остаются совпадающие пары, а все несовпадающие значения в другой таблице должны принимать пустые значения (null). Конкретнее, если мы выполняем операцию внешнего соединения для таблиц CUSTOMER и AGENT, то возможны два сценария:

- левостороннее внешнее соединение (*left outer JOIN*) выводит все строки таблицы CUSTOMER, включая те, которые не имеют совпадающих значений в таблице AGENT. Пример использования такого оператора JOIN представлен на рис. 2.15;

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
►	1217782	Adares	32145	125	6152439887
	1321242	Rodriguez	37134	125	6152439887
	1312243	Rakovski	34129	167	6153426778
	1132445	Walker	32145	231	6152431124
	1657399	Vanloo	32145	231	6152431124
	1542311	Smithson	37134	421	

Рис. 2.15. Левостороннее внешнее соединение

- правостороннее внешнее соединение (*right outer JOIN*) выводит все строки таблицы AGENT, включая те, которые не имеют совпадающих значений в

таблице CUSTOMER. Пример использования такого оператора JOIN представлен на рис. 2.16.

	CUS_CODE	CUS_LNAME	CUS_ZIP	AGENT_CODE	AGENT_PHONE
▶	217782	Adares	32145	125	6152439887
	1321242	Rodriguez	37134	125	6152439887
	1312243	Rakowski	34129	167	6153426778
	1132445	Walker	32145	231	6152431124
	1657399	Vanloo	32145	231	6152431124
				333	9041234445

Рис. 2.16. Правостороннее внешнее соединение (left outer JOIN)

Внешние соединения особенно полезны при определении того, какие из значений в связанных таблицах вызывают проблемы целостности на уровне ссылки — какие из них были созданы, когда значения внешнего ключа не соответствовали значениям первичного ключа в связанных таблицах. На самом деле, если вам необходимо конвертировать крупноформатные таблицы или другую информацию, не упорядоченную в базы данных, в реляционные таблицы, то вы обнаружите, что внешнее соединение экономит массу времени и избавит вас от головной боли при разрешении проблем целостности на уровне ссылки после всех этих преобразований. Мы столько раз были благодарны внешнему соединению в процессе работы с базами данных!

Вы можете спросить, почему внешнее соединение бывает *левосторонним* и *правосторонним*. Это связано с порядком, в котором таблицы перечисляются в SQL-командах. Мы будем изучать такие соединения (JOIN) в гл. 5.

8. DIVIDE (деление). Требуется использования таблицы, состоящей из одного столбца, и второй таблицы, состоящей из двух столбцов. Используя пример рис. 2.17, отметим, что:

- Таблица 1 "разделена" на Таблицу 2. Обе таблицы 1 и 2 содержат столбец CODE, но не используют совместно атрибут LOC;

CODE	LOC	DIVIDE	CODE	получается	LOC
▶	5		▶		5
A	9				
A	4				
B	5				
B	3				
C	6				
D	7				
D	8				
E	8				

Рис. 2.17. DIVIDE (деление)

- будучи включенным в результирующую Таблицу 3, значение в столбце LOC (не используемом таблицами совместно) должно быть ассоциировано (в Таблице-делителе 2) с каждым значением в Таблице-делимом 1;
- единственное значение, ассоциированное как с А, так и с В, — это 5.

2.5. Словарь данных и системный каталог

Словарь данных используется для обеспечения детального учета всех таблиц, находящихся внутри базы данных, созданной пользователем или проектировщиком. Таким образом, словарь данных содержит (по крайней мере) все имена атрибутов и свойства каждой таблицы системы базы данных. Говоря кратко, в словаре данных содержатся метаданные (данные о данных). Используя небольшую базу данных, представленную на рис. 2.4, мы можем составить для нее словарь данных, приведенный в табл. 2.6.

Примечание

Словарь данных, представленный в табл. 2.6, является примером человеческого представления о сущностях, атрибутах и связях. Цель такого словаря данных состоит в том, чтобы все участники группы проектирования и реализации базы данных использовали одни и те же имена таблиц и атрибутов, а также их свойства. Внутренний словарь СУБД содержит дополнительную информацию о типах связей, проверке и обеспечении целостностей на уровнях ссылки и сущности, а также типы индексов и компонентов. Эта дополнительная информация создается в процессе реализации базы данных.

Примечание

Телефонный код города всегда состоит из цифр от 0 до 9. Поскольку код города не используется в арифметических операциях, его лучше всего хранить как символьный тип. К тому же телефонные коды городов всегда состоят из трех цифр. Поэтому для телефонного кода города определен тип CHAR(3). С другой стороны, имена людей могут и не соответствовать стандартно установленной длине. Поэтому имя клиента определяется как VARCHAR(20), т. е. для него можно использовать до 20 символов. Символьные данные отображаются выровненными по левому краю.

Словарь данных иногда называют "базой данных базы данных проектировщика", поскольку в него записываются проектные решения, касающиеся таблиц и их структур.

Системный каталог, как и словарь данных, содержит метаданные. Системный каталог можно считать очень подробным системным словарем данных, который описывает все объекты внутри БД, включая данные об именах таблиц, создателе таблицы и дате ее создания, количестве столбцов в каждой таблице, типе данных соответствующих столбцов, именах индексных файлов, создателе индекса, авторизованных пользователях, полномочиях доступа и т. д. Поскольку системный каталог содержит всю информацию, необходимую для словаря данных, термин "системный каталог" и "словарь данных" зачастую подменяют друг друга. На самом деле, современное программное обеспечение реляционных БД, как правило, сопровождается только системным каталогом, из которого можно извлечь информацию, касающуюся словаря данных проектировщика.

Таблица 2.6. Пример словаря данных

Имя таблицы	Имя атрибута	Содержимое	Тип	Формат	Диапазон	Обязатель- лен	РК или FK	Таблица на кото- рую ссы- ляется FK
CUSTOMER	CUS_CODE	Код учетной записи клиента	CHAR(5)	99999	10000--99999	Да	PK	
	CUS_LNAME	Фамилия клиента	VCHAR(20)	Xxxxxxx		Да		
	CUS_FNAME	Имя клиента	VCHAR(20)	Xxxxxxx		Да		
	CUS_INITIAL	Инициалы клиента	CHAR(1)	X				
	CUS_RENEW_DATE	Дата обновления страхового полиса клиента	DATE	DD_MM_YYYY				
AGENT	AGENT_CODE	Код страхового агента	CHAR(3)	999	100-999		FK	AGENT
	AGENT_CODE	Код страхового агента	CHAR(3)	999		Да	PK	
	AGENT_AREACODE	Телефонный код города агента	CHAR(3)	999		Да		
	AGENT_PHONE	Номер телефона агента	CHAR(8)	999-9999		Да		
	AGENT_LNAME	Фамилия агента	VCHAR(20)			Да		
	AGENT_YTD_SALES	Число продаж с начала года	NUMBER(9,2)	9,999,999.99	0.00-9,999,999.99	Да		

Примечание: FK — внешний ключ; PK — первичный ключ; CHAR — символьные данные фиксированной длины от 1 до 255 символов; VARCHAR (VCHAR) — символьные данные переменной длины (от 1 до 2000 символов); NUMBER — числовые данные; NUMBER(9,2) используется для отображения чисел с двумя знаками после запятой общим количеством до 9 знаков, включая десятичную точку. Некоторые СУБД допускают использование типа данных MONEY или CURRENCY.

Системный каталог фактически является базой данных, созданной системой, а в таблицах этой БД хранятся свойства и информация о базе данных, созданной пользователем или проектировщиком. Поэтому к таблицам системного каталога можно точно так же обращаться с запросами, как и к любым таблицам базы данных пользователя (проектировщика).

В сущности, системный каталог автоматически создает документацию по базе данных. По мере добавления в БД новых таблиц СУБД с помощью такой документации проверяет и устраняет омонимы и синонимы. Вообще говоря, *омонимы* это слова, схожие по звучанию, но разные по значению (омофоны), например "плот" и "плод", "прут" и "пруд", или слова одинакового написания, но имеющие разный смысл, например, "ключ" — родник, "ключ" от квартиры и "ключ" — последовательность знаков, используемая для идентификации записи. В контексте баз данных слово "омоним" обозначает использование одного имени для обозначения разных атрибутов. Например, мы можем написать C_NAME для обозначения названия компании в таблице CUSTOMER и также использовать C_NAME для обозначения названия поставщика в таблице INVENTORY (инвентарь). Чтобы избежать путаницы, необходимо стараться избегать омонимов в базе данных, и словарь данных в этом смысле может оказать неоценимую помощь.

В контексте баз данных *синоним* противоположен по значению омониму и подразумевает использование различных имен для обозначения одних и тех же атрибутов. Например, "машина" и "автомобиль" могут относиться к одному и тому же объекту. Синонимов тоже следует избегать. Поэтому, если один и тот же атрибут встречается в нескольких таблицах, то везде его лучше именовать одинаково. Другими словами, в идеальном случае внешний ключ должен иметь то же имя, что и первичный ключ в порождающей таблице. Вот почему на рис. 2.4 в качестве внешнего ключа в таблице CUSTOMER используется AGENT_CODE, и он же используется в качестве первичного ключа в таблице AGENT.

Примечание

С точки зрения реляционных баз данных, в идеальной таблице имя каждого внешнего ключа совпадает с именем первичного ключа в связанной таблице. Например, в таблице INVOICE может использоваться ключ CUS_CODE для связи каждого счета с таблицей CUSTOMER, в которой CUS_CODE является первичным ключом. Такая практика также желательна и с точки зрения документирования БД. Она к тому же помогает упростить некоторые стороны разработки приложений.

Однако реальные требования иногда вынуждают нас идти на компромисс. Например, некоторая авиакомпания назначает первого и второго пилотов, каждый из которых является сотрудником, на некоторый рейс. Хотя таблица EMPLOYEE (сотрудники) может использовать первичный ключ с именем EMP_NUM, таблица назначенных рейсов не может использовать два внешних ключа с именем EMP_NUM, чтобы связать первого и второго пилотов с таблицей EMPLOYEE (все имена атрибутов внутри таблицы должны быть уникальными). Поэтому таблица назначенных рейсов должна включить внешние ключи с именами PILOT (пилот) и CO_PILOT (второй пилот) для ссылки на один и тот же первичный ключ таблицы EMPLOYEE.

2.6. Связи в реляционной базе данных

Связи в базе данных можно классифицировать по типу "один-к-одному" (1:1), "один-ко-многим" (1:M) и "многие-ко-многим" (M:N или M:M). Как и предполагается, запись 1:1 означает, что данная сущность может быть связана только с одной другой сущностью и наоборот. Например, один декан управляет одним факультетом и на факультете может быть только один декан. Поэтому сущности DEAN (декан) и SCHOOL (факультет) представляют тип связи 1:1.

Существование связи типа 1:1 часто означает, что компоненты сущности определены не совсем корректно. Вдобавок это указывает, что две данные сущности фактически находятся в одной таблице! Однако иногда бывает, что связи 1:1 вполне уместны. Например, предположим, вы управляете базой данных компании, в которой работают пилоты, бухгалтеры, механики, клерки, продавцы, обслуживающий персонал и т. д. Пилоты имеют множество атрибутов, которых нет у других сотрудников, например, лицензию, медицинский сертификат, налетанные часы, дату прохождения контроля профессиональной пригодности и допуск к полетам от постоянной медицинской комиссии. Если вы поместите все специфичные только для пилотов атрибуты в таблицу EMPLOYEE, то в ней будет множество пустых мест (null) для сотрудников, не являющихся пилотами. Чтобы избежать этого, лучше всего выделить атрибуты пилота в отдельную таблицу PILOT и установить ее связь с таблицей EMPLOYEE как 1:1. (Поскольку у пилотов есть много атрибутов, имеющих у всех сотрудников, — имя, дата рождения и дата приема на работу — эти атрибуты разумнее хранить в таблице EMPLOYEE).

Связь 1:M практически идеальна для реляционной модели и является в ней основным кирпичиком. На самом деле связь 1:M настолько важна, что позже мы покажем, как можно конвертировать в нее другие типы связей, чтобы повысить эффективность базы данных.

Для выявления связей сущностей и их моделирования начнем с исследования связи между двумя сущностями — PAINTER (художник) и PAINTING (картина). Поскольку картина создается только одним художником, но каждый художник может написать несколько картин, мы имеем дело со связью 1:M. Эту связь мы можем изучать с помощью ER-модели (модели "сущность-связь"). ER-модель обеспечивает простое и наглядное представление о связях между сущностями.

Для отображения ER-модели мы будем использовать ER-диаграмму (ERD). Изображать ERD мы будем с помощью популярной формы представления на базе модели Чена, а также с помощью модели "птичья лапка"³. Мы будем придерживаться следующих соглашений при изображении ERD:

- как в модели Чена, так и в модели "птичья лапка" для представления сущностей используются прямоугольники;

³ Далее будет показано (см. гл. 3), что некоторые понятийные, концептуальные материалы лучше отображаются с помощью модели Чена, в то время как модель "птичья лапка" лучше подходит при описании реализаций. Поскольку данная книга предполагает активного читателя, мы будем считать модель "птичья лапка" предпочтительным форматом ER-диаграмм, после того как будут разрешены все концептуальные вопросы.

- ❑ как в модели Чена, так и в модели "птичья лапка" в качестве имен сущностей используются имена существительные, изображаемые заглавными буквами. Примеры: EMPLOYEE, INVOICE, DEPARTMENT. Обратите внимание, что имена сущностей приведены в единственном числе, форма множественного числа (EMPLOYEES, INVOICES и DEPARTMENTS) нежелательна;
- ❑ названия связей представляют собой глаголы в активной или пассивной форме. Примеры: "пишет", "управляется", "изготовлена", "работает";
- ❑ в модели Чена названия связей располагаются внутри ромбов. Ромб присоединяется к прямоугольнику, изображающему сущность, с помощью так называемой линии связи. В модели "птичья лапка" названия связей просто пишутся выше, ниже или рядом с линией связи, соединяющей прямоугольники, изображающие сущности;
- ❑ в модели Чена для обозначения части связи, означающей "один", используется цифра 1. В модели "птичья лапка" для этого используется черточка, пересекающая линию связи;
- ❑ в модели Чена для изображения части связи, означающей "многие", используется буква М. При изображении связи "многие-ко-многим" используются буквы М и N. В модели "птичья лапка" для части связи, представляющей "многие", используется стилизованное изображение трехпальцевой птичьей лапки (отсюда и название модели). Для связи "многие-ко-многим" такое изображение используется с обеих сторон линии связи.

На рис. 2.18 представлены ER-диаграммы, отражающие связь 1:М между сущностями PAINTER и PAINTING (художник пишет много картин, но каждая картина написана именно этим художником) и выполненные с учетом этих соглашений.

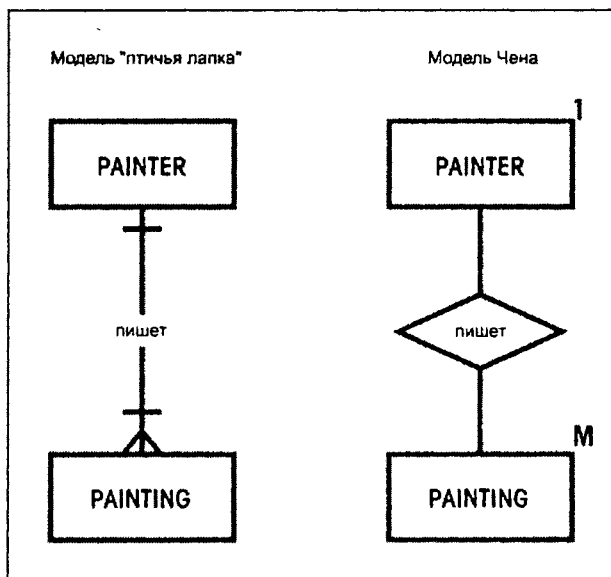


Рис. 2.18. Связи между PAINTER и PAINTING

Ориентация ER-диаграммы не играет существенной роли. Например, связь между PAINTER и PAINTING может быть представлена и так (рис. 2.19).

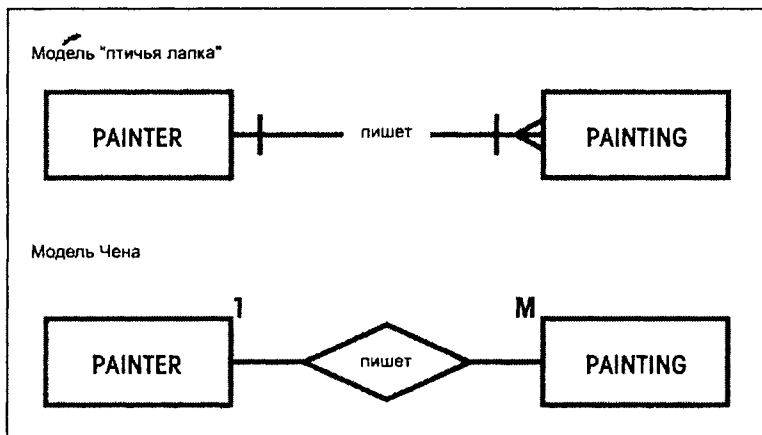


Рис. 2.19. Альтернативный способ представления связи между PAINTER и PAINTING

Примечание

Необходимо помнить, что реляционная таблица эквивалентна набору сущностей. Однако разработчики, а, следовательно, и пользователи ER-моделей применяют термин "сущность" вместо "набор сущностей". Для отображения ER-моделирования на практике мы также будем использовать такую договоренность. Поэтому, когда вы встречаете слово "сущность" в контексте обсуждения ER-моделирования, помните, что здесь имеется в виду набор сущностей. Поэтому в ER-моделировании лучше всего сущности представлять в виде таблиц.

Связь, представленная в предыдущей ER-модели, проиллюстрирована таблицами PAINTER и PAINTING, показанными на рис. 2.20. В этих таблицах следует обратить внимание на следующие особенности:

- каждая картина написана только одним художником, но каждый художник может написать несколько картин. Например, в таблице PAINTING указаны три картины художницы 123 (Georgette P. Ross);
- в таблице PAINTER может быть всего лишь одна строка для любой данной строки таблицы PAINTING, но в таблице PAINTING может быть несколько строк для данной строки таблицы PAINTER.

Точно так же студенты обычного колледжа или университета знают, что на каждом COURSE (курсе) может иметься несколько групп, но каждая группа относится только к данному курсу. Например, на курсе Accounting II (бухгалтерия II) могут иметься две группы: одна занимается по понедельникам, средам и пятницам с 10:00 до 10:50, а другая — по вторникам с 18:00 до 20:40.

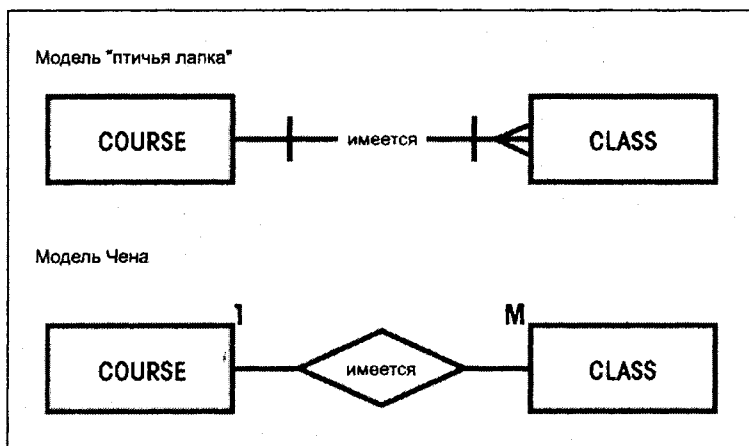


Рис. 2.21. Связь 1:M между COURSE и CLASS

База данных: CH2_COLLEGE					
Таблица: COURSE					
Первичный ключ: CRS_CODE					
Внешний ключ: не задан					
	CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT	
+	ACCT-211	ACCT	Accounting I	3	
+	ACCT-212	ACCT	Accounting II	3	
+	CIS-220	CIS	Intro. to Microcomputing	3	
+	CIS-420	CIS	Database Design and Implementation	4	
+	QM-261	CIS	Intro. to Statistics	3	
+	QM-362	CIS	Statistical Applications	4	

Таблица: CLASS						
Первичный ключ: CLASS_CODE						
Внешний ключ: CRS_CODE						
	CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
+	10012	ACCT-211	1	МВФ 8:00-8:50 a.m.	BUS311	105
+	10013	ACCT-211	2	МВФ 9:00-9:50 a.m.	BUS200	105
+	10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
+	10015	ACCT-212	1	МВФ 10:00-10:50 a.m.	BUS311	301
+	10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
+	10017	CIS-220	1	МВФ 9:00-9:50 a.m.	KLR209	228
+	10018	CIS-220	2	МВФ 9:00-9:50 a.m.	KLR211	114
+	10019	CIS-220	3	МВФ 10:00-10:50 a.m.	KLR209	228
+	10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
+	10021	QM-261	1	МВФ 8:00-8:50 a.m.	KLR200	114
+	10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
+	10023	QM-362	1	МВФ 11:00-11:50 a.m.	KLR200	162
+	10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

Рис. 2.22. Связь 1:M: база данных CH2_COLLEGE

Связь "многие-ко-многим" (M:N) труднее реализовать в реляционной среде. Однако мы покажем, что связь M:N можно разбить на несколько связей 1:M.

Для исследования связи "многие-ко-многим" (M:N) рассмотрим типичный колледж, в котором каждый студент (STUDENT) может входить в несколько групп (CLASS)⁴, а в каждой группе может быть несколько студентов (STUDENT). На рис. 2.23 представлена ER-диаграмма связи M:N.

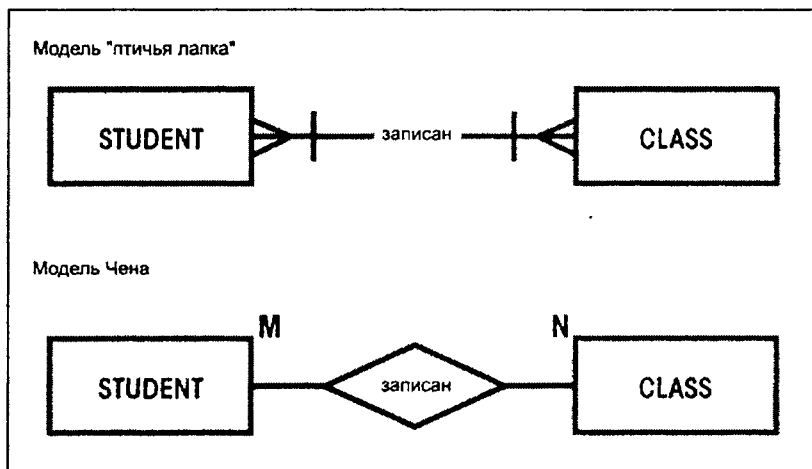


Рис. 2.23. Связь M:N между STUDENT и CLASS

Обратите внимание на особенности ER-диаграммы, представленной на рис. 2.23.

- ❑ В каждой группе (CLASS) может быть несколько студентов (STUDENT), а каждый студент (STUDENT) может посещать несколько групп (CLASS).
- ❑ В таблице CLASS может быть несколько строк для данной строки в таблице STUDENT и в таблице STUDENT может быть несколько строк для данной строки таблицы CLASS.

Чтобы рассмотреть связь M:N более детально, давайте представим себе небольшой колледж с двумя студентами, каждый из которых входит в три группы. В табл. 2.7 представлен регистрационный список двух студентов.

⁴ Система образования в США предусматривает, что *каждый* лекционный курс (COURSE), как правило, разбивается на учебные группы или классы (CLASS). Поскольку студент занимается на нескольких курсах, он может входить в несколько групп (говорят, что студент "берет" — take — несколько курсов). Если провести аналогию с нашей системой высшего образования, то можно сказать, что у нас данный предмет (например, химия) может изучаться несколькими группами, и в тоже время группа изучает несколько предметов. То есть между реляционными таблицами ГРУППА и ПРЕДМЕТ имеет место взаимосвязь M:N. — Прим. пер.

Таблица 2.7. Пример регистрационного списка студентов

Фамилия студента	В какие группы записался
Bowser	Accounting 1 (Бухгалтерский учет 1), ACCT-211, code (код) 10014 Intro, to Microcomputing (Введение в микрокомпьютеры), CIS-220, code 10018 Intro, to Statistics (Введение в статистику), QM-261, code 10021
Smithson	Accounting 1, ACCT-211, code 10014 Intro, to Microcomputing, CIS-220, code 10018 Intro, to Statistics, QM-261, code 10021

Хотя связь M:N можно логически представить в виде ER-диаграмм (см. рис. 2.23), ее трудно изобразить на реляционной схеме (рис. 2.24) по двум главным причинам..

- Таблицы содержат много избыточной информации. Обратите внимание, что в таблице STUDENT несколько раз встречается значение STU_NUM. В реальной ситуации дополнительные атрибуты студента, например, адрес, специализация, домашний телефон и т. д. также содержатся в таблице STUDENT, и каждый из этих атрибутов повторяется в каждой записи, представленной в этой таблице. Точно так же в таблице CLASS содержится множество дублирующихся данных: каждый студент, который выбирает занятия в данной группе, создает запись в таблице CLASS. Проблема станет еще острее, если в таблицу CLASS включить такие атрибуты, как кредитные часы⁵ и описание курса. Такая избыточность приводит к аномалиям, которые мы обсуждали в гл. 1.
- При такой структуре и наполнении этих двух таблиц реляционные операции становятся очень сложными и, как правило, приводят к снижению производительности и ошибкам при выводе информации.

К счастью, можно легко избежать проблем, причиной которых является связь M:N, создав *составную сущность (composite entity)* или *промежуточную сущность (bridge entity)*. Поскольку такая таблица используется для связывания таблиц, изначально связанных связью M:N, в структуру составной сущности включены (в качестве внешнего ключа), по крайней мере, первичные ключи связываемых таблиц. Проектировщик базы данных имеет две основные возможности для определения первичного ключа составной таблицы: использовать комбинацию этих внешних ключей или создать новый первичный ключ.

Необходимо помнить, что каждая сущность в ER-модели представляется таблицей. Поэтому для связывания таблиц CLASS и STUDENT мы можем создать таблицу (составную) ENROLL (регистрационный список), представленную на рис. 2.25. В этом примере первичный ключ таблицы ENROLL представляет собой комбинацию внешних ключей CLASS_CODE и STU_NUM. Но проектировщик может при-

⁵ Каждый курс дает студенту определенное количество "кредитов" или "кредитных часов". Это число примерно равняется количеству часов по данному предмету, которое студент проводит в классе в течение недели. Обычно за один курс студенту засчитывается от 3 до 5 кредитов. — Прим. пер.

нять решение создать первичный ключ, состоящий из одного атрибута, например, ENROLL_LINE, используя его различные значения для уникальной идентификации каждой строки таблицы ENROLL (пользователи Microsoft Access могут использовать тип данных Autonumber — автонумератор — для автоматического создания такого значения в каждой строке).

База данных: CH2_TEXT

Таблица: STUDENT_FIG2_24

	STU_NUM	STU_LNAME	CLASS_CODE
▶	321452	Bowser	10014
	321452	Bowser	10018
	321452	Bowser	10021
	324257	Smithson	10014
	324257	Smithson	10018
	324257	Smithson	10021

Таблица: CLASS_FIG2_24

	CLASS_CODE	STU_NUM	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
▶	10014	321452	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
	10014	324257	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
	10018	321452	CIS-220	2	MAF 9:00-9:50 a.m.	KLR211	114
	10018	324257	CIS-220	2	MAF 9:00-9:50 a.m.	KLR211	114
	10021	321452	QM-261	1	MAF 8:00-8:50 a.m.	KLR200	114
	10021	324257	QM-261	1	MAF 8:00-8:50 a.m.	KLR200	114

Рис. 2.24. Связь "многие-ко-многим" между STUDENT и CLASS

Поскольку таблица ENROLL на рис. 2.25 связывает две таблицы, она еще называется *связующей таблицей* (linking table). Другими словами, связующая таблица представляет собой реализацию составной сущности.

Примечание

В дополнение к связующим атрибутам составная таблица ENROLL может содержать такие значимые атрибуты, как GRADE (оценка за курс). Фактически составная таблица может содержать любое количество атрибутов, которые пожелает учесть проектировщик. Имейте в виду, что составная сущность, несмотря на то, что она представлена в виде реальной таблицы, является концептуально логической сущностью, созданной как средство для устранения потенциальной избыточности в исходной связи M:N.

Связующая таблица ENROLL, представленная на рис. 2.25, приводит связь M:N к виду 1:M. Заметим, что составная сущность, представленная таблицей ENROLL, должна содержать, по крайней мере, первичные ключи таблиц CLASS и STUDENT (соответственно CLASS_CODE и STU_NUM), для которых она служит связующим звеном. Особое внимание обратите на то, что таблицы STUDENT и CLASS теперь содержат только одну строку на каждую сущность. Связующая таблица ENROLL содержит несколько вхождений значения внешнего ключа, но такая контролируемая избыточность не вызовет аномалию данных при надлежащем обеспечении целостности на уровне ссылки.

Таблица: STUDENT_FIG2_25

Первичный ключ: STU_NUM

Внешний ключ: не задан

	STU_NUM	STU_LNAME
▶	321452	Bowser
	324257	Smithson

Таблица: ENROLL_FIG2_25

Первичный ключ: CLASS_CODE+STU_NUM

Внешний ключ: CLASS_CODE, STU_NUM

	CLASS_CODE	STU_NUM	ENROLL_GRADE
▶	10014	321452	C
	10014	324257	B
	10018	321452	A
	10018	324257	B
	10021	321452	C
	10021	324257	C

Таблица: CLASS_FIG2_25

Первичный ключ: CLASS_CODE

Внешний ключ: CRS_CODE

	CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
▶	10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
	10018	CIS-220	2	MMWF 9:00-9:50 a.m.	KLR211	114
	10021	QM-261	1	MMWF 8:00-8:50 a.m.	KLR200	114

Рис. 2.25. Конвертирование связи M:N в две связи 1:M

При необходимости можно ввести дополнительные атрибуты. В данном случае для создания отчетов выбран атрибут ENROLL_GRADE. Также обратите внимание, что первичный ключ таблицы ENROLL состоит из двух атрибутов CLASS_CODE и STU_NUM, поскольку для определения оценки студента по данному курсу необходимо знать как код группы (CLASS_CODE), так и идентификационный номер студента (STU_NUM). Естественно, это соглашение отражается и на ER-модели. Исправленная связь представлена на рис. 2.26.

На рис. 2.26 видно, что составная сущность с именем ENROLL представляет собой связующую таблицу между таблицами STUDENT и CLASS. В ER-диаграмме Чена для указания составной сущности использован ромб внутри прямоугольника.

Мы проиллюстрировали связь 1:M между таблицами COURSE и CLASS на рис. 2.21. С помощью этой связи мы можем увеличить количество доступной информации при контролируемой избыточности базы данных. Поэтому мы можем расширить рис. 2.26 так, чтобы включить связь 1:M между COURSE и CLASS (рис. 2.27). Обратите внимание, что эта модель позволяет обрабатывать несколько секций группы (CLASS), в то же время контролируя избыточность посредством того, что все данные по курсу (COURSE), общие для каждой группы (CLASS), хранятся в таблице COURSE.

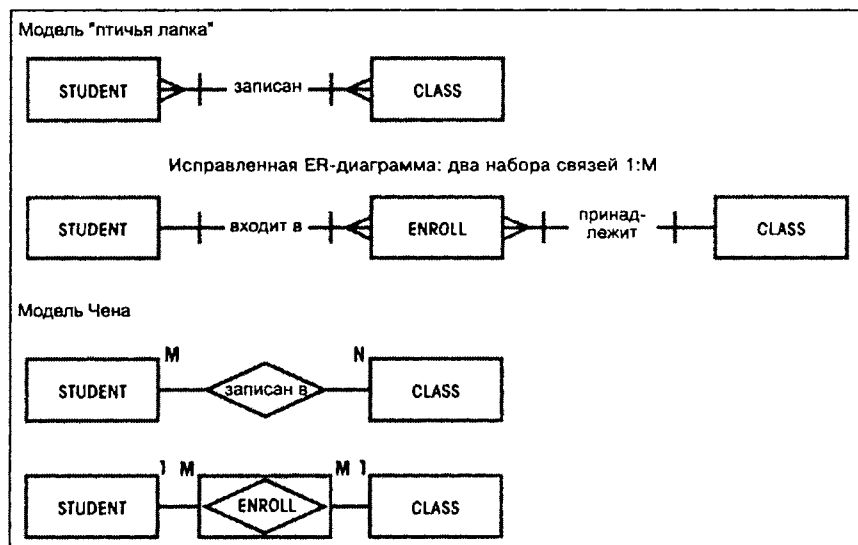


Рис. 2.26. Замена связи M:N на две связи 1:M

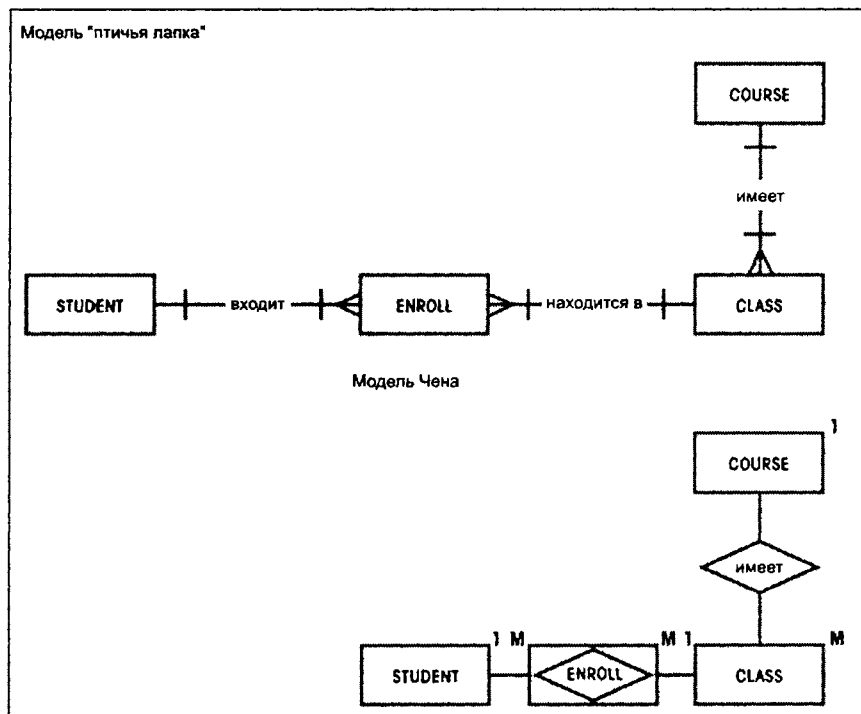


Рис. 2.27. Расширенная ER-модель

Реляционная схема, соответствующая ER-диаграмме на рис. 2.27, представлена на рис. 2.28. (При необходимости еще раз обратитесь к представлению реляционных схем в гл. 1, рис. 1.12.)

Мы исследуем эту ER-диаграмму более детально в гл. 3, чтобы показать, как можно использовать ее при проектировании более сложных баз данных. Эту же ER-диаграмму мы возьмем за основу при разработке и реализации реального проекта БД в гл. 7 и 8 (проектирование и реализация University Lab — лаборатории университета).

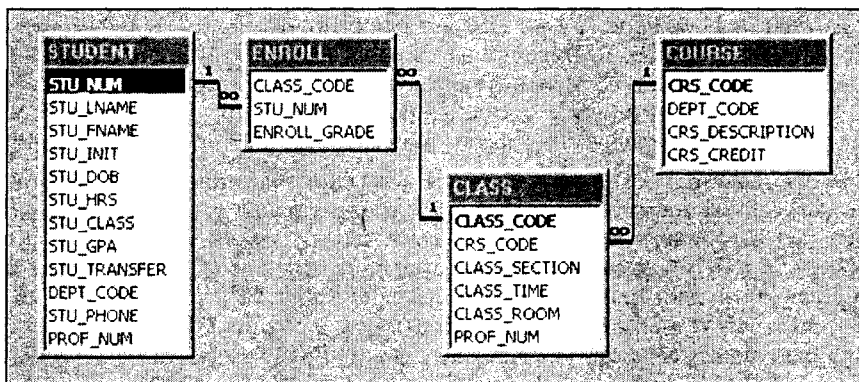


Рис. 2.28. Реляционная схема для ER-диаграммы с рис. 2.27

2.7. Еще раз об избыточности данных

В гл. 1 мы пришли к выводу, что избыточность данных является причиной аномалии данных. Такие аномалии могут снизить эффективность БД. Мы также выяснили, что реляционная база данных позволяет контролировать избыточность данных с помощью общих атрибутов, совместно используемых таблицами. Такие общие атрибуты называются внешними ключами.

Надлежащее использование внешних ключей играет решающую роль при контроле избыточности данных. Однако стоит подчеркнуть, что, строго говоря, использование внешних ключей не устраняет избыточность данных, поскольку внешний ключ может повторяться несколько раз. Тем не менее, правильное использование внешних ключей минимизирует избыточность данных, т. е. вероятность появления деструктивных аномалий в данных.

Примечание

Некоторые понимают избыточность несколько по-другому, утверждая, что реальная проверка избыточности состоит не в том, чтобы определять, сколько хранится копий данного атрибута, а в том, приведет ли удаление того или иного атрибута к потере нужной информации. Поэтому, если после удаления атрибута корректная информация все еще может быть создана с помощью реляционной алгебры, то включение такого атрибута можно считать избыточным. При таком взгляде на избыточность данных внешние ключи, оче-

видно, не будут влиять на избыточность, несмотря на их неоднократное появление в таблице. Однако даже при таком ограниченном подходе к избыточности данных необходимо иметь в виду, что контролируемая избыточность часто используется в системе для обеспечения скорости транзакций и/или удовлетворения требований клиента к представлению информации. Если для получения необходимой информации полагаться исключительно на реляционную алгебру, то при этом, конечно, можно добиться элегантного дизайна, который, тем не менее, на практике работать не будет.

В гл. 3 мы узнаем, что проектировщикам БД приходится учитывать три взаимоисключающих условия: элегантный дизайн, скорость обработки и требования клиента к представлению информации. А в гл. 13 будет показано, что при надлежащем проектировании информационного хранилища для его правильного функционирования просто необходима тщательно выверенная контролируемая избыточность данных. Говоря кратко, независимо от подхода к определению избыточности данных уровень ее потенциальной опасности зависит от способа реализации и тщательности контроля.

Контроль избыточности данных имеет очень важное значение, и может случиться так, что нам придется фактически увеличить уровень избыточности данных с целью заставить БД выполнять определенные критические задачи. Давайте рассмотрим небольшую систему учета счетов. В систему входят клиенты (CUSTOMER), которые могут приобретать один или несколько товаров (PRODUCT), тем самым инициируя выпуск счета (INVOICE). Поскольку клиент может купить более одного товара за один раз, счет может содержать несколько строк (LINE), в каждой из которых представлены сведения о купленном товаре. Таблица PRODUCT должна содержать цену товара, обеспечивая согласованное ценообразование по каждому продукту, который может появиться в счете. Таблицы этой системы представлены на рис. 2.29. Реляционная схема показана на рис. 2.30.

Если внимательно посмотреть на таблицы этой системы (рис. 2.29) и на связи, представленные на реляционной схеме (рис. 2.30), то можно проследить типовые продажи. Например, отследив связи между четырьмя таблицами, мы обнаружим, что клиент 10014 (Myron Orlando) приобрел две позиции 8 января 2002 года (January 8, 2002), что было отражено в выписанном счете за номером 1001: одну бытовую 16-дюймовую цепную пилу и два тонких напильника (*Подсказка: найдите две первых строки в таблице LINE, затем еще два совпадения PROD_CODE в таблице LINE со значениями PROD_CODE в таблице PRODUCT. Проследите связь INV_NUMBER таблицы LINE с таблицей INVOICE и строки 1001 таблицы INVOICE с таблицей CUSTOMER через CUS_CODE*). Для выписки правильного чека будет использоваться прикладная программа, в которой число приобретенных позиций в строке счета (LINE_UNITS) умножается на цену каждой единицы (LINE_PRICE), результаты суммируются, начисляется соответствующий налог и т. д. Затем можно использовать другие программы, которые могут создавать отчеты по продажам, отслеживать и сравнивать объем продаж за неделю, месяц или год.

В процессе исследования продаж разумно предположить, что цена товара, которая заносится в чек клиента, извлекается из таблицы PRODUCT, поскольку именно там хранится информация о товаре. *Но почему та же цена товара встречается снова в таблице LINE? Не является ли это избыточностью данных?* Да, конечно, является. Но здесь избыточность необходима для эффективной работы системы. Копирование цены продукта из таблицы PRODUCT в таблицу LINE позволяет обеспечить ретроспективную точность (historical accuracy) транзакции.

Таблица: CUSTOMER

База данных: CH2_SALE_CO

Первичный ключ: CUS_NUM

Внешний ключ: не задан

	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE
▶	1000	Ramas	Alfred	A	615	844-2573
	10011	Dunne	Leona	K	713	594-1236
	10012	Smith	Walter	W	615	894-2285
	10013	Olowski	Paul	P	615	894-2100
	10014	Orlando	Myron		615	722-1672
	10015	O'Brien	Amy	B	713	442-3001
	10016	Drown	James	G	615	297-1226
	10017	Williams	George		615	290-2556
	10018	Fieress	Anne	G	713	362-7185
	10019	Smith	Geoffe	K	615	297-3809

Таблица: INVOICE

Первичный ключ: INV_NUMBER

Внешний ключ: CUS_CODE

	INV_NUMBER	CUS_CODE	INV_DATE
▶	1001	10014	08-Jan-02
+	1002	10011	08-Jan-02
+	1003	10012	08-Jan-02
+	1004	10011	09-Jan-02

Таблица: LINE

Первичный ключ: INV_NUMBER+LINE_NUMBER

Внешний ключ: INV_NUMBER, PROD_CODE

	INV_NUMBER	LINE_NUMBER	PROD_CODE	LINE UNITS	LINE PRICE
▶	1001	1	123-210UY	1	\$189.99
	1001	2	SRE-657UG	3	\$2.99
	1002	1	GER-34256	2	\$18.63
	1003	1	ZZX-3245Q	1	\$6.79
	1003	2	SRE-657UG	1	\$2.99
	1003	3	001270-AU	1	\$12.95
	1004	1	001270-AU	1	\$12.95
	1004	2	SRE-657UG	2	\$2.99

Название таблицы: PRODUCT

Первичный ключ: PROD_CODE

Внешний ключ: не задан

	PROD_CODE	PROD_DESCRIPT	PROD_PRICE	PROD_ON HAND
▶	001270-AU	Claw hammer	\$12.95	23
+	123-210UY	Housette chain saw, 15-in. bar	\$189.99	4
+	GER-34256	Sledge hammer, 15-lb. head	\$18.63	6
+	SRE-657UG	Rail nail file	\$2.99	15
+	ZZX-3245Q	Steel tape, 12-ft. length	\$6.79	8

Рис. 2.29. Небольшая система учета счетов

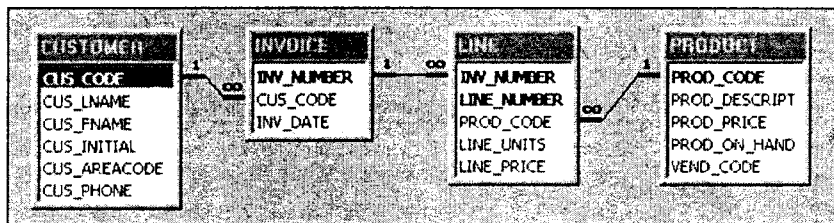


Рис. 2.30. Реляционная схема для системы учета счетов с рис. 2.29

Предположим, что мы не записываем `LINE_PRICE` в таблице `LINE`, а используем для этого атрибут `PROD_PRICE` (цена товара) из таблицы `PRODUCT` при расчете выручки. Теперь предположим, что атрибут `PROD_PRICE` таблицы `PRODUCT` изменился. Новые цены должным образом будут отражены во всех текущих расчетах. Но, к сожалению, при расчете доходов и от старых сделок теперь также будут учитываться новые цены, что не соответствует действительности! Поэтому расчет доходов по всем последующим транзакциям будет неправильным, и это делает невозможным сравнение объемов продаж во времени. С другой стороны, если цены на товар копируются из таблицы `PRODUCT` и хранятся вместе с параметрами транзакции в таблице `LINE`, то эта цена в точности соответствует именно той сделке, которая имела место в *то время*. В дальнейшем станет понятно, что такая запланированная избыточность — обычное явление в хорошо спроектированных БД.

Наконец, может вызвать удивление использование атрибута `LINE_NUMBER` в таблице `LINE` (см. рис. 2.29). Почему нельзя в качестве составного первичного ключа использовать комбинацию атрибутов `INV_NUMBER` и `PROD_CODE` и разве при этом атрибут `LINE_NUMBER` не является избыточным? Да конечно, атрибут `LINE_NUMBER` избыточен, но эта избыточность достаточно часто создается самим программным обеспечением выписки счетов, в котором номера строк счета генерируются автоматически. В данном случае эта избыточность не является необходимой. Но учитывая то, что номера строк создаются автоматически, эта избыточность не будет причиной аномалии данных. Подключение атрибута `LINE_NUMBER` дает еще одно преимущество: порядок извлечения данных по счетам будет всегда соответствовать тому порядку, в котором эти данные были введены. Если код товара использовать в качестве составляющей первичного ключа, индексация упорядочит эти коды, как только выписка счета завершена, а данные сохранены. Вы только представьте себе возможную путаницу, когда клиент звонит и сообщает вам, что "во второй позиции в моем счете неправильно указана цена", а вы находите счет, в котором порядок строк совершенно не такой, как у клиента!

2.8. Индексы

Предположим, что вам необходимо найти определенную книгу в библиотеке. Будете ли вы искать книгу, просматривая каждую книгу библиотеки, до тех пор, пока не будет найдена нужная книга? Конечно, нет: вы воспользуетесь библиотечным каталогом, который индексирован по заголовкам, темам и авторам. Индекс (как в компьютерной, так и в неавтоматизированной системе) "указывает" вам местоположение вашей книги, тем самым обеспечивая быстрый и удобный поиск книги.

Другой пример: вам необходимо найти в этой книге тему "ER-модели". Есть ли смысл листать все страницы в книге, пока вы не наткнетесь на искомую тему? Конечно же нет, проще обратиться к алфавитному указателю (индексу), найти фразу "ER-модель" и рядом с ней ссылку на нужную страницу. В каждом таком случае для быстрого поиска нужной позиции использовалось индексирование.

Индексы в реляционной базе действуют так же, как только что описанные индексы. С концептуальной точки зрения индекс состоит из индексного ключа и набора указателей. *Индексный ключ (index key)* в сущности представляет собой контрольную точку индекса. Предположим, нам необходимо в базе данных MUSEUM (музей) найти все картины данного художника (см. рис. 2.20). Без индексирования мы должны считывать каждую строку таблицы PAINTING и искать в ней соответствие атрибута PAINTING_NUM выбранному нами художнику. Однако если мы проиндексируем таблицу PAINTER по индексному ключу PAINTER_NUM, необходимо будет лишь найти подходящий атрибут PAINTER_NUM в индексе и найти соответствующего ему художника. Вообще говоря, такое индексирование должно выглядеть примерно так, как это представлено на рис. 2.31. Более формально — индексирование это упорядоченное расположение ключей и указателей. Каждый ключ указывает местоположение данных, идентифицированных ключом.

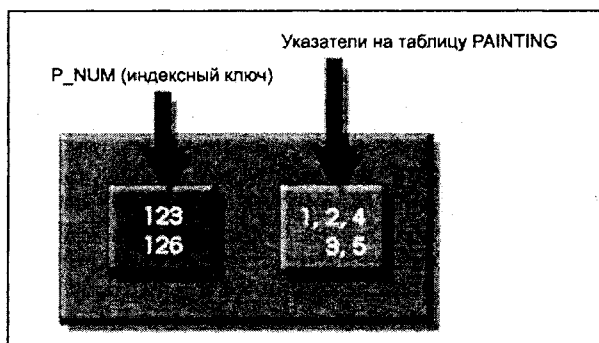


Рис. 2.31. Компоненты индекса

Примечание

Выполнить индексирование очень просто. В гл. 5 мы узнаем, что простая команда языка SQL позволяет создать необходимый индекс.

Резюме

Сущности (логические объекты) являются главными строительными блоками реляционных баз данных. Сгруппированные связанные сущности, называемые наборами сущностей, хранятся в таблице. Вообще говоря, реляционная таблица состоит из пересекающихся строк и столбцов. Каждая строка является отдельной сущностью (экземпляром сущности), а каждый столбец содержит свойство (атрибут) сущности.

Ключи — центральная составляющая реляционных таблиц. Ключи определяют функциональные зависимости. То есть другие атрибуты зависят от ключа и могут, следовательно, быть найдены по его значению. Ключи в соответствии с их свойствами подразделяются на суперключи (superkey), потенциальные ключи (candidate key), первичные ключи (primary key), вторичные ключи (secondary key) и внешние ключи (foreign key).

Каждая строка таблицы должна иметь первичный ключ. Первичный ключ представляет собой атрибут или комбинацию атрибутов, которые уникально идентифицируют все остальные атрибуты в любой строке. Поскольку первичный ключ должен быть уникален, в нем для обеспечения целостности не допускаются пустые значения (null).

Несмотря на то что все таблицы независимы, их можно связать при помощи совместно используемых атрибутов. Так, мы можем связать таблицы с помощью контролируемой избыточности; первичный ключ одной таблицы появляется вновь как внешний ключ в таблице, с которой устанавливается связь. Чтобы удостовериться, что внешний ключ производит корректное перемещение указателя, он должен поддерживать целостность на уровне ссылки, т. е. внешний ключ должен содержать или значения, соответствующие первичному ключу в другой таблице, или пустые значения (null).

Реляционные базы данных можно классифицировать в соответствии со степенью поддержки функций реляционной алгебры SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT и DIVIDE, а также по степени поддержки целостности (на уровнях ссылки и сущности). Полностью реляционные БД поддерживают все восемь функций реляционной алгебры и обеспечивают целостность на уровне как ссылки, так и сущности. В относительно полных БД отсутствуют только правила обеспечения целостности. Минимально реляционные БД поддерживают функции SELECT, PROJECT и JOIN и не обеспечивают соблюдение правил целостности. Табличная база данных поддерживает только функции реляционной алгебры, которые имеются в минимально реляционной БД, и требует определения путей доступа к таблицам. Чем выше степень реляционности БД, тем она более удобна для пользователя и тем проще проектировать приложения с ее помощью.

Если вы используете, по крайней мере, относительно полную БД, то ее РСУБД берет на себя выполнение большей части работы. Например, если вы создаете базу данных, РСУБД автоматически порождает для нее структуру размещения словаря данных и системного каталога. Каждый раз при создании в БД новой таблицы РСУБД обновляет словарь данных и системный каталог, таким образом документируя базу данных. При определении атрибутов таблиц базы данных полностью реляционная РСУБД проверяет (и предотвращает) некорректное использование омонимов и синонимов и обеспечивает выполнение правил целостности.

Ознакомившись с основами реляционных баз данных, можно приступить непосредственно к проектированию. Хороший проект начинается с определения сущностей и атрибутов, а также связей между сущностями. Такие связи (1:1, 1:M и M:N) могут быть представлены с помощью ER-моделей. Использование ER-моделей позволяет создать простой логический проект. Связи 1:M наиболее просто включаются в проект, необходимо лишь удостовериться, что первичный ключ таблицы, отражающий сторону связи "один" (1), включен в таблицу стороны "многие" (M). В лучших про-

ектах используются исключительно связи 1:M. К счастью, связи 1:1 и M:N могут быть конвертированы в связи 1:M.

Скорость выборки данных можно значительно увеличить с помощью индексов. Индексы проще всего создать с помощью SQL-команд.

Основные термины

Атрибут — attribute

Внешнее соединение — outer JOIN

Внешний ключ — foreign key

Вторичный ключ — secondary key

Диаграмма "сущность-связь" (ER-диаграмма) — entity relationship diagram (ERD)

Домен — domain

Домен атрибута — attribute domain

Естественное соединение — natural JOIN

Индекс, индексирование — index

Индексный ключ — index key

Ключ — key

Ключевой атрибут — key attribute

Кортеж — tuple

Левостороннее внешнее соединение — left outer JOIN

Модель "сущность-связь" (ER-модель) — entity relationship model (E-R model)

Набор сущностей — entity set

Омоним — homonym

Определяемость — determination

Первичный ключ — primary key

Полная функциональная зависимость — full functional dependence

Потенциальный ключ — candidate key

Правостороннее внешнее соединение — right outer JOIN

Пустое значение — null

Реляционная алгебра — relational algebra

Реляционная база данных — relational database

Реляционная схема — relational schema

Связующая таблица — linking table

Синоним — synonym

Система управления реляционной базой данных, реляционная система управления базой данных (РСУБД) — relational database management system (RDBMS)

Системный каталог — system catalog
Словарь данных — data dictionary
Совместимость по объединению — union-compatible
Составная (промежуточная) сущность — composite (bridge) entity
Составной ключ — composite key
Столбец соединения — join column
Суперключ — superkey
Сущность — entity
Таблица — table
Тета-соединение — theta JOIN
Флаг — flags
Функциональная зависимость — functional dependence
Целостность на уровне ссылки — referential integrity
Целостность на уровне сущности — entity integrity
Эквисоединение — equiJOIN

Вопросы

1. В чем различие между базой данных и таблицей?
2. Что имеет в виду специалист по базам данных, когда он говорит, что БД обеспечивает целостность на уровне как сущности, так и ссылки?
3. Почему целостности на уровнях ссылки и сущности очень важны для БД?
4. В руководстве пользователя по базам данных отмечается, что "файл содержит двести записей, каждая из которых содержит девять полей". Используйте соответствующую терминологию реляционных баз данных, чтобы "перевести" это предложение.
5. Используйте небольшую БД, представленную на рис. 2.32, для иллюстрации различия между естественным соединением (natural JOIN), эквисоединением (equiJOIN) и внешним соединением (outerJOIN).
6. Нарисуйте основную ER-диаграмму для базы данных, представленной на рис. 2.32.
7. Нарисуйте реляционную схему для базы данных, представленной на рис. 2.32.
8. Предположим, что у вас есть ER-модель, представленная на рис. 2.33. Как можно конвертировать эту модель в ER-модель, на которой будут присутствовать только связи 1:M? (Убедитесь, что вы рисуете исправленную реляционную модель.)
9. Что такое омонимы (homonyms) и синонимы (synonyms) и почему их следует избегать при проектировании БД?

База данных: CH2_QUESTIONS	
Таблица: STUDENT	
STU_CODE	PROF_CODE
100278	
128569	2
512272	4
531235	2
531268	
553427	1
Таблица: PROFESSOR	
PROF_CODE	DEPT_CODE
+	1 2
+	2 6
+	3 6
+	4 4

Рис. 2.32. База данных к вопросам 5–7

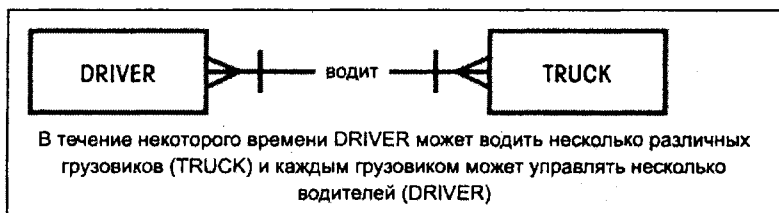


Рис. 2.33. ER-диаграмма к вопросу 8

10. Как можно реализовать связь 1:M в базе данных, состоящей из двух таблиц? Приведите пример.
11. Определите и опишите компоненты таблицы, представленной на рис. 2.34, используя корректную терминологию. Используйте ваше знание соглашения об именовании для определения подходящего внешнего ключа (или ключей) таблицы.
12. Предположим, что вы используете базу данных, состоящую из двух таблиц (рис. 2.35).
 - определите первичные ключи;
 - определите внешние ключи;
 - нарисуйте ER-модель;
 - нарисуйте реляционную схему, чтобы показать связь между таблицами DIRECTOR (режиссер) и PLAY (роль);

- предположим, что вам необходимо быстро получить список всех ролей у данного режиссера. Какую таблицу следует взять за основу при создании индексной таблицы (INDEX), и что будет представлять собой индексный ключ?
- что будет представлять собой таблица INDEX с концептуальной точки зрения. Поясните содержимое таблицы (концептуальной) INDEX.

Таблица: EMPLOYEE		База данных: CH2_QUESTIONS				
	EMP_NUM	EMP_LNAME	EMP_INITIAL	EMP_FNAME	DEPT_CODE	JOB_CODE
▶	11234	Friedman	K	Robert	MKTG	12
	11238	Olanski	D	Delbert	MKTG	12
	11241	Fontein		Juliette	INFS	5
	11242	Cruazona	J	Maria	ENG	9
	11245	Smithson	B	Bernard	INFS	6
	11248	Washington	G	Oleta	ENGR	8
	11256	McBride		Randall	ENGR	8
	11257	Kachinn	D	Melanie	MKTG	14
	11258	Smith	W	William	MKTG	14
	11260	Ratula	A	Katrina	INFS	5

Рис. 2.34. Таблица EMPLOYEE для вопроса 11

Таблица: DIRECTOR		База данных: CH2_QUESTIONS	
	DIR_NUM	DIR_LNAME	DIR_DOB
▶ +	100	Broadway	12-Jan-65
+	101	Hollywoody	18-Nov-53
+	102	Goofy	21-Jun-62

Таблица: PLAY		
PLAY_CODE	PLAY_NAME	DIR_NUM
▶ 1001	Cat On a Cold, Bare Roof	102
1002	Hold the Mayo, Pass the Bread	101
1003	I Never Promised You Coffee	102
1004	Silly Putty Goes To Washington	100
1005	See No Sound, Hear No Sight	101
1006	Starstruck in Biloxi	102
1007	Stranger In Parrot Ice	101

Рис. 2.35. База данных к вопросу 12

Задачи

Для решения задач 1—7 используйте БД, представленную на рис. 2.36.

Таблица: EMPLOYEE			База данных: CH2_P1	
	EMP_CODE	EMP_LNAME	JOB_CODE	
▶	+	14 Rudell	2	
	+	15 McDade	1	
	+	16 Ruellardo	1	
	+	17 Smith	3	
	+	20 Smith	2	

Таблица: BENEFIT		
	EMP_CODE	PLAN_CODE
▶	14	2
	15	3
	16	1
	17	1
	17	3
	17	4
	20	3

Таблица: JOB		
	JOB_CODE	JOB_DESCRIPTION
▶	+	1 Clerical
	+	2 Technical
	+	3 Managerial

Таблица: PLAN		
	PLAN_CODE	PLAN_DESCRIPTION
▶	+	1 Term life
	+	2 Stock purchase
	+	3 Long-term disability
	+	4 Dental

Рис. 2.36. База данных для задач 1—7

Обратите внимание, что эта база данных состоит из четырех таблиц и отражает следующие связи:

- у сотрудника (EMPLOYEE) есть только один код работы (JOB_CODE), но данный код работы (JOB_CODE) может относиться к нескольким сотрудникам (EMPLOYEE);

- ☐ сотрудник (EMPLOYEE) может иметь несколько льгот (BENEFIT), и каждая льгота (BENEFIT) может быть связана с несколькими сотрудниками (EMPLOYEE).

Также заметим, что связь M:N должна быть разбита на две связи 1:M, для которых таблица BENEFIT будет играть роль составной (промежуточной) сущности.

1. Для каждой таблицы базы данных определите первичный ключ и внешний ключ (или ключи) и оформите это в виде таблицы (табл. 2.8). Если таблица не имеет внешнего ключа, то в соответствующей позиции запишите "нет".

Таблица 2.8. Определение ключей таблиц базы данных (рис. 2.36)

Таблица	Первичный ключ	Внешний ключ
EMPLOYEE		
BENEFIT		
JOB		
PLAN		

2. Нарисуйте ER-диаграмму для связи между EMPLOYEE и JOB.
3. Нарисуйте реляционную схему для связи между EMPLOYEE и JOB.
4. Обеспечивается ли в этих таблицах целостность на уровне сущности? Ответьте "да" или "нет" и поясните свой ответ (используйте для этого форму табл. 2.9).

Таблица 2.9. Определение выполнения условий целостности базы данных на уровне сущности (рис. 2.36)

Таблица	Целостность на уровне сущности	Пояснения
EMPLOYEE		
BENEFIT		
JOB		
PLAN		

5. Обеспечивается ли в таблицах целостность на уровне ссылки? Ответьте "да" или "нет". Поставьте прочерк (табл. 2.10), если таблица не имеет внешнего ключа.

Таблица 2.10. Определение выполнения условий целостности базы данных на уровне ссылки (рис. 2.36)

Таблица	Целостность на уровне ссылки	Пояснения
EMPLOYEE		
BENEFIT		

Таблица 2.10 (окончание)

Таблица	Целостность на уровне ссылки	Пояснения
JOB		
PLAN		

- Нарисуйте ER-диаграмму для иллюстрации связей между таблицами EMPLOYEE, BENEFIT, JOB и PLAN.
- Нарисуйте реляционную схему, показывающую связи между таблицами EMPLOYEE, BENEFIT, JOB и PLAN.

Для решения задач 8—15 используйте базу данных, представленную на рис. 2.37.

Таблица: EMPLOYEE							
База данных: CH2_STORE_CO							
	EMP_CODE	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	STORE_CODE
+	1	Mr.	Williamson	John	W	21-May-64	3
+	2	Ms.	Ratula	Nancy		09-Feb-69	2
+	3	Ms.	Greenboro	Lottie	R	02-Oct-61	4
+	4	Mrs.	Rumperstro	Jennie	S	01-Jun-71	5
+	5	Mr.	Smith	Robert	L	23-Nov-59	3
+	6	Mr.	Renselaer	Cary	A	25-Dec-65	1
+	7	Mr.	Ogello	Roberto	S	31-Jul-62	3
+	8	Ms.	Johnsson	Elizabeth	I	10-Sep-68	1
+	9	Mr.	Eidsmar	Jack	W	19-Apr-55	2
+	10	Mrs.	Jones	Rose	R	06-Mar-66	4
+	11	Mr.	Broderick	Tom		21-Oct-72	3
+	12	Mr.	Washington	Alan	Y	08-Sep-74	2
+	13	Mr.	Smith	Peter	N	25-Aug-64	3
+	14	Ms.	Smith	Sherry	H	25-May-66	4
+	15	Mr.	Olenko	Howard	U	24-May-64	5
+	16	Mr.	Archialo	Barry	V	03-Sep-60	5
+	17	Ms.	Grimaldo	Jeanine	K	12-Nov-70	4
+	18	Mr.	Rosenberg	Andrew	D	24-Jan-71	4
+	19	Mr.	Rosten	Peter	F	03-Oct-68	4
+	20	Mr.	McKee	Robert	S	06-Mar-70	1
+	21	Ms.	Baumann	Jennifer	A	11-Dec-74	3

Таблица: STORE					
	STORE_CODE	STORE_NAME	STORE_YTD_SALES	REGION_CODE	EMP_CODE
+	1	Access Junction	\$1,003,455.76	2	8
+	2	Database Corner	\$1,421,987.39	2	12
+	3	Tuple Charge	\$986,783.22	1	7
+	4	Attribute Alley	\$944,568.56	2	3
+	5	Primary Key Point	\$2,930,098.45	1	15

Таблица: REGION	
	REGION_CODE
+	1 East
+	2 West

Рис. 2.37. База данных к задачам 8—15

8. Для каждой таблицы определите первичный ключ и внешний ключ (или ключи). Если таблица не имеет внешнего ключа, в соответствующей графе запишите "нет" (табл. 2.11).

Таблица 2.11. Определение ключей таблиц базы данных (рис. 2.37)

Таблица	Первичный ключ	Внешний ключ
EMPLOYEE (сотрудник)		
STORE (магазин)		
REGION (район)		

9. Обеспечивается ли в таблицах целостность на уровне сущности? Ответьте "да" или "нет" и поясните ответ (табл. 2.12).

Таблица 2.12. Определение выполнения условий целостности базы данных на уровне сущности (рис. 2.37)

Таблица	Целостность на уровне сущности	Пояснения
EMPLOYEE		
STORE		
REGION		

10. Обеспечивается ли в таблицах целостность на уровне ссылки? Ответьте "да" или "нет". Поставьте прочерк, если таблица не имеет внешнего ключа (табл. 2.13).

Таблица 2.13. Определение выполнения условий целостности базы данных на уровне ссылки (рис. 2.37)

Таблица	Целостность на уровне ссылки	Пояснения
EMPLOYEE		
STORE		
REGION		

- Опишите типы связей между таблицами STORE и REGION.
- Нарисуйте ER-диаграмму для связи между таблицами STORE и REGION.
- Нарисуйте реляционную схему для связи между таблицами STORE и REGION.
- Опишите типы связей между таблицами EMPLOYEE и STORE. (Подсказка: в каждом магазине работает несколько сотрудников, один из которых является директором.)

15. Нарисуйте ER-диаграмму, показывающую связи между таблицами EMPLOYEE, STORE и REGION.
16. Нарисуйте реляционную схему, иллюстрирующую связи между таблицами EMPLOYEE, STORE и REGION.

При решении задач 17—22 используйте базу данных, представленную на рис. 2.38.

Таблица: PRODUCT		База данных: CH2_P17				
	PROD_CODE	PROD_DESCRIPTION	PROD_STOCK_DATE	PROD_ON_HAND	PROD_PRICE	VEND_CODE
▶	12-VWVFP	7.25-in. power saw blade	07-Apr-02	12	\$1.19	123
	1QQ23-55	2.5-in. wood screw, 100	19-Mar-02	123	\$4.49	123
	231-78-VV	PVC pipe, 3.5-in., 8 ft.	07-Dec-01	45	\$8.87	121
	33564AJ	Rat-tail file, 0.125-in., fine	08-Mar-02	18	\$1.19	123
	AR3/TYR	Cordless drill, 0.25-in.	29-Nov-01	8	\$45.99	121
	DT-34-VWV	Phillips screwdriver pack	20-Dec-01	11	\$23.29	123
	EE3-67AV	Sledge hammer, 12 lb.	25-Feb-02	9	\$17.99	121
	ER-56/DF	Housette chain saw, 16-in.	28-Dec-01	7	\$235.49	125
	FRE-TRY9	Jigsaw, 12-in blade	12-Aug-01	67	\$1.45	125
	SE-67-89	Jigsaw, 8-in. blade	11-Oct-01	34	\$1.35	125
	ZW-QR/AV	Hardware cloth, 0.25-in.	23-Apr-02	14	\$12.99	123
	ZX-VVRFR	Claw hammer	01-Mar-02	15	\$8.95	121

Таблица: VENDOR					
	VEND_CODE	VEND_NAME	VEND_CONTACT	VEND_AREACODE	VEND_PHONE
▶	+	20 BargainSnapper, Inc.	Melanie T. Travis	615	899-1234
	+	121 Cut'nGlow Co.	Henry J. Otero	615	342-9896
	+	122 Rip & Rattle Supply Co.	Anne R. Morrins	901	225-1127
	+	123 Tools 'R Us	Juliette G. McHenry	615	546-7894
	+	124 Trowel & Dowel, Inc.	George F. Frederick	901	453-4567
	+	125 Bow & Wow Tools	Bill S. Sedwick	904	324-9988

Рис. 2.38. База данных для задач 17—22

17. Для каждой таблицы определите первичный ключ и внешний ключ (или ключи). Если таблица не имеет внешнего ключа, в соответствующей графе запишите "нет" (табл. 2.14).

Таблица 2.14. Определение ключей таблиц базы данных (рис. 2.38)

Таблица	Первичный ключ	Внешний ключ
PRODUCT (товар)		
VENDOR (поставщик)		

18. Обеспечивается ли в таблицах целостность на уровне сущности? Ответьте "да" или "нет" и поясните ответ (табл. 2.15).

Таблица 2.15. Определение выполнения условий целостности базы данных на уровне сущности (рис. 2.38)

Таблица	Целостность на уровне сущности	Пояснения
PRODUCT		
VENDOR		

19. Обеспечивается ли в таблицах целостность на уровне ссылки? Ответьте "да" или "нет". Поставьте прочерк, если таблица не имеет внешнего ключа (табл. 2.16).

Таблица 2.16. Определение выполнения условий целостности базы данных на уровне ссылки (рис. 2.38)

Таблица	Целостность на уровне ссылки	Пояснения
PRODUCT		
VENDOR		

20. Нарисуйте ER-диаграмму для этой базы данных.
 21. Нарисуйте реляционную схему этой базы данных.
 22. Создайте словарь данных этой БД.

Для решения задач 23—29 используйте базу данных, представленную на рис. 2.39.

23. Для каждой таблицы определите первичный ключ и внешний ключ (или ключи). Если таблица не имеет внешнего ключа, в соответствующей графе запишите "нет" (табл. 2.17).

Таблица 2.17. Определение ключей таблиц базы данных (рис. 2.39)

Таблица	Первичный ключ	Внешний ключ
TRUCK (трейлер)		
BASE (база)		
TYPE (тип трейлера)		

24. Обеспечивается ли в таблицах целостность на уровне сущности? Ответьте "да" или "нет" и поясните ответ (табл. 2.18).

Таблица 2.18. Определение выполнения условий целостности базы данных на уровне сущности (рис. 2.39)

Таблица	Целостность на уровне сущности	Пояснения
TRUCK		

Таблица 2.18 (окончание)

Таблица	Целостность на уровне сущности	Пояснения
BASE		
TYPE		

Таблица: TRUCK							База данных: CH2_TRUCK_CO	
	TRUCK_NUM	BASE_CODE	TYPE_CODE	TRUCK_MILES	TRUCK_BUY_DATE	TRUCK_SERIAL_NUM		
▶	1001	501	1	32123.5	23-Sep-01	AA-322-12212-VV11		
	1002	502	1	76984.3	05-Feb-98	AC-342-22134-Q23		
	1003	501	2	12346.6	11-Nov-01	AC-445-78656-Z99		
	1004		1	2894.3	06-Jan-02	VWQ-112-23144-T34		
	1005	503	2	45673.1	01-Mar-02	FR-998-32245-VV12		
	1006	501	2	193245.7	15-Jul-96	AD-456-00845-R45		
	1007	502	3	32012.3	17-Oct-99	AA-341-96573-Z84		
	1008	502	3	44213.6	07-Aug-01	DR-559-22189-D33		
	1009	503	2	10932.9	12-Feb-02	DE-887-98456-E94		

Таблица: BASE								
	BASE_CODE	BASE_CITY	BASE_STATE	BASE_AREA_CODE	BASE_PHONE	BASE_MANAGER		
▶	501	Murfreesboro	TN	615	123-4567	Andrea D. Gallager		
+	502	Lexington	KY	568	234-5678	George H. Delarosa		
+	503	Cape Girardeau	MO	456	345-6789	Maria J. Talindo		
+	504	Dalton	GA	901	456-7890	Peter F. McAvree		

Таблица: TYPE				
	TYPE_CODE	TYPE_DESCRIPTION		
▶	1	Single box, double-axle		
+	2	Single box, single-axle		
+	3	Tandem trailer, single-axle		

Рис. 2.39. База данных к задачам 23–29

25. Обеспечивается ли в таблицах целостность на уровне ссылки? Ответьте "да" или "нет". Поставьте прочерк, если таблица не имеет внешнего ключа (табл. 2.19).

Таблица 2.19. Определение выполнения условий целостности базы данных на уровне ссылки (рис. 2.39)

Таблица	Целостность на уровне ссылки	Пояснения
TRUCK		
BASE		
TYPE		

26. Определите потенциальный ключ (ключи) таблицы TRUCK.
27. Для каждой таблицы определите суперключ и вторичный ключ (табл. 2.20).

Таблица 2.20. Определение суперключа и вторичного ключа базы данных (рис. 2.39)

Таблица	Суперключ	Вторичный ключ
TRUCK		
BASE		
TYPE		

28. Нарисуйте ER-диаграмму этой базы данных.
29. Нарисуйте реляционную схему этой базы данных.

Для решения задач 30—33 используйте базу данных, представленную на рис. 2.40.

ROBCOR — некая чартерная авиакомпания, предоставляющая чартерные авиарейсы по требованию с помощью флота из четырех самолетов. Самолет идентифицируется уникальным регистрационным номером, поэтому регистрационный номер самолета подходит в качестве первичного ключа таблицы AIRCRAFT (авиалайнера). Пустые значения столбца CHAR_COPILOT (второй пилот) таблицы CHARTER (чартерный рейс) указывают на то, что на некоторых рейсах и на некоторых типах самолетов второй пилот не требуется. (Правила Federal Aviation Administration, FAA — Федерального Управления гражданской Авиации США — требуют наличия второго пилота на реактивных самолетах и на самолетах, имеющих общий взлетный вес более 12 500 фунтов (~5600 кг). Ни один из самолетов таблицы AIRCRAFT не подпадает под эти требования, однако некоторые клиенты могут потребовать наличия второго пилота по страховым причинам.) Все чартерные маршруты записаны в таблице CHARTER.

Примечание

Ранее в этой главе мы отмечали, что лучше по возможности избегать омонимов и синонимов. В этом примере как первый, так и второй пилот в таблице PILOT (пилот) называются одинаково — пилот (pilot), но мы не можем использовать EMP_NUM для них обоих в таблице CHARTER. Поэтому в таблице CHARTER мы используем синонимы CHAR_PILOT и CHAR_COPILOT.

Хотя такое "решение" в данном случае срабатывает, его следует применять в очень ограниченных случаях, поскольку оно создает пустые места, если второй пилот не требуется. Например, если понадобятся дополнительные члены экипажа (помимо первого и второго пилотов), мы будем вынуждены ввести дополнительные синонимы. Хуже того, использование максимального числа членов экипажа приводит к появлению дополнительных пустых мест в таблице CHARTER. У вас есть возможность исправить эту ошибку проектирования в задаче 33.

30. Для каждой таблицы, там, где это возможно, определите:
- первичный ключ;
 - суперключ;
 - потенциальный ключ;
 - внешний ключ;
 - вторичный ключ.

Таблица: CHARTER (первые 6 атрибутов)

База данных: CH2_AVIA_CO

CHAR_TRIP	CHAR_DATE	CHAR_PILOT	CHAR_COPILOT	AC_NUMBER	CHAR_DESTINATION
10001	05-Feb-02	104		2289L	ATL
10002	05-Feb-02	101		2778V	BNA
10003	05-Feb-02	105	109	4278Y	GNV
10004	06-Feb-02	106		1484P	STL
10005	06-Feb-02	101		2289L	ATL
10006	06-Feb-02	109		4278Y	STL
10007	06-Feb-02	104	105	2778V	GNV
10008	07-Feb-02	106		1484P	TYS
10009	07-Feb-02	105		2289L	GNV
10010	07-Feb-02	109		4278Y	ATL
10011	07-Feb-02	101	104	1484P	BNA
10012	08-Feb-02	101		2778V	MOB
10013	08-Feb-02	105		4278Y	TYS
10014	09-Feb-02	106		4278Y	ATL
10015	09-Feb-02	104	101	2289L	GNV
10016	09-Feb-02	109	105	2778V	MGY
10017	10-Feb-02	101		1484P	STL
10018	10-Feb-02	105	104	4278Y	TYS

Пункт назначения указывается стандартным трехбуквенным кодом аэропорта.

Например, STL - Сан-Луис, ATL - Атланта, GA, BNA - Nashville, TN и т. д.

Таблица: CHARTER (следующие 4 атрибута, первичный ключ повторен, чтобы связь с первой частью была более наглядной)

CHAR_TRIP	CHAR_DISTANCE	CHAR_HOURS_FLOWN	CHAR_HOURS_WAIT	CHAR_FUEL_GALLONS
10001	936	5.1	2.2	354.1
10002	320	1.6	0	72.6
10003	1574	7.8	0	339.8
10004	472	2.9	4.9	97.2
10005	1023	5.7	3.5	397.7
10006	472	2.6	5.2	117.1
10007	1574	7.9	0	348.4
10008	644	4.1	0	140.6
10009	1574	6.6	23.4	459.9
10010	998	6.2	3.2	279.7
10011	352	1.9	5.3	66.4
10012	884	4.8	4.2	215.1
10013	644	3.9	4.5	174.3
10014	936	6.1	2.1	302.6
10015	1645	6.7	0	459.5
10016	312	1.5	0	67.2
10017	508	3.1	0	105.5
10018	644	3.8	4.5	187.4

CHAR_HOURS = Количество налетанных часов

CHAR_FUEL_GALLONS = Израсходованное топливо в галлонах

CHAR_DISTANCE = Дистанция полета

Рис. 2.40. База данных к задачам 30–33

Таблица: CHARTER (оставшиеся атрибуты, первичный ключ повторен, чтобы связь с первыми двумя частями была более наглядной)

	CHAR_TRIP	CHAR_ON_QTS	CUS_CODE
▶	10001	1	10011
	10002	0	10016
	10003	2	10014
	10004	1	10019
	10005	2	10011
	10006	0	10017
	10007	2	10012
	10008	1	10014
	10009	0	10017
	10010	0	10016
	10011	1	10012
	10012	0	10010
	10013	1	10011
	10014	0	10017
	10015	2	10016
	10016	0	10011
	10017	0	10014
	10018	0	10017

Таблица: AIRCRAFT

	AC_NUMBER	MOD_CODE	AC_TTAF	AC_TTEL	AC_TTER
▶	1434P	PA23-250	1833.1	1833.1	101.8
+	2289L	C-90A	4243.8	768.9	1123.4
+	2778V	PA31-350	7992.9	1513.1	789.5
+	4278Y	PA31-350	2147.3	622.1	243.2

AC_TTAF = Общий налет фюзеляжа самолета (часы)

AC_TTEL = Общий налет левого двигателя (часы)

AC_TTER = Общий налет правого двигателя (часы)

В полностью разработанной базе данных такие значения атрибутов должны обновляться прикладным программным обеспечением, когда обновляется таблица CHARTER

Таблица: MODE

	MOD_CODE	MOD_MANUFACTURER	MOD_NAME	MOD_SEATS	MOD_CHG_MILE
▶	C-90A	Beechcraft	KingAir	8	\$2.67
+	PA23-250	Piper	Aztec	6	\$1.93
+	PA31-350	Piper	Navajo Chieftain	10	\$2.35

С клиентов взимается плата за полную дистанцию полета на основе значения атрибута MOD_CHG_MILE. Атрибут MOD_SEAT дает общее число мест в самолете, включая места первого и второго пилотов, поэтому маршрут PA31-350, пилотируемый двумя пилотами, может перевозить 8 пассажиров

Рис. 2.41. База данных к задачам 30—33 (продолжение)

Таблица: PILOT

	PIL_ID	PIL_CRS	PIL_RATINGS	PIL_MED_TYP	PIL_MED_DATE	PIL_PT135_DATE
+	101	ATP	SEL/MEL/Instr/CFII	1	20-Jan-02	11-Mar-02
+	104	ATP	SEL/MEL/Instr	1	18-Dec-01	17-Dec-01
+	105	COM	SEL/MEL/Instr/CFI	2	05-Mar-02	08-Nov-01
+	106	COM	SEL/MEL/Instr	2	10-Dec-01	17-Feb-02
+	109	COM	SEL/MEL/SES/Instr/CFII	1	22-Feb-02	15-Jan-02

Представленные здесь лицензии пилотов включают разрешения ATP (пилот транспортных авиалиний) и COM (пилот коммерческих авиалиний). Предприятия, работающие "по требованию", подчиняются разделу 135 правил FAA (эти предприятия также называются "Part 135 Operators").

Данные правила требуют, чтобы пилоты проходили аттестацию каждые шесть месяцев.

Дата прохождения аттестации записана в атрибуте PIL_PT135_DATE. Для коммерческих полетов пилоты должны иметь, по крайней мере, медицинский сертификат 2 класса (PIL_MED_TYPE = 2)

Рейтинг пилота (PIL_RATINGS) включает в себя:

SEL = одномоторные, над землей

MEL = многомоторные, над землей

SES = одномоторные, над морем

Instr = инженер

CFI = сертифицированный инструктор

CFII = сертифицированный инструктор, инженер

Таблица: EMPLOYEE

	EMP_ID	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB	EMP_HIRE_DATE
+	100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-88
+	101	Ms.	Lewis	Rhonda	G	19-Mar-85	25-Apr-86
+	102	Mr.	Vandam	Rhett		14-Nov-58	18-May-93
+	103	Ms.	Jones	Anne	M	11-May-74	26-Jul-99
+	104	Mr.	Lenge	John	P	12-Jul-71	20-Aug-90
+	105	Mr.	Williams	Robert	D	14-Mar-75	19-Jun-01
+	106	Mrs.	Duzak	Jeanine	K	12-Feb-88	13-Mar-89
+	107	Mr.	Diante	Jorge	D	01-May-75	02-Jul-97
+	108	Mr.	Wesenbach	Paul	R	14-Feb-66	03-Jun-93
+	109	Ms.	Travis	Elizabeth	K	18-Jun-61	14-Feb-02
+	110	Mrs.	Genkazi	Leighla	W	19-May-70	29-Jun-90

Таблица: CUSTOMER

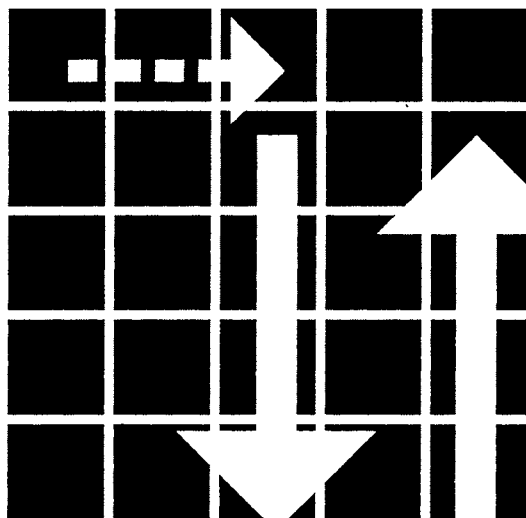
	CUS_CODE	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
+	10010	Ramas	Alfred	A	615	844-2573	\$0.00
+	10011	Dunne	Leona	K	713	894-1238	\$0.00
+	10012	Smith	Kathy	W	615	894-2285	\$896.54
+	10013	Olowski	Paul	F	615	894-2180	\$1,285.19
+	10014	Orlando	Myron		615	222-1672	\$673.21
+	10015	O'Brien	Amy	B	713	442-3381	\$1,014.56
+	10016	Brown	James	G	615	297-1228	\$0.00
+	10017	Williams	George		615	290-2556	\$0.00
+	10018	Farriss	Anne	G	713	382-7185	\$0.00
+	10019	Smith	Olette	K	615	297-3808	\$453.98

Рис. 2.42. База данных к задачам 30—33 (продолжение)

31. Создайте ER-диаграмму. (Подсказка: посмотрите на содержимое таблиц. Вы обнаружите, что AIRCRAFT может совершать множество чартерных рейсов (CHARTER), но каждый рейс (CHARTER) выполняется одним самолетом

(AIRCRAFT). Точно так же может быть несколько самолетов (AIRCRAFT) данной модели (MODEL), но данный самолет (AIRCRAFT) имеет только одну модель (MODEL) и т. д.)

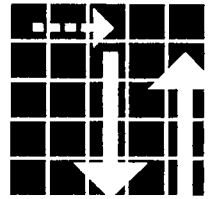
32. Создайте реляционную схему.
33. Измените ER-диаграмму, созданную в задаче 31, чтобы устранить проблемы использования синонимов. (*Подсказка:* измените структуру таблицы CHARTER, убрав атрибуты CHAR_PILOT и CHAR_COPILOT, а затем создайте составную таблицу с названием CREW (экипаж), чтобы связать таблицы CHARTER и EMPLOYEE. (Некоторые члены экипажа, например, инженеры, могут не быть пилотами, вот почему таблица EMPLOYEE входит в эту связь.))



ЧАСТЬ II

ОСНОВЫ ПРОЕКТИРОВАНИЯ И РЕАЛИЗАЦИИ

Глава 3



ER-моделирование

В этой главе мы обсудим следующее:

- ☐ что такое концептуальная модель и каково ее предназначение;
- ☐ различие между внутренней и внешней моделями;
- ☐ роль внутренней и внешней модели в процессе проектирования;
- ☐ поиск и выявление связей между сущностями и включение этих связей в процесс проектирования базы данных;
- ☐ влияние компонентов ER-диаграммы на проектирование и реализацию баз данных;
- ☐ толкование графических символов моделирования для четырех наиболее популярных инструментальных средств ER-моделирования;
- ☐ противоречивые требования при проектировании реальных баз данных.

Обзор

В этой главе мы исследуем роль моделирования данных в процессе проектирования БД. Фактически моделирование данных — первый шаг на пути проектирования базы данных, это переход от объектов реального мира к компьютерной модели базы данных.

Мы покажем, что одна из самых неприятных проблем проектирования базы данных состоит в том, что проектировщики, программисты и конечные пользователи имеют разное представление об информации. Например, различные точки зрения на одни и те же данные могут привести к тому, что БД не будет отражать реальное положение дел в компании и, следовательно, не будет отвечать потребностям конечных пользователей и требованиям эффективной обработки данных.

Чтобы избежать неправильного представления о характере функционирования предприятия, проектировщики баз данных должны получить точное описание данных и характера их использования внутри организации. Между проектировщиками БД, программистами и конечными пользователями должно существовать полное взаимопонимание. Моделирование данных играет ключевую роль в таком взаимодействии, позволяя распутывать сложные связи объектов реального мира, облегчая процесс обобщения при выявлении сущностей и связей между ними.

Мы детально исследуем наиболее известную модель данных, называемую ER-моделью (или моделью "сущность-связь"). Мы будем использовать ER-модель для

объединения различных представлений данных на концептуальном (понятийном) уровне. Во многих отношениях эта глава — самая важная в книге, поскольку ER-моделирование — основа проектирования БД. Чтобы помочь вам глубже понять процесс моделирования, в наших рассуждениях и иллюстрациях мы будем опираться на две модели: модель Чена (Chen) и модель "птичья лапка" (Crow's Foot). Модель Чена особенно полезна для иллюстрации некоторых концептуальных сторон проектирования БД. И мы будем использовать эту модель в основном при первоначальном обсуждении основных понятий моделирования БД. Модель "птичья лапка" более ориентирована на реализацию, чем модель Чена, и в связи с этим мы будем ее чаще использовать при изучении собственно процесса концептуального моделирования. Поскольку во главу угла книги поставлен практический подход к проектированию БД, как правило, в последующих главах будет применяться именно модель "птичья лапка". (Наше решение использовать эту модель в качестве стандартной вызвано ее частым применением в инструментальных средствах проектирования БД). Мы также вкратце обсудим еще две модели: Rein85 и IDEF1X.

3.1. Основы моделирования

Традиционно проектировщики баз данных полагались на свою профессиональную интуицию, считая, что она-то и поможет им создать хороший проект. К сожалению, зачастую интуиция свойственна стороннему наблюдателю ("каждый мнит себя страгелом, видя бой со стороны"). Интуиция, как правило, приходит лишь после того, когда набито много шишек и совершено много ошибок (особенно ошибок!). А ведь в настоящее время даже концепция хорошего проектирования еще не разработана! По крайней мере, сегодня проектирование БД можно считать в равной степени как искусством, так и наукой. К счастью, сегодня искусство проектирования баз данных подкрепляется мощными инструментальными средствами проектирования.

Словарь Вебстера (Webster's Dictionary) определяет модель так: "модель — это описание или аналогия, используемая для визуализации чего-либо, что не может наблюдаться непосредственно"¹. Другими словами, модель это некая абстракция сложных реальных объектов. Основное предназначение модели — облегчить понимание структур данных и их свойств, связей и ограничений.

Проектировщик БД использует модели данных как средство коммуникации, обеспечивающее взаимодействие проектировщиков, прикладных программистов и конечных пользователей. Если модель данных тщательно проработана, то она поможет лучше понять положение дел в той организации, для которой разрабатывается проект базы данных. Эта важная сторона моделирования данных была четко сформулирована одним заказчиком:

Я создавал это предприятие, я работал на этом предприятии в течение многих лет и только сейчас, наконец, понял, как все это работает на самом деле.

Хороший проект базы данных — основа хороших приложений. Независимо от уровня мастерства программистов и/или эффективности генератора прикладных про-

¹ Словарь Вебстера компьютерных терминов (Dictionary of Computer Terms, Eight Edition) определяет модель, как "имитацию ситемы, существующей в реальном мире". — *Прим. пер.*

грамм мы не сможем разработать хорошие приложения без хорошо спроектированной базы данных. А хороший проект базы данных начинается с построения полноценной модели данных.

Важность моделирования данных трудно переоценить. Данные составляют основу всех информационных устройств, используемых в системе. Приложения создаются для управления данными и структурирования информации. Сравните, например, представление о компании руководителя и простого служащего. Несмотря на то что и руководитель, и служащий работают на одном и том же предприятии, менеджер, по всей вероятности, имеет более широкое представление о деятельности предприятия, чем служащий.

Даже два руководителя по-разному представляют данные предприятия. Например, президент компании, скорее всего, имеет некое универсальное представление о данных, поскольку он должен увязывать между собой все подразделения компании в единое общее представление (базу данных). Заведующий отделом закупок этой же компании наверняка имеет несколько иное представление о данных организации, чем руководитель службы материально-технического снабжения. В сущности, руководитель каждого подразделения работает с неким подмножеством данных всей компании. Сфера интересов руководителя службы материально-технического снабжения связана с учетом имеющегося оборудования, в то время как руководитель отдела закупок имеет дело со стоимостью товаров и взаимоотношениями с поставщиками этих товаров.

Прикладные программисты также имеют собственное представление о данных предприятия, связанное с размещением данных, их форматом и специфическими требованиями к отчетности. По существу прикладные программисты переводят политику компании и различные методики из различных источников в соответствующие интерфейсы, отчеты и экранные запросы.

Различные пользователи и владельцы информации зачастую напоминают известную притчу о "слоне и слепцах": слепцы, которые ощупывали хобот слона, представляли его совершенно другим, чем те, которые ощупывали его ноги или хвост. Все, что нам нужно, — это возможность увидеть слона в целом. Аналогично, дом — это произвольная совокупность комнат; если уж мы собираемся строить дом, то должны, прежде всего, иметь о нем некое общее представление — проект. Точно так же нормальная конфигурация данных требует общего проекта базы данных, основанного на подходящей модели данных.

Если у нас есть хороший проект, то не имеет значения, что представление о данных некоторых прикладных программистов будет отличаться от представления руководителя и/или конечного пользователя. И наоборот, если у нас проекта нет, проблемы, скорее всего, возникнут. Например, программа учета материальных средств или система учета счетов может не вписаться в общие требования к системе, что может потребовать от компании дополнительных затрат в тысячи, а то и миллионы долларов.

Следует помнить, что проект дома есть некая абстракция: вы же не можете жить внутри проекта. Точно так же модель данных представляет собой абстракцию: вы не можете вытащить нужные вам данные из модели данных. И как не построить хороший дом без проекта, так вряд ли можно создать хорошую базу данных без выбора подходящей модели данных.

3.2. Модели данных: уровни абстракции данных

Национальный институт стандартизации США (ANSI) и комитет по планированию выпуска стандартов и технических условий (SPARC) определяют три различных модели данных в соответствии с уровнями абстракции: концептуальную (conceptual), внешнюю (external) и внутреннюю (internal) (рис. 3.1). Если внимательно посмотреть на рис. 3.1, то можно прийти к выводу, что классификацию моделей данных ANSI/SPARC необходимо расширить, добавив физическую (physical) модель данных, чтобы в этой классификации нашли отражение проблемы реализации, специфичные для СУБД во внутренней модели. К тому же, чтобы отразить представление о данных со стороны проектировщика, мы должны явно выделить связь между внешней моделью данных и внутренней моделью.

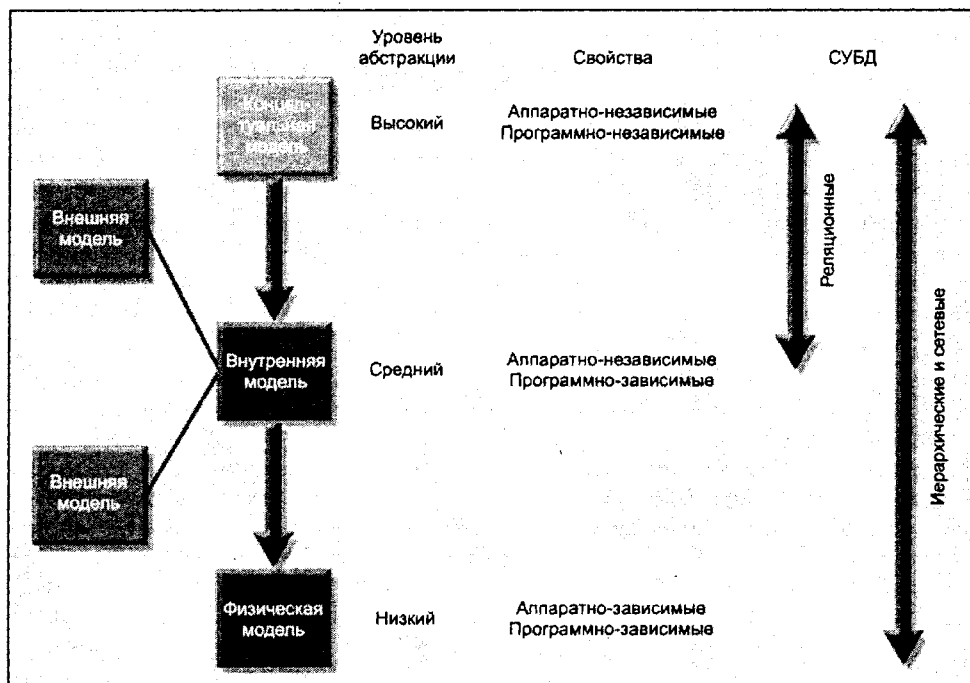


Рис. 3.1. Четыре модифицированные (ANSI/SPARC) абстрактные модели данных

Примечание

Успешный проект базы данных, прежде всего, базируется на требованиях конечного пользователя. Несмотря на то что основная работа проектировщика состоит во всестороннем взаимодействии с конечным пользователем, именно проектировщик принимает техническое решение, влияющее на структуру проекта. Поскольку мы уделяем основное внимание тех-

ническим аспектам проектирования БД, мы будем придерживаться представления о данных со стороны проектировщика. Иначе говоря, если вы хотите спроектировать хорошую машину, то вы должны достаточно много времени уделить тем людям, которые собираются ее купить и ездить на ней. И в то же время вы не должны заставлять водителя определять порядок зажигания в камерах сгорания или метод отливки головки блока.

Перед тем как приступить к детальному изучению моделей данных, представленных на рис. 3.1, обратите внимание, что они могут быть реализованы с помощью иерархических, сетевых или реляционных СУБД. С точки зрения проектировщика реляционная СУБД ориентирована скорее на концептуальную модель, сетевая же и иерархическая СУБД больше соответствуют требованиям физической модели.

3.2.1. Концептуальная модель

Концептуальная модель (conceptual model) представляет общий взгляд на данные. Это представление о данных всего предприятия с точки зрения менеджеров высшего уровня. Поэтому эта модель расположена на вершине абстракции (см. рис. 3.1).

Концептуальная модель — основа для идентификации и описания основных объектов данных за исключением подробностей. Наиболее широко используемая концептуальная модель — модель "сущность-связь" (ER-модель), представленная в гл. 2. В гл. 1 и 2 было показано, что ER-модели иллюстрируются с помощью диаграмм "сущность-связь" (ER-диаграмм, ERD), которые на самом деле являются основой проектирования. ER-диаграммы используются для графического представления концептуальной модели базы данных.

Чтобы проиллюстрировать использование концептуальной модели, исследуем информационную среду колледжа Tiny College. Основными объектами колледжа являются студенты, преподаватели, курсы, группы и аудитории. Эти объекты и есть основные сущности, для которых требуется собирать и хранить данные.

Примечание

В гл. 2 говорилось, что набор сущностей это совокупность связанных сущностей. Например, всех студентов колледжа Tiny College можно считать набором сущностей с именем STUDENT (студент). Однако для обозначения набора сущностей словарь моделирования данных использует термин "сущность". То есть разработчик модели данных полагает, что все студенты колледжа представляют собой сущность с именем STUDENT. Если забыть об этой особенности терминологии, то может возникнуть путаница при описании связей между сущностями².

Сущности колледжа Tiny College с названиями STUDENT (студент), PROFESSOR (преподаватель), COURSE (курс), CLASS (группа) и ROOM (аудитория), а также некоторые примеры атрибутов сущности STUDENT представлены на рис 3.2.

Используя сущности, представленные на рис. 3.2, мы можем описать *связи* (которые еще называют *ассоциациями* — associations или *интеракциями* — interactions) между

² В литературе по базам данных также применяется следующая терминология: *сущность* определяется как некоторый объект, представляющий интерес (то, что в данной книге называется *набором сущностей*), а этот объект состоит из *экземпляров сущности*, отличающихся друг от друга и допускающих однозначную идентификацию — *Прим ред.*

этими сущностями. В гл. 2 было показано, что существуют следующие типы связей: "один-к-одному" (1:1), "один-ко-многим" (1:M) и "многие-ко-многим" (M:N).

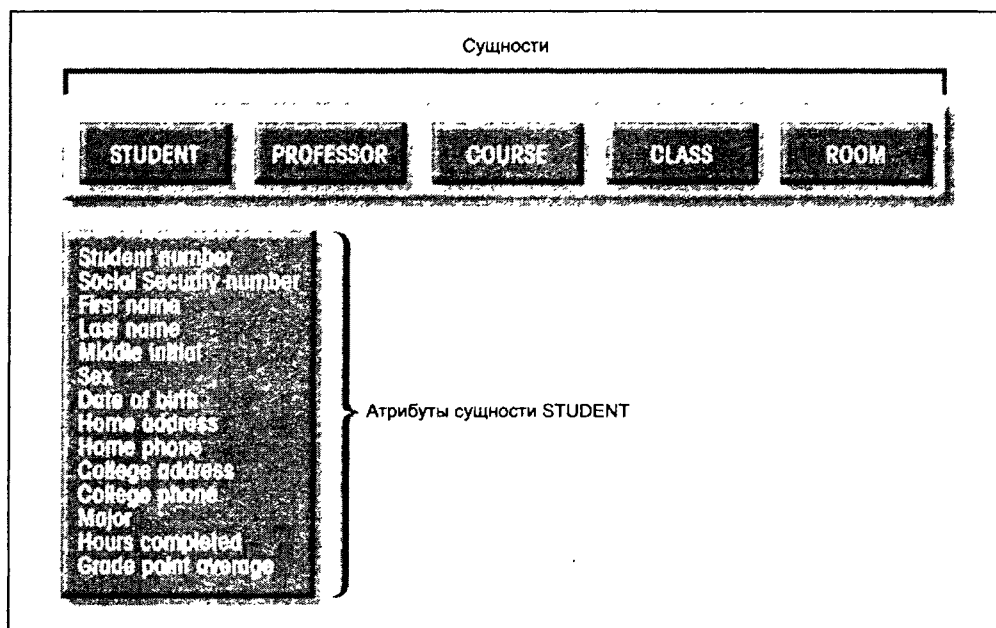


Рис. 3.2. Сущности колледжа Tiny College

Идентифицировав сущности, используем концептуальную модель, графически представленную ER-диаграммой (рис. 3.3), для связывания одной сущности с другой. На ER-диаграмме (см. рис 3.3) можно заметить, что связи описываются глаголами "обучает", "имеются", "создает" и "используется для". То есть преподаватель (PROFESSOR) обучает группу (CLASS), в группе (CLASS) имеются студенты (STUDENT), а аудитория (ROOM) используется для группы (CLASS). Обратите внимание, что связи сущностей на рис. 3.3 подразделяются на типы 1:M или M:N. Например, преподаватель (PROFESSOR) может обучать несколько групп (CLASS), но каждая группа (CLASS) обучается только у одного преподавателя (PROFESSOR), т. е. между преподавателем (PROFESSOR) и группой (CLASS) существует связь 1:M. В группу (CLASS) может войти несколько студентов (STUDENT), и каждый студент (STUDENT) может обучаться в нескольких группах (CLASS), т. е. между ними есть связь M:N. Мы обсуждали в гл. 2, что на каждом курсе (COURSE) может быть несколько групп (CLASS), но каждая группа (CLASS) связана только с одним курсом (COURSE). Например, может существовать несколько групп (секций) на курсе по базам данных (шифр курса CIS-420). Одна из этих групп может заниматься по понедельникам, средам и пятницам с 8:00 до 8:50, вторая по понедельникам, средам и пятницам с 13:00 до 13:50, третья — по четвергам с 18:00 до 20:40, и все эти три группы имеют шифр курса CIS-420. Наконец, группе (CLASS) необходима одна аудитория (ROOM), но в данной аудитории (ROOM) в соответствии с расписанием

могут заниматься разные группы (CLASS), т. е. одна группа может заниматься в данной аудитории с 9:00, другая с 11:00 и т. д. Другими словами, между аудиторией (ROOM) и группой (CLASS) есть связь 1:M.

У концептуальной модели имеется ряд важных достоинств. Во-первых, она обеспечивает относительно простой и понятный взгляд на информационную среду (на макро-уровне). Например, концептуальная модель, представленная на рис. 3.3, позволяет получить общее представление о конфигурации данных колледжа Tiny College.

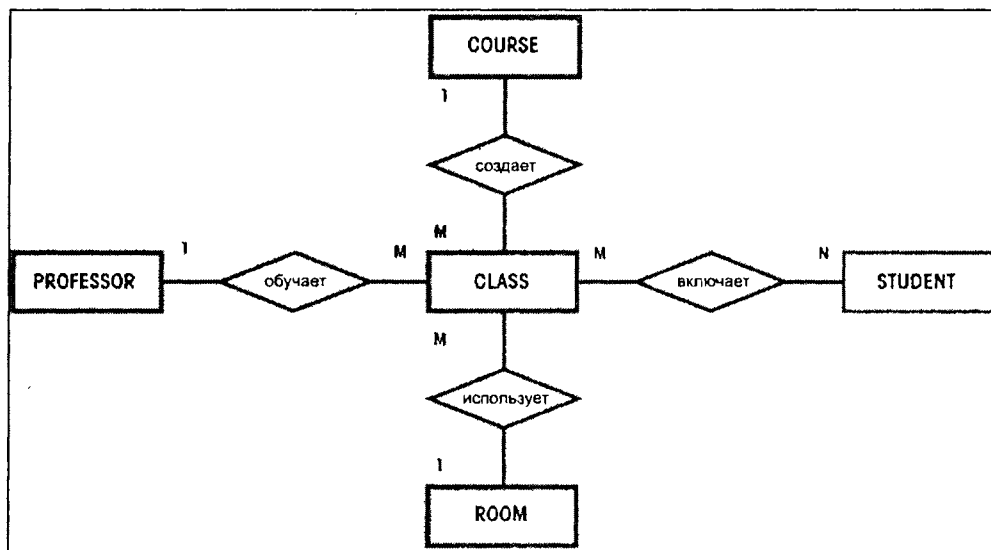


Рис. 3.3. Концептуальная модель колледжа Tiny College

Для СУБД имеет значение только название связи — ей нет дела до того, как описывается связь или как она читается. Однако человек предпочитает, чтобы связь была описана на литературном языке. По этой причине описание связи 1:M "читается" от стороны "1" к стороне "M", несмотря на то, как расположены прямоугольники, обозначающие сущности. Следовательно, мы читаем связь между группой (CLASS) и аудиторией (ROOM) на рис. 3.3 как "в аудитории (ROOM) занимается несколько групп (CLASS)", а связь между PROFESSOR и CLASS — как "преподаватель (PROFESSOR) обучает несколько групп (CLASS)".

Во-вторых, концептуальная модель не зависит ни от программного обеспечения, ни от аппаратных средств. Независимость от программного обеспечения означает, что реализация данной модели не связана с программным обеспечением СУБД. Независимость от аппаратного обеспечения означает, что реализация данной модели не зависит от оборудования. Следовательно, изменения как в аппаратных средствах, так и в программном обеспечении СУБД не оказывают воздействия на концептуальную модель базы данных.

3.2.2. Внутренняя модель

После выбора определенной СУБД концептуальная модель адаптируется к ней с помощью внутренней модели. *Внутренняя модель (internal model)* есть представление базы данных "с точки зрения" СУБД. Иначе говоря, внутренняя модель требует, чтобы проектировщик привел свойства и ограничения концептуальной модели в соответствие с выбранной моделью реализации базы данных.

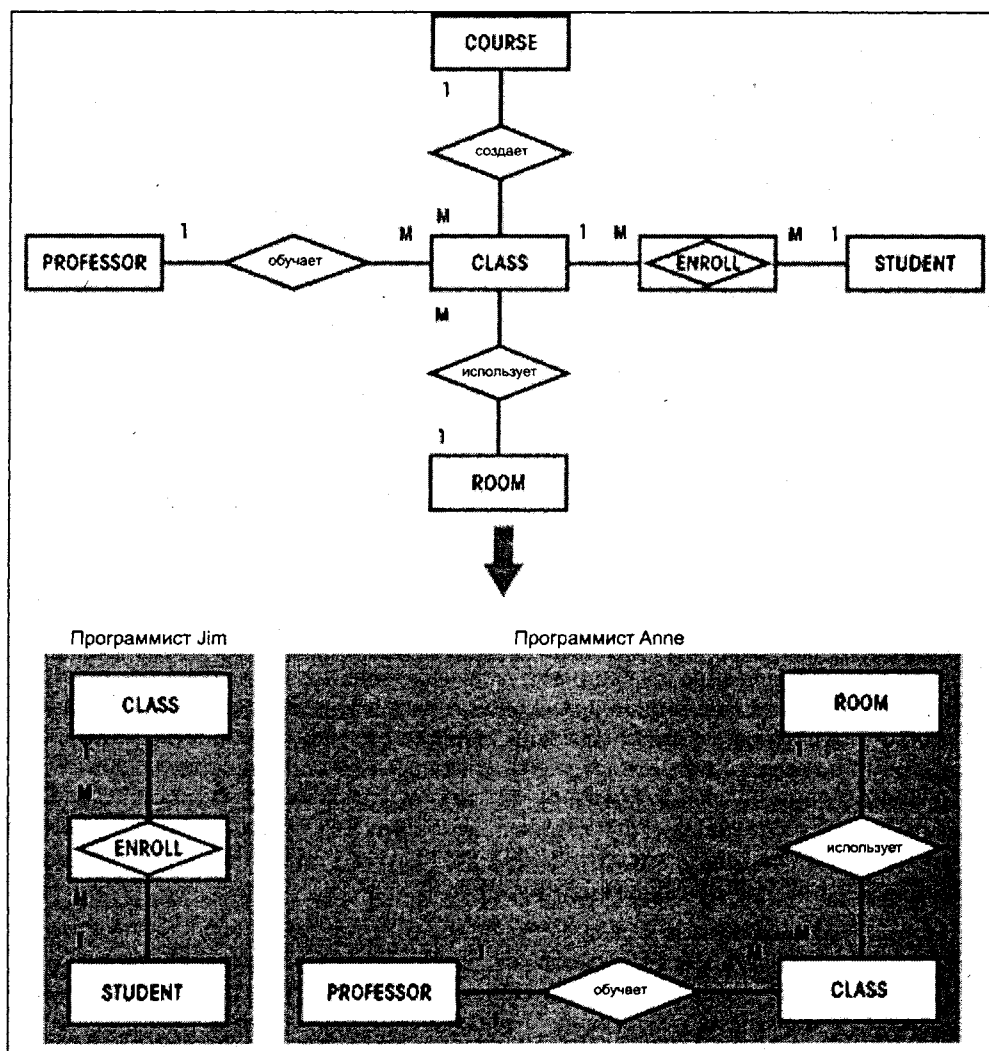


Рис. 3.4. Деление внутренней модели на внешние модели

Поскольку внутренняя модель зависит от существующего специфичного программного обеспечения БД, говорят, что она программно-зависима. Поэтому изменения в программном обеспечении СУБД потребуют изменений во внутренней модели для того, чтобы она отвечала требованиям СУБД.

Тщательная разработка внутренней модели особенно важна для проектировщиков баз данных, имеющих дело с иерархическими или сетевыми моделями баз данных, поскольку в этих моделях требуется очень точное описание местоположения хранимых данных и путей доступа к ним. В отличие от этого, реляционная модель базы данных требует меньшей детализации внутренней модели, поскольку в большинстве РСУБД способ доступа к данным скрыт от пользователя, т. е. проектировщику не нужно беспокоиться о точном описании пути доступа к данным. Тем не менее, в программном обеспечении реляционных баз данных тоже, как правило, требуется определение пути доступа к хранилищу данных, особенно на универсальных машинах (мэйнфреймах, mainframe). Например, в DB2 требуется, чтобы были определены группа хранения данных, местоположение базы данных в группе хранения и местоположение таблиц внутри базы данных.

В случае, представленном на рис. 3.3, мы реализуем внутреннюю модель, создав базу данных для Tiny College с помощью таблиц PROFESSOR, COURSE, CLASS, STUDENT и ROOM. Мы также создадим промежуточную сущность между CLASS и STUDENT с названием ENROLL, как это показано на рис. 3.4. Внутренняя модель все еще не зависит от аппаратного обеспечения, поскольку на нее никак не влияет выбор компьютера, на котором установлено данное программное обеспечение. Следовательно, изменение устройства хранения или даже изменение операционной системы не повлияет на требования к разработке внутренней модели.

3.2.3. Внешняя модель

Внешняя модель (external model) основана на внутренней модели и отражает представление конечного пользователя о конфигурации данных. Под конечными пользователями мы понимаем людей, использующих прикладные программы, а также тех, кто их разрабатывает и внедряет. Конечные пользователи обычно оперируют приложениями в определенных подразделениях. Предприятие обычно делится на несколько самостоятельных организационных единиц (подразделений), таких как Продажи, Финансы, Маркетинг и т. д. В каждом подразделении есть собственные специфика, система ограничений и требований, и в каждом из них используется некое подмножество всей информационной среды предприятия. Следовательно, прикладные программисты, работающие в таком подразделении, рассматривают свое подмножество данных отдельно от той внутренней модели (или как внешнее по отношению к ней), из которой они были получены.

Поэтому с точки зрения прикладного программиста желательно, чтобы специалисты по моделированию все множество требований и ограничений разбивали на функциональные модули, которые можно исследовать в рамках структуры внешних моделей. Например, внутренняя модель на рис. 3.4 разбита на два функциональных модуля, представляющих собой две внешние модели. Каждая из внешних моделей является полем деятельности прикладного программиста. В каждую внешнюю модель включены все необходимые сущности, связи, процессы и ограничения, определяемые данным подразделением. Обратите внимание, что каждая внешняя модель представляет

собой подмножество внутренних моделей, показанных на рис. 3.4. Также важно отметить, что *хотя представления (views) изолированы друг от друга, они совместно используют общую сущность*. Например, программисты Jim и Anne совместно используют сущность CLASS. В хорошем проекте такие связи должны учитываться, и программистам должен быть представлен набор ограничений на общие сущности. Внешние модели для Tiny College могут выглядеть так, как это показано на рис. 3.5.

Примечание

В гл. 1 был впервые представлен SQL — основной стандарт языков запросов РСУБД. При изучении рис. 3.5 имейте в виду, что при создании представления (view) для конечного пользователя в реляционной среде используется оператор CREATE VIEW языка SQL. Например, после того как вы изучите материал гл. 5, вы узнаете, что программистка Anne будет использовать в своих определениях сущности ROOM, CLASS и PROFESSOR. Поэтому она может разрабатывать приложения на основе представления с названием CLASS_VIEW, которое может быть создано следующей последовательностью SQL-команд:

```
CREATE VIEW CLASS_VIEW AS
SELECT (CLASS_ID, CLASS_NAME, PROF_NAME, CLASS_TIME, ROOM_ID)
FROM CLASS, PROFESSOR, ROOM
WHERE CLASS.PROF_ID = PROFESSOR.PROF_ID AND
      CLASS.ROOM_ID = ROOM.ROOM_ID;
```

(Предполагается, что при использовании команды CREATE VIEW таблицы и сущности ROOM, CLASS и PROFESSOR существуют.) Хотя пока мы еще не подошли к изучению SQL, мы полагаем, что вы уже смогли оценить полезность использования SQL в реальном окружении. Может быть, этот небольшой пример убедит вас, что знать SQL просто необходимо, если вы хотите научиться внедрять спроектированные вами базы данных. Считайте это примечание "прелюдией" к получению "полного удовольствия" при изучении SQL в гл. 5.

Использование внешних моделей дает следующие важные преимущества:

- ☐ если в каждую прикладную программу включать весь набор связей, имеющихся в БД, то разработка такого приложения будет очень затруднена. Использование подмножеств БД упрощает разработку прикладных программ;
- ☐ использование подмножеств облегчает задачу проектировщика, упрощая определение специфичных требований к информации, имеющихся в различных подразделениях предприятия;
- ☐ внешние модели также облегчают работу проектировщика, обеспечивая согласование с концептуальной моделью. Конкретнее, концептуальная модель проверяется на соответствие всем процессам, а также всем требованиям и ограничениям, определяемым внешней моделью (см. рис. 3.5);
- ☐ создание внешних моделей помогает обеспечить безопасность при проектировании БД. Гораздо труднее разрушить базу данных, если каждое подразделение работает только с ее подмножеством.

Внешняя модель зависит от СУБД, но не зависит от аппаратных средств, поскольку она проектируется для конкретной СУБД и не зависит от оборудования, на котором реализуется система в целом. Поэтому проектировщики баз данных могут сосредоточиться на проектировании, не заботясь об ограничениях, которые может наклады-

вать оборудование. Следовательно, при подключении и установке новых аппаратных средств хорошо спроектированные внешние схемы изменять не придется.

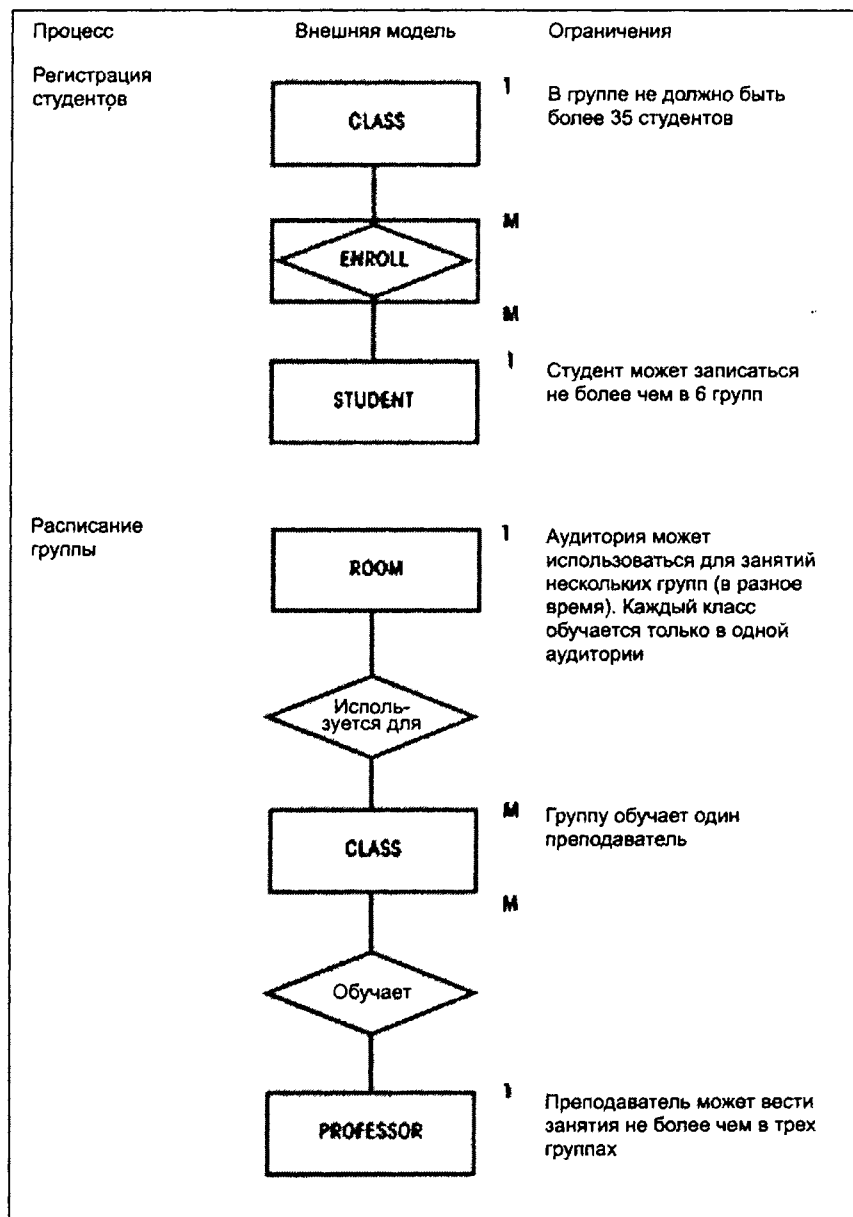


Рис. 3.5. Внешние модели колледжа Tiny College

3.2.4. Физическая модель

Физическая модель (physical model) действует на самом нижнем уровне абстракции, описывая способы хранения информации на носителях, например, жестких дисках и магнитных лентах. Физическая модель требует определения как устройства физического хранения, так и метода физического доступа, необходимого для извлечения данных с физического носителя.

Поскольку физическая модель требует конкретных формулировок, она зависит как от программного обеспечения, так и от оборудования. Используемые структуры хранения зависят от программного обеспечения (СУБД, операционная система) и от типа носителя, поддерживаемого компьютером. Конкретные требования в физической модели требуют от проектировщика БД глубоких знаний аппаратных средств и программного обеспечения, используемых при реализации проекта БД.

Иерархические модели (и, в меньшей степени, сетевые) принуждают проектировщиков принимать в расчет требования к хранению данных физической модели. Как в иерархической, так и в сетевой моделях выбор определенного метода доступа может помочь оптимизировать операции с БД; в то же время неудачно выбранный метод доступа может снизить эффективность даже хорошего концептуального проекта. Однако доминирующая сегодня реляционная модель в основном ориентирована на логический, а не на физический уровень представления, поэтому она не требует такого детального физического моделирования, как ее предшественницы.

Примечание

Хотя в реляционной модели от проектировщика не требуется детальное знание свойств физической организации хранения данных, при внедрении реляционной модели может возникнуть необходимость точной настройки для увеличения производительности БД. Точная настройка особенно важна при установке очень больших баз данных на мэйн-фреймах. Но все же даже такая настройка не требует особых знаний о физическом способе хранения данных реляционной модели. Говоря кратко, реляционная модель более дружелюбна проектировщику, нежели иерархическая и сетевая модели.

3.3. Модель "сущность-связь" (ER-модель)

Большинство базовых концепций и определений, используемых в ER-модели, были представлены в гл. 2. Например, основные компоненты сущностей и связей, а также их представление вам уже хорошо знакомы. Обсуждение ER-модели в этой главе будет отличаться расширенным описанием графического представления связей между сущностями и обсуждением возможности обработки большого количества данных для реализации хорошего проекта.

Из гл. 2 мы помним, что на основе ER-модели строятся *ER-диаграммы*. ER-диаграмма представляет собой концептуальное представление базы данных для конечного пользователя. На таких ER-диаграммах отображаются три основных компонента ER-модели: сущности, атрибуты и связи. Поскольку сущность представляет собой объект реального мира, слова "сущность" и "объект" часто обозначают одно и

то же. Таким образом, сущности проекта базы данных колледжа Tiny College, который мы будем разрабатывать в этой главе, представляют собой студентов, группы, преподаватели, аудитории и т. д. Порядок, в котором компоненты ER-диаграмм рассматриваются в этой главе, определяется способом использования инструмента моделирования ER-диаграммы, который может быть применен в данном случае.

3.3.1. Сущности

В гл. 2 мы указывали, что под *сущностью* на уровне ER-моделирования на самом деле подразумевается *набор сущностей* (entity set), а не единственная сущность. *Иначе говоря, слово "сущность" в ER-моделировании соответствует таблице, а не строке в реляционной среде.* В ER-модели отдельная строка таблицы называется *экземпляром сущности* (entity instance, entity occurrence). Как в модели Чена, так и в модели "птичья лапка", сущность изображается прямоугольником, в котором записано имя сущности. Имя сущности — имя существительное — обычно записывается заглавными буквами. Хотя ER-диаграммы "птичья лапка", которые вы увидите в этой главе, соответствуют общепринятым стандартам моделирования, чаще всего программное обеспечение, с помощью которого они создаются, допускает выбор различных форм представления. Например, имя сущности можно выделить жирным шрифтом, прямоугольник, изображающий сущность, можно выделить цветом, и т. д.

3.3.2. Атрибуты

В гл. 2 было показано, что атрибуты описывают свойства сущностей. Например, сущность STUDENT включает в себя атрибуты STU_LNAME (фамилия студента), STU_FNAME (имя студента), STU_INITIAL (инициалы студента). В модели Чена атрибуты изображаются овалами, соединенными линиями с прямоугольником, изображающим сущность. В каждом овале содержится имя того атрибута, который он представляет. В модели "птичья лапка" атрибуты просто записываются в прямоугольнике атрибутов, присоединенном снизу к прямоугольнику сущности (рис. 3.6). Поскольку представление диаграмм с помощью модели Чена требует значительно больше места, некоторые поставщики программного обеспечения, создающего диаграммы Чена, используют стиль изображения атрибутов, принятый в модели "птичья лапка".

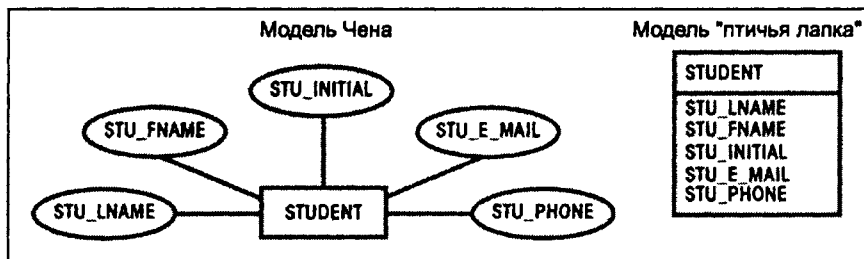


Рис. 3.6. Атрибуты сущности STUDENT:
модели Чена и "птичья лапка"

Домены

У атрибутов имеются домены. *Домен* это набор возможных значений атрибута. Например, домен для числового атрибута средней оценки студента (Grand Point Average, GPA) может быть записан в виде интервала [0,4], поскольку минимальное значение GPA равно 0, а максимальное значение — 4. Домен для символьного атрибута SEX (пол) состоит только из двух возможных букв М (Male, мужской) или F (Female, женский) или какого-либо другого допустимого обозначения. Домен даты приема на работу в компанию состоит из всех дат, входящих в определенный диапазон (например, от даты создания компании до текущей даты).

Домен может совместно использоваться атрибутами. Например, адреса студентов и адреса преподавателей совместно используют домен всех возможных адресов. На самом деле, словарь данных позволяет новому атрибуту наследовать характеристики существующего атрибута, если для атрибута используется то же самое имя. Например, сущности PROFESSOR и STUDENT могут каждая иметь атрибут с именем ADDRESS (адрес).

Первичные ключи

Первичные ключи (Primary Key, PK) на ER-диаграммах подчеркиваются. Ключевые атрибуты также подчеркиваются при текстовой записи табличных структур, где, как правило, используется следующий формат:

ИМЯ ТАБЛИЦЫ (КЛЮЧЕВОЙ АТРИБУТ 1, АТРИБУТ 2, АТРИБУТ 3 ... АТРИБУТ K)

Например, сущность CAR (автомобиль) может быть представлена так:

CAR (CAR_VIN MOD_CODE, CAR_YEAR, CAR_COLOR)

(VIN — это стандартный акроним для идентификационного номера транспортного средства, vehicle identification number).

С учетом соглашения об именах из гл. 2 можно заметить, что атрибут с именем MOD_CODE (код модели), по всей видимости, является внешним ключом (foreign key, FK), используемым для связывания таблицы автомобилей (CAR) с какой-то другой таблицей. В этом случае другая таблица, допустим, с именем MODEL (модель), возможно, содержит атрибуты, специфичные для каждой модели (например, MOD_TYPE), в которых хранятся такие описательные элементы, как "двухдверный спортивный купе", "четыrehдверный седан" или "кузов-универсал".

В идеальном случае первичный ключ (PK) состоит из единственного атрибута. Например, в таблице на рис. 3.7 в качестве первичного ключа используется единственный атрибут CLASS_CODE. Однако можно использовать и *составной ключ*, т. е. первичный ключ, состоящий более чем из одного атрибута. Например, администратор базы данных колледжа Tiny College может принять решение идентифицировать каждый экземпляр сущности CLASS (группа) с помощью составного первичного ключа, представляющего собой комбинацию из атрибутов CRS_CODE и CLASS_SECTION, вместо использования одного атрибута CLASS_CODE. При любом подходе каждый экземпляр сущности уникально идентифицируется первичным ключом. В структуре таблицы CLASS, представленной на рис. 3.7, атрибут CLASS_CODE является первичным ключом.

чом, а комбинация атрибутов CRS_CODE и CLASS_SECTION представляет собой допустимый потенциальный ключ. Если атрибут CLASS_CODE удалить из сущности CLASS, то потенциальный ключ (CRS_CODE и CLASS_SECTION) станет подходящим составным первичным ключом.

Примечание

Напомним, что в гл. 2 мы указали на отличие курса (COURSE) от группы (CLASS). Группа (CLASS) входит в определенный курс (COURSE) и определяется описанием курса и его секциями. Таким образом, преподаватель, читающий лекции по группам "Базы данных I, Секция 2"; "Базы данных I, Секция 5"; "Базы данных I, Секция 8" и "Таблицы II, Секция 6" читает два курса (Базы данных I и Таблицы II), но ведет занятия в четырех группах (CLASS)! Обычно предлагаемые курсы (COURSE) печатаются в каталоге курсов, в то время как предлагаемые группы (CLASS) печатаются каждый раз во время регистрации групп (семестр, триместр, четверть) в соответствии с расписанием.

	CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
▶	10012	ACCT-211	1	MWTF 8:00-8:50 a.m.	BUS311	105
	10013	ACCT-211	2	MWTF 9:00-9:50 a.m.	BUS200	105
	10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
	10015	ACCT-212	1	MWTF 10:00-10:50 a.m.	BUS311	301
	10016	ACCT-212	2	Th 6:00-8:40 p.m.	BUS252	301
	10017	CIS-220	1	MWTF 9:00-9:50 a.m.	KLR209	228
	10018	CIS-220	2	MWTF 9:00-9:50 a.m.	KLR211	114
	10019	CIS-220	3	MWTF 10:00-10:50 a.m.	KLR209	228
	10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
	10021	QM-261	1	MWTF 8:00-8:50 a.m.	KLR200	114
	10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
	10023	QM-382	1	MWTF 11:00-11:50 a.m.	KLR200	162
	10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

Рис. 3.7. Компоненты и содержимое таблицы (сущности) CLASS

Если на рис. 3.7 атрибут CLASS_CODE используется в качестве первичного ключа, то сущность CLASS может быть представлена в текстовом виде следующим образом:

CLASS (CLASS_CODE, CRS_CODE, CLASS_SECTION, CLASS_TIME,
CLASS_ROOM, PROF_NUM)

С другой стороны, если атрибут CLASS_CODE удалить и использовать в качестве первичного ключа комбинацию атрибутов CRS_CODE и CLASS_SECTION, то сущность CLASS можно выразить так:

CLASS (CRS_CODE, CLASS_SECTION, CLASS_TIME, CLASS_ROOM,
PROF_NUM)

Обратите внимание, что в обозначении этой сущности подчеркнуты *оба* ключевых атрибута.

Составные и простые атрибуты

Атрибуты подразделяются на простые и составные. *Составной атрибут (composite attribute)* (не путать с составным ключом!) это атрибут, который может быть в дальнейшем разделен на несколько дополнительных атрибутов. Например, атрибут ADDRESS (адрес) можно разделить на *street* (улица), *city* (город), *state* (штат) и *zip code* (почтовый индекс). Точно так же атрибут PHONE_NUMBER может быть разделен на *area code* (код города) и *exchange number* (городской номер телефона). *Простой атрибут (simple attribute)* не может быть разделен на атрибуты. Например, *age* (возраст), *sex* (пол) и *marital status* (семейное положение) можно считать простыми атрибутами. Чтобы облегчить детализацию запроса, обычно стоит заменить составные атрибуты на несколько простых.

Однозначные атрибуты

Однозначный атрибут (single-valued attribute) это атрибут, который может принимать единственное значение. Например, человек может иметь только один номер социального страхования (social security number, SSN), изделие может иметь только один серийный номер. Помните, что *однозначные атрибуты не обязательно являются простыми атрибутами*. Например, серийный номер изделия SE-08-02-189935 — однозначный атрибут, но в то же время это составной атрибут, поскольку его можно разделить на регион, в котором изделие было выпущено (SE), код завода в этом регионе (08), выпускающую смену (02) и номер изделия (189935).

Многозначные атрибуты

Многозначный атрибут (multivalued attribute) это атрибут, который может принимать множество значений. Например, человек может окончить несколько колледжей, семья может иметь несколько телефонных номеров. Точно так же цвет машины можно разделить на цвет крыши, корпуса и отделки. В ER-модели Чена многозначные атрибуты показываются двойной линией, связывающей атрибут и сущность. В модели "птичья лапка" многозначные атрибуты никак не выделяются. ER-диаграмма на рис. 3.8 содержит все представленные нами ранее компоненты. Если внимательно посмотреть на рис. 3.8, то можно заметить, что атрибут CAR_VIN является первичным ключом (PK), а CAR_COLOR — это многозначный атрибут сущности CAR. (Предполагается, что каждый автомобиль идентифицируется уникальным номером CAR_VIN.)

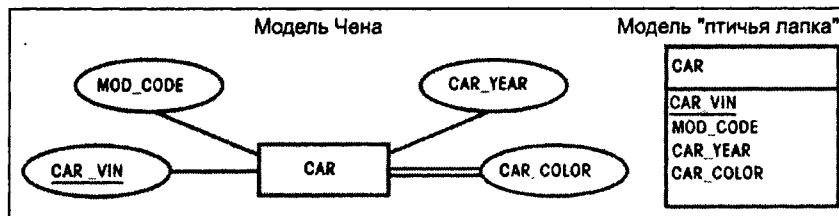


Рис. 3.8. Многозначный атрибут сущности

Хотя концептуальная модель может работать со связями M:N и многозначными атрибутами, вы не должны использовать их в реляционных СУБД. Поэтому если имеются многозначные атрибуты, проектировщик должен предпринять одно из следующих действий:

- создать внутри данной сущности несколько новых атрибутов, по одному на каждый компонент многозначного атрибута. Например, можно разделить атрибут CAR_COLOR (цвет машины) сущности CAR (машина) и создать новые атрибуты CAR_TOPCOLOR (цвет верха), CAR_BODYCOLOR (цвет корпуса) и CAR_TRIMCOLOR (цвет отделки), как показано на рис. 3.9, и ввести их в сущность CAR;

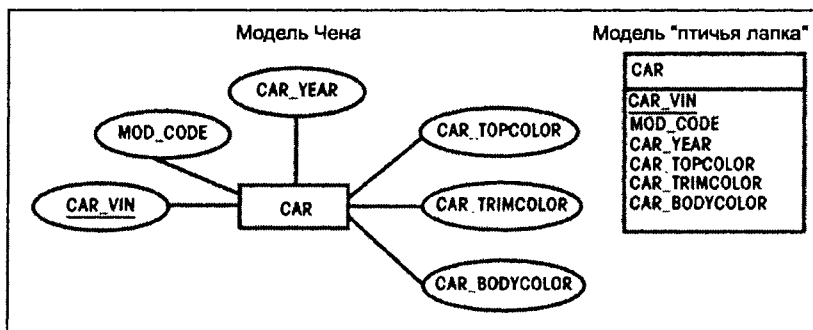


Рис. 3.9. Разбиение многозначного атрибута на новые атрибуты

- создать новую сущность, состоящую из компонентов многозначного атрибута (рис. 3.10). Новую независимую сущность **COLOR** затем необходимо связать с исходной сущностью **CAR** связью 1:M. Обратите внимание, что такое решение позволяет проектировщику определить цвет для множества различных частей машины (табл. 3.1).

Таблица 3.1. Компоненты многозначного атрибута

Часть кузова	Цвет
Top (верх)	White (белый)
Body (корпус)	Blue (синий)
Trim (отделка)	Green (зеленый)
Interior (салон)	Blue (синий)

Используя подход, представленный в табл. 3.1, мы получаем дополнительное преимущество: теперь можно назначить цвет любому количеству частей кузова без изменения структуры таблицы. Обратите внимание, что в ER-модели на рис. 3.10 отображены компоненты, перечисленные в табл. 3.1.

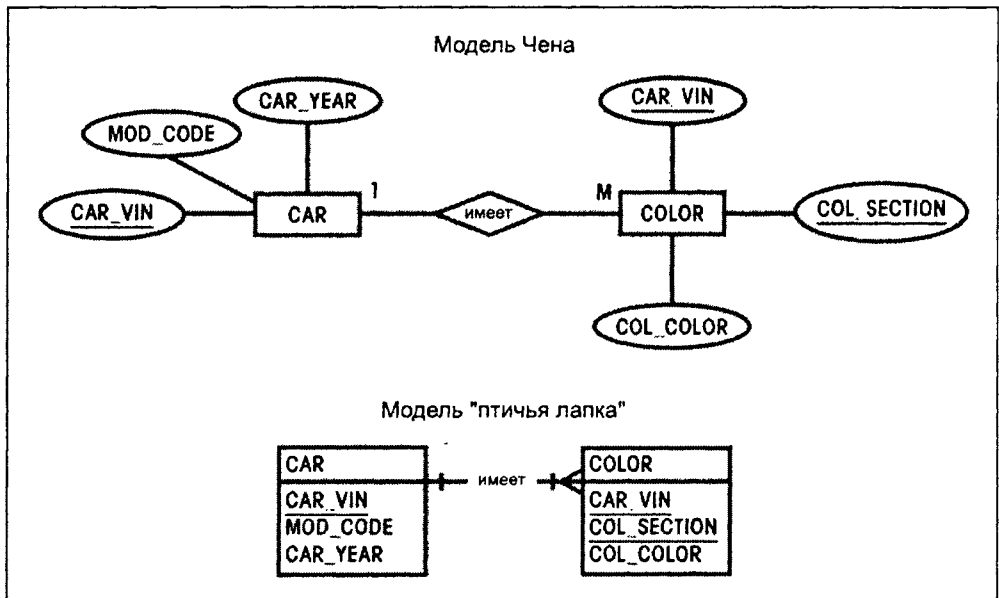


Рис. 3.10. Новый набор сущностей, составленный из компонентов многозначного атрибута

Производные атрибуты

Наконец, атрибут может быть производным атрибутом. *Производный атрибут (derived attribute)* это атрибут, который не нужно хранить в базе данных; вместо этого его получают с помощью некоторого алгоритма. Например, возраст служащего **EMP_AGE** можно получить как целое значение разности между текущей датой и датой рождения служащего (**EMP_DOB**). Если вы используете MS Access, то можно записать такое выражение:

`INT((DATE() - EMP_DOB)/365)`

Если вы используете Oracle, то вместо **DATE()** можно воспользоваться функцией **SYSDATE**. (Предполагается, конечно, что в атрибуте **EMP_DOB** даты хранятся в юлианском формате.) Точно так же общую стоимость заказа можно получить умножением количества заказанных единиц на цену одной позиции; оценку средней скорости можно получить делением общей протяженности пути на время прохождения маршрута. Производный атрибут в модели Чена обозначается штриховой линией, соединяющей атрибут и сущность (рис. 3.11). В модели "птичья лапка" нет специального обозначения для производного атрибута. Принятие решения о хранении производных атрибутов в таблицах БД зависит от требований и ограничений, установленных для отдельного приложения. Проектировщик должен привести проект в соответствие с этими ограничениями.

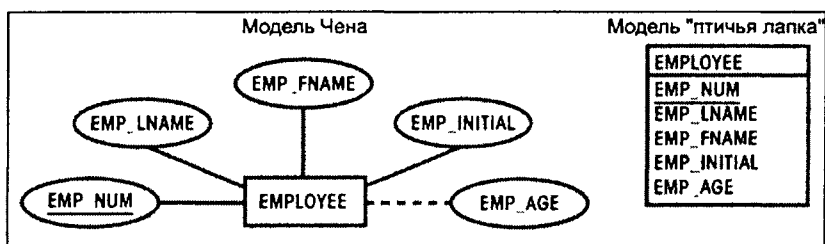


Рис. 3.11. Отображение производного атрибута

3.3.3. Связи

Связь (relationship) это ассоциирование сущностей. Сущности, участвующие в связи, называются *участниками (participants)*. Вы помните из гл. 2, что каждая связь идентифицируется описательным названием. В качестве названия связи используется глагол в активной или пассивной форме. Например, "студент (STUDENT) *занимается* в группе (CLASS)", "преподаватель (PROFESSOR) *ведет занятия* в группе (CLASS)", "факультет (DEPARTMENT) *нанимает* преподавателя (PROFESSOR)", "отделом (DIVISION) *руководит* служащий (EMPLOYEE)", "самолет (AIRCRAFT) *управляется* экипажем (CREW)".

Связи между сущностями всегда действуют в обоих направлениях, т. е. чтобы установить связь между сущностями CUSTOMER и INVOICE, необходимо определить, что:

- ☐ клиент (CUSTOMER) может создать несколько счетов (INVOICE);
- ☐ каждый счет (INVOICE) создается одним клиентом (CUSTOMER).

Поскольку мы знаем оба направления связи между CUSTOMER и INVOICE, легко увидеть, что эту связь нужно отнести к типу 1:M.

Связь трудно классифицировать, если известна только одна ее сторона. Например, если вы определите, что:

- ☐ отделом (DIVISION) руководит служащий (EMPLOYEE),

то вы не знаете, относится ли эта связь к типу 1:1 или 1:M. Поэтому здесь необходимо задаться вопросом: "Может ли служащий руководить более чем одним отделом?" Если ответ на этот вопрос положительный (да), то это связь 1:M и вторая сторона связи записывается так:

- ☐ служащий (EMPLOYEE) может руководить несколькими отделами (DIVISION).

Если сотрудник не может управлять более чем одним отделом, то это связь 1:1 и вторая часть связи должна быть записана так:

- ☐ служащий (EMPLOYEE) может управлять только одним отделом (DIVISION).

3.3.4. Связность и мощность связи

В гл. 2 вы узнали, что связи между сущностями можно классифицировать как "один-к-одному", "один-ко-многим" и "многие-ко-многим", еще было показано, как

такие связи отображаются в моделях Чена и "птичья лапка". Для обозначения типов связей используется термин *связность* (*connectivity*).

Мощность связи (*cardinality*) выражает определенное число экземпляров сущностей, связанных с одним экземпляром связанной сущности. В модели Чена мощность связи обозначается указанием соответствующих чисел рядом с сущностями в формате (x,y) . Первое число представляет собой минимальное значение мощности связи, второе — ее максимальное значение.

Сведения о максимальном и минимальном количествах экземпляров сущности могут пригодиться в прикладном программном обеспечении. Например, колледж Tiny College может поставить условие, что группа не может начать занятия, пока в нее не запишется, по крайней мере, десять студентов. Точно так же если аудитория может вместить только тридцать студентов, то прикладная программа должна использовать значение мощности связи для ограничения списка группы. Однако помните, что СУБД не может оперировать с мощностью связи на уровне таблиц — эта возможность предоставляется программным обеспечением или триггерами. О том, как создавать и применять триггеры, мы поговорим в гл. 5.

При исследовании модели Чена, представленной на рис. 3.12, нужно помнить, что мощность указывает на число экземпляров в связанной сущности.

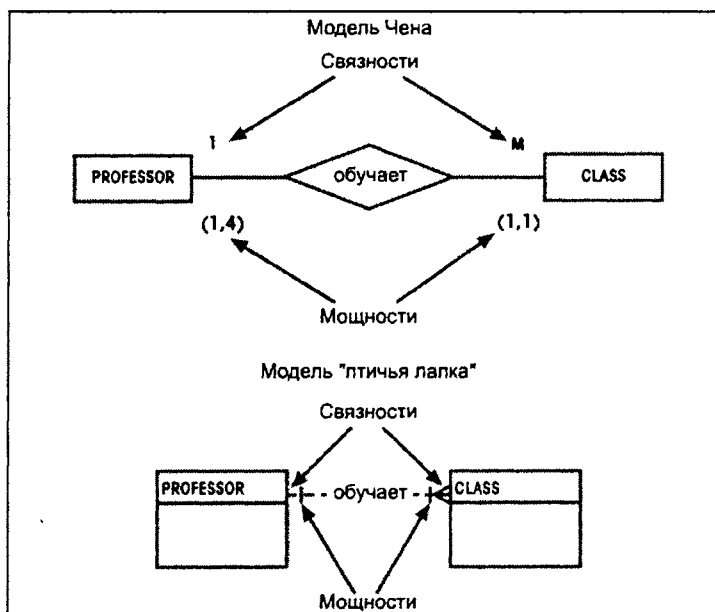


Рис. 3.12. Связность и мощность в ER-диаграммах

Например, мощность $(1,4)$ записанная рядом с сущностью PROFESSOR в связи "преподаватель (PROFESSOR) ведет занятия в группе (CLASS)", показывает, что значение внешнего ключа (FK) таблицы PROFESSOR встречается, по крайней мере, один

раз, но не более четырех раз в таблице CLASS. Если бы мощность была записана в виде (1,N), то это означало бы отсутствие ограничения на количество групп, которые ведет преподаватель. Точно так же мощность (1,1), записанная рядом с сущностью CLASS, указывает на то, что в каждой группе вести занятия может один и только один преподаватель. (Каждый экземпляр сущности — т. е. строка — в таблице PROFESSOR может встречаться один и только один раз для каждого преподавателя).

В стандартной модели "птичья лапка" числовой диапазон значений мощности не отображается на ER-диаграммах. Программное обеспечение Visio, с помощью которого мы создавали ER-диаграммы, соответствует именно этому стандарту. Например, обратите внимание, что мощность связи по Чену (1,4) на фрагменте диаграммы "птичья лапка" (см. рис. 3.12) представляет "4" как символ "многие". Также в отличие от модели Чена, на которой отображается число экземпляров в связанной таблице, мощность связи в модели "птичья лапка" соответствует количеству экземпляров сущностей в порождающей таблице. (Сравните два изображения на рис. 3.12.)

Примечание

Связность и мощность связи определяются очень лаконичными утверждениями, которые называются *бизнес-правилами* (*business rules*). Эти правила, полученные на основе точных и детальных описаний информационной среды предприятия, также устанавливают сущности, атрибуты, связи, и ограничения в ER-моделях. Поскольку бизнес-правила определяют все компоненты ER-модели, тщательное, полное и точное выявление всех необходимых бизнес-правил является очень важной частью проектирования БД, поэтому мы подробно исследуем вывод этих правил в гл. 7. А навыки моделирования, которые вы получите при изучении гл. 3, пригодятся при разработке реальной базы данных в гл. 7.

3.3.5. Сила связей

Вспомним, что связи устанавливаются между сущностями. Для лучшего понимания связей необходимо знать свойства сущностей, что поможет определить эти связи. Поэтому перед исследованием основ классификации связей мы должны кратко обобщить, как в структуру связей вписываются имеющиеся зависимости.

Зависимость существования

Если сущность зависит от существования одной или более других сущностей, то говорят, что она *зависима от существования* (*existence-dependent*). Например, если сотрудник корпорации XYZ имеет одного или более иждивенцев, то для исчисления налогов можно установить связь "сотрудник (EMPLOYEE) имеет иждивенца (DEPENDENT)". В этом случае сущность DEPENDENT (иждивенец), очевидно, зависит от наличия сущности EMPLOYEE (сотрудник), поскольку невозможно существование иждивенца отдельно от сотрудника базы данных компании XYZ.

Если сущность может существовать вне одной или более связанных сущностей, то говорят, что она *независима от существования* (*existence-independent*). Например, предположим, что корпорация XYZ использует какие-то детали при выпуске своей продукции. Далее предположим, что некоторые из этих деталей производятся "у себя", а другие покупаются у поставщиков. При таком сценарии возможно, что деталь (PART) существует независимо от поставщика (VENDOR) в связи "деталь (PART)

приобретается у поставщика (VENDOR)". (Как-никак, по крайней мере некоторыми деталями предприятие будет обеспечено и без поставщика). Поэтому сущность PART не зависит от наличия сущности VENDOR.

Слабые (неидентифицируемые) связи

Если одна сущность независима от существования другой сущности, связь между ними называется *слабой связью* (*weak relationship*), также называемой *неидентифицируемой связью* (*non-identifying relationship*). С точки зрения проектирования БД слабые связи имеют место, если PK связанной сущности не содержит первичные компоненты порождающей сущности. Предположим, что мы определили сущности COURSE (курс) и CLASS (группа) как

COURSE(CRS_CODE, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT)

CLASS(CLASS_CODE, CRS_CODE, CLASS_SECTION, CLASS_TIME, ... и т. д.)

В этом случае между сущностями COURSE и CLASS есть слабая связь, поскольку атрибут CLASS_CODE является PK сущности CLASS, в то время как атрибут CRS_CODE сущности CLASS является только FK. В данном случае PK сущности CLASS не наследует компонент первичного ключа из сущности COURSE.

На рис 3.13 показано, что в модели "птичья лапка" слабая связь изображается штриховой линией связи между связанными сущностями. В модели Чена не делается различий между слабой и сильной связями.

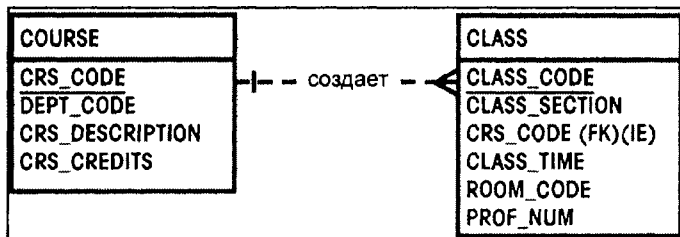


Рис. 3.13. Слабая (неидентифицируемая) связь между сущностями COURSE и CLASS

На рис. 3.13 мы видим, что FK CRS_CODE помечен как "(FK)(IE)". Сгенерированное приложением Visio обозначение (IE) указывает на существование инвертированного входа (Inversion Entry, IE), который определен в Visio как "неуникальный идентификатор доступа к сущности". Далее будет показано, что обозначение IE всегда появляется в слабых (неидентифицируемых) связях.

Примечание

Если вы привыкли видеть реляционные схемы, сгенерированные, например, в MS Access, то вы ожидаете увидеть линии связи в реляционной схеме, протянутые от PK к FK. Однако соглашения реляционных схем не обязательно находят отражение в ER-диаграммах. В ER-диаграммах все сконцентрировано на сущностях и связях между ними, а не на способе реализации этих связей. На самом деле, если CASE-инструментарий ти-

на VISIO используется правильно, связи в диаграммах модели "птичья лапка" должны устанавливаться *перед* созданием FK с помощью команды "обновить FK", поэтому невозможно закрепить линию связи на FK до завершения обновления. (Такое обновление гарантирует, что свойства атрибута FK всегда соответствуют свойствам атрибута PK, на который указывает FK.) Вы можете, конечно, переместить точку привязки линии связи после завершения обновления — но такое решение, очевидно, отражает ваш выбор, а не реальное положение вещей. Далее будет показано, что размещение линий связи в сложных ER-диаграммах, включающих как горизонтальное, так и вертикальное расположение сущностей, в основном определяется решением проектировщика в целях повышения "читабельности" проекта. (Помните, что ER-диаграмма используется для взаимодействия между проектировщиком и конечными пользователями.) В связи с этим мы склоняемся к размещению точек закрепления линий связи в позициях FK/PK только в том случае, если они не снижают "читабельность" диаграммы.

Примеры таблиц, между которыми существуют слабые связи, приведены на рис. 3.14.

База данных: CH3_TEXT				
Таблица: COURSE				
	CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
▶	ACCT-211	ACCT	Accounting I	3
	ACCT-212	ACCT	Accounting II	3
	CIS-220	CIS	Intro. to Microcomputing	3
	CIS-420	CIS	Database Design and Implementation	4
	MATH-243	MATH	Mathematics for Managers	3
	QM-261	CIS	Intro. to Statistics	3
	QM-362	CIS	Statistical Applications	4

Таблица: CLASS						
	CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	ROOM_CODE	PROF_NUM
▶	10012	ACCT-211	1	MWF 8:00-8:50 a.m.	BUS311	105
	10013	ACCT-211	2	MWF 9:00-9:50 a.m.	BUS200	105
	10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
	10015	ACCT-212	1	MWF 10:00-10:50 a.m.	BUS311	301
	10016	ACCT-212	2	Th 8:00-8:40 p.m.	BUS252	301
	10017	CIS-220	1	MWF 9:00-9:50 a.m.	KLR209	228
	10018	CIS-220	2	MWF 9:00-9:50 a.m.	KLR211	114
	10019	CIS-220	3	MWF 10:00-10:50 a.m.	KLR209	228
	10020	CIS-420	1	W 6:00-8:40 p.m.	KLR209	162
	10021	QM-261	1	MWF 8:00-8:50 a.m.	KLR200	114
	10022	QM-261	2	TTh 1:00-2:15 p.m.	KLR200	114
	10023	QM-362	1	MWF 11:00-11:50 a.m.	KLR200	162
	10024	QM-362	2	TTh 2:30-3:45 p.m.	KLR200	162

Рис. 3.14. Иллюстрация слабых связей между сущностями COURSE и CLASS

Сильные (идентифицируемые) связи

Сильная связь (*strong relationship*), также называемая *идентифицируемой связью* (*identifying relationship*), имеет место, если связанные сущности зависимы от существования. С точки зрения проектирования базы данных сильная связь между двумя

сущностями имеет место всякий раз, когда PK связанной сущности содержит компонент PK порождающей сущности. Например, определения сущностей COURSE и CLASS

COURSE(CRS_CODE, DEPT_CODE, CRS_DESCRIPTION, CRS_CREDIT)

CLASS(CRS_CODE, CLASS_SECTION, CLASS_TIME, ... и т. д.)

указывают на то, что между сущностями COURSE и CLASS есть сильная связь, поскольку составной PK сущности CLASS включает в себя CRS_CODE и CLASS_SECTION. (Обратите внимание, что CRS_CODE в таблице CLASS является также FK для сущности COURSE.)

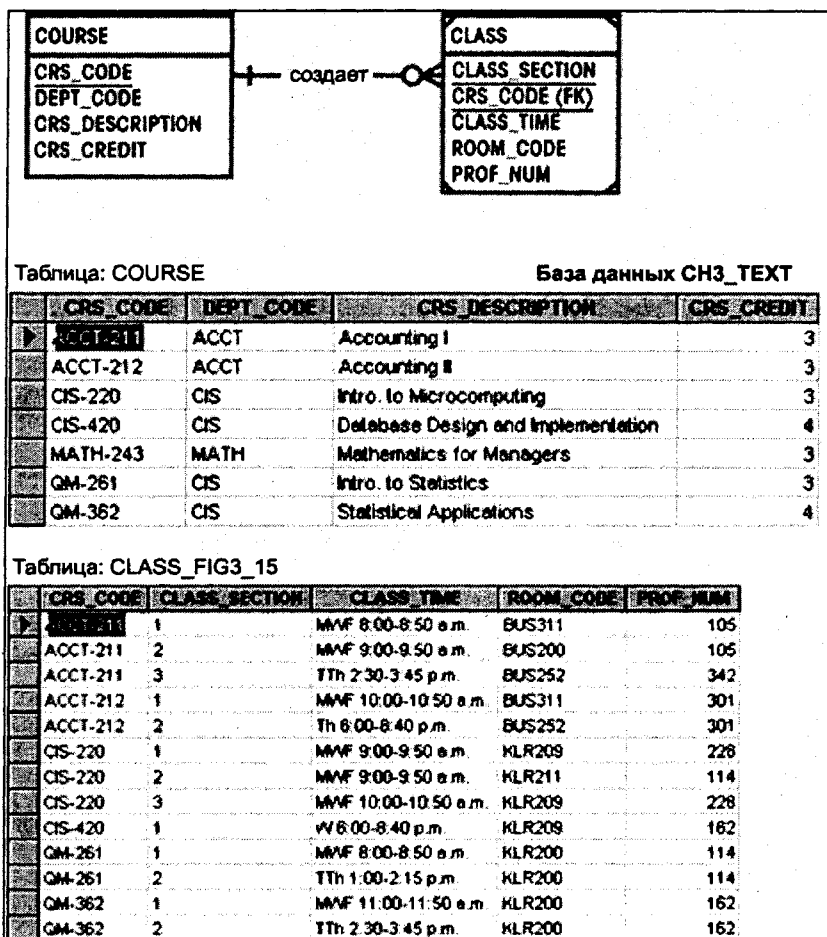


Рис. 3.15. Сильная (идентифицируемая) связь между COURSE и CLASS

На рис. 3.15 показано, что в модели "птичья лапка" сильные (идентифицируемые) связи показываются сплошными линиями связи между сущностями. Существование сильной связи также оказывает воздействие на отображение самой сущности. Обратите внимание, что сущность CLASS на рис. 3.15 имеет по углам прямоугольника небольшие дополнительные сегменты. Обозначение внешнего ключа в сильной связи также выглядит, как (FK), а не как (FK)(IE), которое использовалось в слабой связи на рис. 3.13. Будет ли связь между COURSE и CLASS сильной или слабой, зависит от того, как определен первичный ключ сущности CLASS.

Необходимо иметь в виду, что порядок, в котором таблицы создаются и загружаются, имеет существенное значение. Например, в связи "на курсе (COURSE) создаются группы (CLASS)" таблица COURSE должна быть создана до таблицы CLASS. В конце концов, неприемлема ситуация, когда внешний ключ таблицы CLASS ссылается на еще не существующую таблицу COURSE. В некоторых СУБД проблема последовательности создания таблиц не возникает до тех пор, пока в таблицы не загружаются данные. На самом деле, чтобы избежать нарушения целостности на уровне ссылки, в связи 1:M вы должны, прежде всего, загружать сторону "1", независимо от того, является ли связь сильной или слабой.

На рис. 3.15 может вызвать удивление символ "кружок" рядом с обозначением сущности CLASS. В следующем разделе мы расскажем о его предназначении.

Помните, что природа связи зачастую определяется проектировщиком БД, который должен полагаться на свою профессиональную интуицию при определении типа связи, лучше всего подходящего для эффективного выполнения транзакций и полное удовлетворяющего требования заказчика. Мы будем часто обращаться на это ваше внимание!

3.3.6. Участие в связи

Участие сущности в связи может быть либо необязательным, либо обязательным. Участие сущности *необязательно* (optional participation), если один экземпляр сущности не требует наличия соответствующего экземпляра сущности в отдельной связи. Например, в связи "на курсе (COURSE) создаются группы (CLASS)", по крайней мере, на некоторых курсах могут и не создаваться группы. Иначе говоря, экземпляр сущности (строка) в таблице COURSE не требует обязательного наличия соответствующего экземпляра сущности в таблице CLASS. (Помните, что каждая сущность реализуется в виде таблицы.) Поэтому сущность CLASS рассматривается как необязательная по отношению к сущности COURSE. Как в диаграммах Чена, так и в диаграммах "птичья лапка" необязательная связь между сущностями показывается небольшим кружком со стороны необязательной сущности (см. рис. 3.15). Существование *необязательности* (optionality) указывает на то, что для необязательной сущности минимальное значение мощности связи равно 0. (Термин "необязательность" используется для обозначения условия, при котором существует одна или более необязательных связей.)

Примечание

Необходимо помнить, что ответственность за установление связи всегда ложится на сущность, содержащую внешний ключ. В большинстве случаев это будет сущность со стороны "многие".

Участие сущности в связи *обязательно* (*mandatory participation*), если один экземпляр сущности обязательно требует соответствующего экземпляра сущности в отдельной связи. Если около сущности не изображен никакой дополнительный символ, то это означает, что данная сущность участвует в обязательной связи со связанной сущностью. Наличие обязательной связи указывает на то, что для обязательной сущности минимальная мощность связи равна 1.

Примечание

Вам может показаться, что связи между необязательными сущностями являются слабыми, а между обязательными — сильными. К сожалению, это суждение не всегда верно. Помните, что участие в связи и сила связи это не одно и то же. Весьма возможно наличие сильной связи между сущностями, когда одна из них является необязательной по отношению к другой. Например, связь между EMPLOYEE (сотрудник) и DEPENDENT (иждивенец) очевидно сильная, но также очевидно, что DEPENDENT для EMPLOYEE необязателен. И в самом деле, вы не можете требовать от сотрудника, чтобы он имел иждивенцев. Возможно также наличие слабой связи между сущностями, когда одна из них обязательна для другой. *Сила связи зависит от того, как формируется РК связанной сущности, в то время как участие в связи зависит от того, как составлены бизнес-правила.* Например, бизнес-правила

Каждая деталь должна быть получена от поставщика

и

Деталь может или не может быть получена от поставщика

по-разному влияют на *необязательность* для одних и тех же сущностей! Непонимание этого различия может стать причиной плохого проекта, а также вызвать большие проблемы при вставке и удалении строк в таблицах.

Поскольку участие в связи — столь важный компонент процесса проектирования БД, рассмотрим еще несколько сценариев. Предположим, что колледж Tiny College принял на работу несколько преподавателей, которые ведут исследования, но не проводят занятия в группах. Если рассмотреть связь "преподаватель (PROFESSOR) обучает группу (CLASS)", то вполне возможен вариант, что преподаватель (PROFESSOR) и не проводит занятия с группой (CLASS). Следовательно, CLASS — *необязательная* сущность для сущности PROFESSOR. С другой стороны, группа (CLASS) должна заниматься у преподавателя (PROFESSOR). Следовательно, сущность PROFESSOR *обязательна* для сущности CLASS. Обратите внимание, что в модели Чена на рис. 3.16 рядом с сущностью PROFESSOR показана мощность связи (0,3), указывающая на то, что преподаватель может не вести занятия в группах вообще или же вести занятия не более чем в трех группах. В модели "птичья лапка" не определены точные пределы мощности связи, здесь указаны минимальная мощность (0) и неопределенная максимальная (M). А в модели Чена указывается число экземпляров сущности в связанной таблице. Поэтому модель Чена показывает, что в таблице CLASS либо вообще не будет связей с сущностью PROFESSOR (0), либо их будет не более трех. И каждая строка таблицы CLASS ссылается на одну и только одну строку сущности PROFESSOR — исходя из того, что в каждой группе ведет занятия только один преподаватель.

Непонимание различия между обязательным и необязательным участием в связи может стать причиной создания проектов, в которых придется размещать неуклю-

жие (и ненужные) временные строки (экземпляры сущностей) только для того, чтобы оправдать создание требуемых сущностей. Поэтому очень важно понимать основные идеи обязательного и необязательного участия в связи.

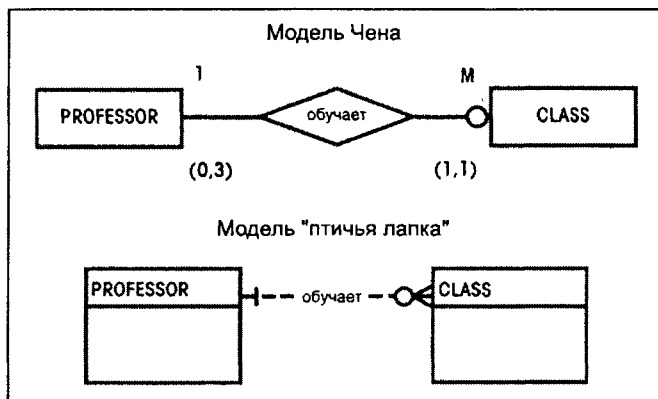


Рис. 3.16. Необязательная сущность CLASS в связи "преподаватель (PROFESSOR) ведет занятия в группе (CLASS)"

Также очень важно понимать, что семантика задачи может определять тип участия в связи. Например, Tiny College предлагает несколько курсов; каждый курс имеет несколько групп. Опять-таки, обратите здесь внимание на различие между *группой* (class) и *курсом* (course): в группе (CLASS) ведутся занятия по определенным темам (разделам) данного курса (COURSE). (Обычно курсы описаны в каталоге университета, в то время как темы занятий в группах описаны в расписании занятий, которое студенты используют при записи в группы.)

Анализируя вклад сущности CLASS в связь "на курсе (COURSE) создаются группы (CLASS)", можно заметить, что сущность CLASS не может существовать без сущности COURSE. Следовательно, сущность COURSE *обязательна* в этой связи. Но мы можем предположить два сценария для сущности CLASS, показанные на рис. 3.17 и 3.18. Различные сценарии зависят от семантики задачи, т. е. они зависят от того, как определены связи.

- Сущность CLASS *необязательна*. Возможно, на факультете сначала создается сущность COURSE, а затем после назначения преподавателей создается сущность CLASS. В действительности такой сценарий вполне вероятен; могут иметься курсы, для которых разделы (группы) еще не определены. На самом деле, некоторые курсы могут читаться только раз в году и при этом группы создаются не в каждом семестре.
- Сущность CLASS *обязательна*. Это условие создается посредством ограничения, которое выражается семантической конструкцией вида "на каждом курсе (COURSE) создается одна или более групп (CLASS)". В терминах ER в каждой группе (COURSE) созданной связи должна иметься, по крайней мере, одна группа (CLASS). Следовательно, при создании курса (COURSE) всегда должна создаваться группа (CLASS) для увязки с семантикой задачи.

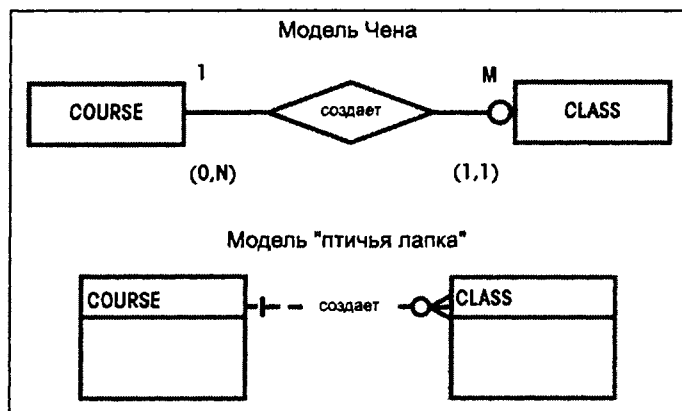


Рис. 3.17. Сущность CLASS необязательна для сущности COURSE

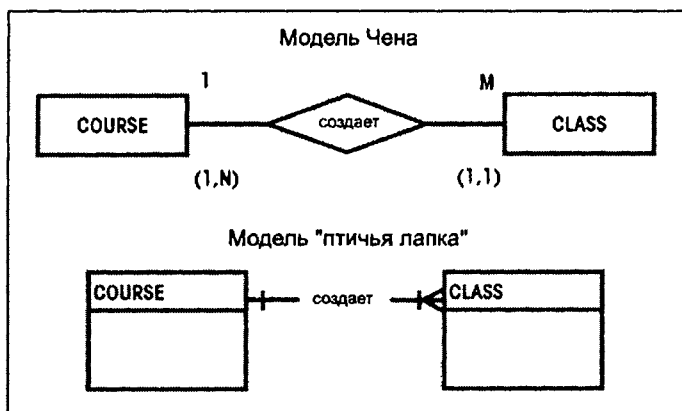


Рис. 3.18. Сущности COURSE и CLASS в обязательной связи

Помните о практической стороне сценария, представленного на рис. 3.18. Учитывая семантику этой связи, в систему не должны включаться курсы, на которых нет, по крайней мере, одного раздела (группы). Желательна ли такая жесткая конфигурация с оперативной точки зрения? Например, когда мы создаем новый курс (COURSE) в базе данных, прежде всего, обновляется таблица COURSE, тем самым вводится сущность COURSE, которая еще не имеет связанной с ней группы (CLASS). Естественно, эта проблема снимается, когда сущности CLASS будут вставлены в соответствующую таблицу CLASS. Однако при такой семантике обязательной связи система будет находиться в состоянии временного несоответствия бизнес-правилам. На практике для большей гибкости проекта желательно определить сущность CLASS как необязательную.

Наконец при изучении сценариев, представленных на рис. 3.17 и 3.18, помните о роли СУБД. Для обеспечения целостности данных СУБД должна обеспечивать, чтобы сторона "многие" (CLASS) ассоциировалась с таблицей COURSE через правила внешнего ключа. Ответственность за обеспечение целостности данных лежит на стороне "многие", а не на стороне "один". СУБД поддерживает ограничения целостности данных, представленные в бизнес-правилах. (Некоторые реализации СУБД, например, MS SQL Server и Oracle, позволяют определять бизнес-правила на уровне СУБД.)

3.3.7. Сила связи и слабые сущности

В разд. 3.3.5 данной главы мы ввели понятия зависимости существования, а также слабых (неидентифицируемых) и сильных (идентифицируемых) связей. Было показано, что сильные связи определяют изображение связанных сущностей. В терминах проектирования БД существование сильной связи между порождающей сущностью и связанной с ней сущностью (или сущностями) ассоциируется со слабыми сущностями. Поскольку слабые сущности играют важную роль при реализации проекта БД, рассмотрим их подробнее.

Слабой сущностью (weak entity) называется сущность, которая удовлетворяет двум условиям:

- условию зависимости от существования, т. е. она не может существовать без сущности, с которой она связана;
- ее первичный ключ (PK) частично или целиком произведен из порождающей сущности данной связи.

Например, страховой полис компании может страховать служащего и его иждивенцев. В бланке страхового полиса служащий (EMPLOYEE) может указать (или не указать) иждивенцев (DEPENDENT), но иждивенец (DEPENDENT) должен быть связан со служащим (EMPLOYEE). Более того, DEPENDENT не может существовать без EMPLOYEE, т. е. вы не можете получить страховой полис в компании XYZ как иждивенец, если вы не являетесь иждивенцем сотрудника компании! Другими словами, DEPENDENT это слабая сущность в связи "у сотрудника (EMPLOYEE) есть иждивенцы (DEPENDENT)".

В модели "птичья лапка" слабые сущности изображаются небольшими сегментами в каждом из четырех углов прямоугольника сущности CLASS. В модели Чена слабые сущности отображаются с помощью двойной окантовки прямоугольника, который обозначает сущность (рис. 3.19).

Слабая сущность наследует все части первичного ключа своего сильного партнера по связи. Например, по крайней мере часть ключа сущности DEPENDENT будет наследоваться от сущности EMPLOYEE:

EMPLOYEE (EMP_NUM, EMP_LNAME, EMP_FNAME, EMP_INITIAL,
EMP_DOB)

DEPENDENT (EMP_NUM, DEP_NUM, DEP_FNAME, DEP_DOB)

На рис. 3.20 проиллюстрирована связь между слабой сущностью (DEPENDENT) и порождающей ее сущностью или сильной стороной связи (EMPLOYEE). Обратите внимание, что первичный ключ сущности DEPENDENT состоит из двух атрибутов:

EMP_NUM и DEP_NUM, а EMP_NUM наследуется от сущности EMPLOYEE. С помощью этой связи можно определить, что Jeanine J. Callifante имеет трех иждивенцев — Jill, Annelise и Jorge; William K. Smithson не имеет иждивенцев, а Herman H. Washington имеет двух иждивенцев — Robert и Suzanne.

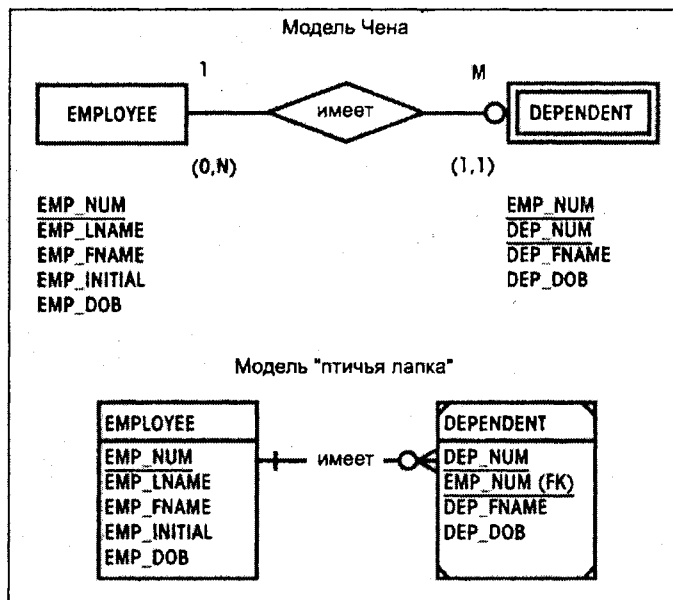


Рис. 3.19. Слабая сущность на ER-диаграммах

База данных: CH3_TEXT					
Таблица: EMPLOYEE_FIG3_20A					
	EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOB
▶	1001	Callifante	Jeanine	J	12-Mar-61
	1002	Smithson	William	K	23-Nov-70
	1003	Washington	Herman	H	15-Aug-68

Таблица: DEPENDENT_FIG3_20B				
	EMP_NUM	DEP_NUM	DEP_FNAME	DEP_DOB
▶	1001	1	Jill	21-Jun-92
	1001	2	Annelise	05-Dec-94
	1001	3	Jorge	30-Sep-98
	1003	1	Robert	17-Nov-93
	1003	2	Suzanne	25-Oct-01

Рис. 3.20. Иллюстрация слабой сущности DEPENDENT в сильной связи между DEPENDENT и EMPLOYEE

Необходимо помнить, что именно проектировщик БД обычно решает, нужно или нет объявлять сущность слабой. Например, при исследовании связи между COURSE и CLASS (см. рис. 3.14) вы можете решить, что сущность CLASS должна быть слабой по отношению к сущности COURSE. Кроме того, если вы посмотрите строки сущности CLASS на рис. 3.14, то станет ясно, что CLASS не может существовать без COURSE, поэтому здесь налицо зависимость от существования. Например, студент не может записаться на Accounting I, группа ACCT-211, раздел 3 (CLASS_CODE 10014), если нет курса ACCT-211. Однако обратите внимание, что первичным ключом таблицы CLASS является атрибут CLASS_CODE, который не получен из порождающей сущности COURSE, т. е. мы можем представить сущность CLASS так:

CLASS(CLASS_CODE, CRS_CODE, CLASS_SECTION, CLASS_TIME,
ROOM_CODE, PROF_NUM)

Второе требование к слабой сущности не выполняется, поэтому сущность CLASS на рис. 3.13 нельзя считать слабой. С другой стороны, если первичный ключ сущности CLASS будет определен как составной ключ, состоящий из CRS_CODE и CLASS_SECTION, то мы можем представить сущность CLASS так:

CLASS (CRS_CODE, CLASS_SECTION, CLASS_TIME, ROOM_CODE,
PROF_NUM)

В этом случае первичный ключ сущности CLASS частично произведен от COURSE, поскольку CRS_CODE является первичным ключом сущности COURSE. При таком положении дел сущность CLASS является слабой сущностью по определению. В любом случае, сущность CLASS всегда зависима от существования COURSE, не взирая на то, определена она как слабая или нет.

3.3.8. Степень связи

Степень связи (relationship degree) указывает на число ассоциированных сущностей или участников (participants). *Унарная связь (unary relationship)* существует тогда, когда ассоциация поддерживается внутри единственной сущности. *Бинарная связь (binary relationship)* существует тогда, когда ассоциируются две сущности. *Тернарная связь (ternary relationship)* имеет место тогда, когда связываются три сущности. Хотя существуют и более высокие степени связи, они довольно редки и не имеют особых названий. (Например, ассоциация четырех сущностей описывается просто как *связь четырех сущностей*). На рис. 3.21 представлены связи различной степени.

Если сравнить диаграммы Чена и "птичья лапка" (рис. 3.21), то можно заметить, что модель "птичья лапка", нацеленная на реализацию, требует перевода *связи* CFR (Contributor-Fund-Recipient, спонсор-фонд-получатель) в *сущность* CFR, через которую будет реализована связь. На концептуально ориентированной модели Чена сразу видно, что CFR на самом деле представляет собой тернарную *связь*.

В случае унарной связи, представленной на рис. 3.21, курс внутри сущности COURSE является предварительным условием для существования другого курса внутри данной сущности. В данном случае наличие предвещающего курса означает, что сущность COURSE требует наличия сущности COURSE — т. е. сущность COURSE имеет связь сама с собой. Такую связь называют также *рекурсивной связью*.

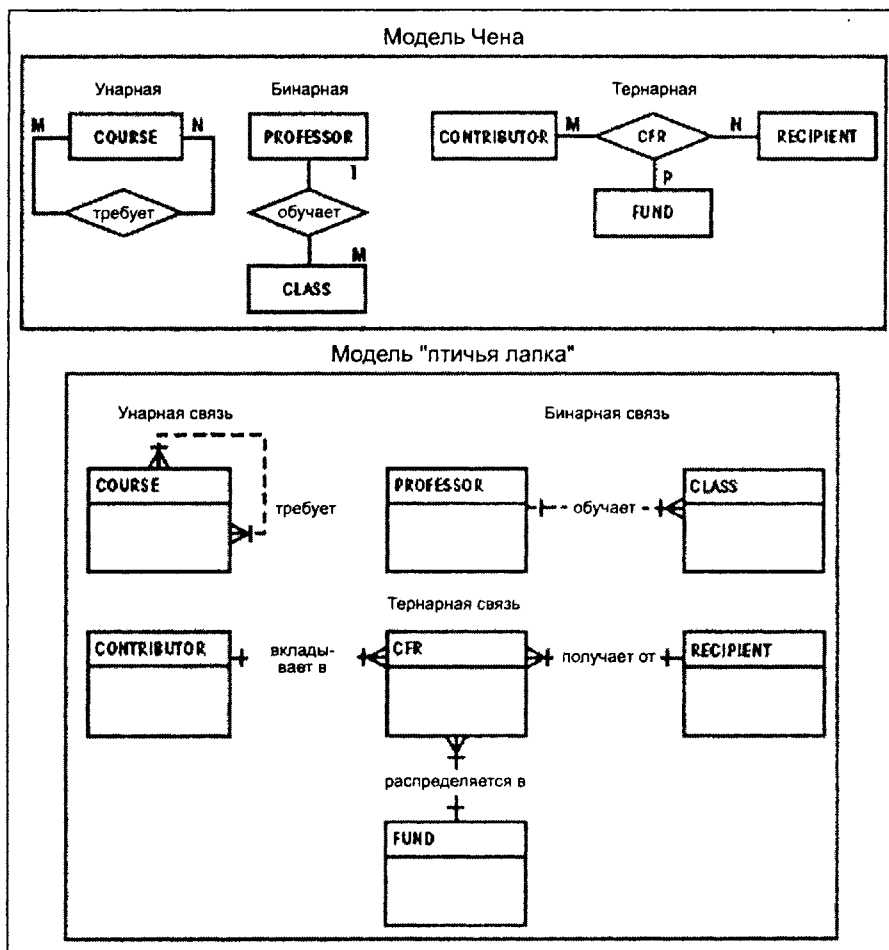


Рис. 3.21. Три типа связей

Примечание

Бинарные связи имеют наибольшее распространение. На самом деле, для упрощения концептуального проектирования большинство связей высокой степени (тернарные и выше) по возможности разбиваются на соответствующие эквиваленты бинарных связей.

Хотя большинство связей являются бинарными, использование тернарных (и более высокого порядка) связей дает проектировщику некоторую свободу в отношении семантики задачи. Например, обратите внимание на связи (и их результат!), представленные на рис. 3.22.

- Люди или учреждения, входящие в сущность CONTRIBUTOR (спонсоры), жертвуют средства в специальный исследовательский фонд (FUND). (Исследова-

тельные фонды могут курировать несколько направлений. Например, если фонд создан для поддержки медицинских исследований, то в нем может существовать раковое направление, направление борьбы со СПИДом, направление сердечно-сосудистых заболеваний и т. д.)

- Исследователи, входящие в сущность RECIPIENT (получатели), финансируются из фонда (FUND).
- Все связи здесь можно отнести к типу M:N. Например, спонсоры могут делать вклады в несколько фондов. Фонд может иметь множество спонсоров. Фонд может поддерживать множество исследователей, которые являются получателями средств (RECEPIENT), а исследователи могут получать помощь от нескольких фондов.
- Три сущности (CONTRIBUTOR, RECIPIENT и FUND) связаны тернарной связью, которую мы обозначили CFR. Поскольку тернарную связь невозможно выразить одним глаголом, мы используем акроним CFR для обозначения связи между сущностями CONTRIBUTOR, FUND и RECIPIENT. Таким образом, сущность CFR позволяет конечному пользователю БД сопоставить спонсоров фонда с получателями фонда, как показано на рис. 3.22.

Таблица: CONTRIBUTOR			Таблица: FUND			
CON_ID	CON_NAME		FUND_ID	FUND_NAME	CON_ID	FUND_AMOUNT
C1	Brown		F1	Heart	C1	\$50,000.00
C2	Iglesas		F1	Heart	C2	\$10,000.00
C3	Smith		F2	Cancer	C1	\$10,000.00
			F2	Cancer	C2	\$5,000.00
			F2	Cancer	C3	\$10,000.00

Таблица: RECIPIENT			Таблица: CFR			
REC_ID	REC_TYPE		FUND_ID	CON_ID	CFR_AMOUNT	REC_ID
R1	Cancer		F1	C1	\$30,000.00	R2
R2	Heart		F1	C1	\$20,000.00	R3
R3	Heart		F1	C2	\$10,000.00	R2
			F2	C1	\$10,000.00	R1
			F2	C2	\$5,000.00	R1

Рис. 3.22. Реализация тернарной связи

При изучении содержимого таблиц на рис. 3.22 обратите внимание, что здесь имеется возможность проследить все транзакции. Например, исследователь R1, специализирующийся на раковых заболеваниях, получил в общей сложности \$15 000 (\$10 000 от спонсора C1 — Brown и \$5000 от спонсора C2 — Iglesias). Исследователь R2, специализирующийся на сердечно-сосудистых заболеваниях, получил в общей сложности \$40 000 (\$30 000 от спонсора C1 и \$10 000 от спонсора C2). Кроме того, тернарная связь CFR позволяет нам связать доступные фонды со спонсорами. Например,

содержимое таблицы показывает, что спонсор C1 сделал вклад \$50 000 на исследования в области сердечно-сосудистых заболеваний, откуда \$30 000 были получены исследователем R2, в то время как оставшиеся \$20 000 были получены исследователем R3. Заметим, что в модели "птичья лапка" на рис. 3.21 уже было показано, что тернарная связь должна быть реализована как сущность, что упрощает отслеживание транзакций, которые мы только что обсудили.

Тернарная связь, описанная здесь, оказывает влияние на тип и размер информации, доступной в запросах к БД. Например, если 500 человек жертвуют на направление фонда в области сердечно-сосудистых заболеваний, а мы не будем использовать тернарную связь, то не будет способа идентифицировать конкретный источник таких фондов, когда получателю будут предоставлены соответствующие денежные средства. Поэтому если есть необходимость отслеживать отдельных спонсоров и получателей фонда, то придется применять тернарные связи. Как следует из предыдущего примера, тернарная связь не всегда эквивалентна нескольким связям типа 1:M. Следовательно, проектировщик должен принимать во внимание семантику поставленной задачи: упрощения не всегда отвечают потребностям пользователя!

Рекурсивные связи

Рекурсивная связь (recursive relationship) имеет место, когда есть связь между экземплярами одного и того же набора сущностей. (Естественно, такое условие соблюдается в унарных связях.)

Например, унарная связь 1:M может быть представлена как "сотрудник (EMPLOYEE) руководит несколькими сотрудниками (EMPLOYEE), а каждый сотрудник подчинен только одному сотруднику (EMPLOYEE)". Также, поскольку полигамия незаконна, унарную связь типа 1:1 представляет и следующее высказывание: "сотрудник (EMPLOYEE) может состоять в браке с одним и только одним сотрудником (EMPLOYEE)". Наконец, унарную связь M:N можно выразить как "курс (COURSE) может предшествовать многим другим курсам (COURSE), а каждый курс (COURSE) может иметь в качестве предшествующих множество других курсов (COURSE)". Эти связи представлены на рис. 3.23.

Связь 1:1, представленная на рис. 3.23, может быть реализована единственной таблицей (рис. 3.24). Обратите внимание: мы можем определить, что James Ramirez женат на Louise Ramirez, которая, в свою очередь, замужем за James Ramirez. А Anne Jones замужем за Anton Shapiro, который, в свою очередь, женат на Anne Jones.

Унарные связи часто имеют место в промышленности. Например, рис. 3.25 иллюстрирует, что пропеллер (вентилятора, например) в сборе (C-130) состоит из множества деталей, но каждая деталь используется для создания только одной сборки пропеллера. На рис. 3.25 показано, что сборка пропеллера состоит из четырех шайб (2,5 см), двух шплинтов, одного хвостовика (2,5 см), четырех лопастей (10,25 см) и двух шестигранных гаек (2,5 см). Связь, реализованная на рис. 3.25, позволяет нам проследить каждую деталь внутри каждой выпускаемой сборки пропеллера.

Если деталь используется для различных сборок и сама состоит из нескольких деталей, необходимы две таблицы для реализации связи "деталь (PART) входит в деталь (PART)". На рис. 3.26 демонстрируется такая конфигурация. Сегодня отслеживание деталей становится все более важным, поскольку менеджеры осознают, что современное производство сильно разветвлено, а продукция значительно усложнилась.

На самом деле, в промышленности, особенно связанной с авиацией, полное отслеживание деталей является жизненной необходимостью.

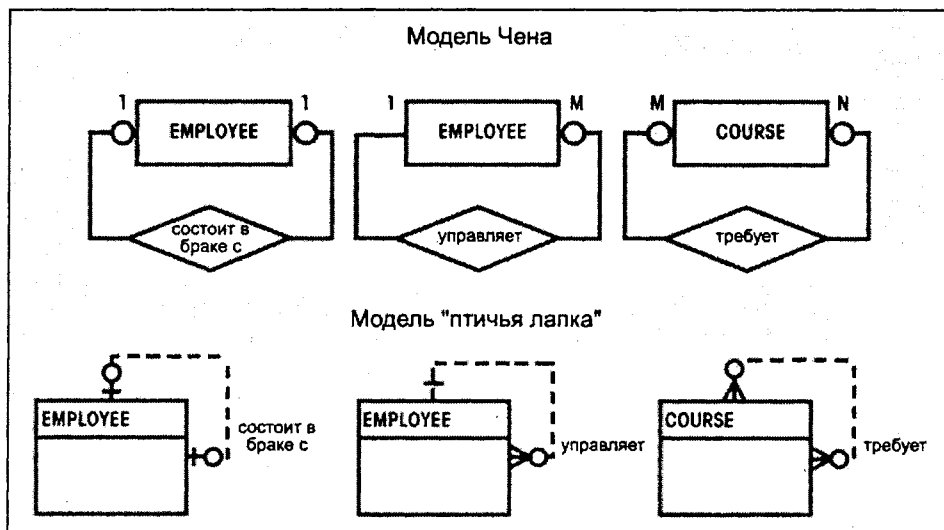


Рис. 3.23. ER-представление рекурсивных связей

Таблица: EMPLOYEE_FIG3_24		База данных: CH3_TEXT		
	EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_SPOUSE
▶	345	Ramirez	James	347
▶	346	Jones	Anne	349
▶	347	Ramirez	Louise	345
▶	348	Delaney	Robert	
▶	349	Shapiro	Anton	346

Рис. 3.24. Рекурсивная связь 1:1 "сотрудник (EMPLOYEE) состоит в браке с сотрудником (EMPLOYEE)"

Таблица: PART_FIG3_25		База данных: CH3_TEXT			
	PART_CODE	PART_DESCRIPTION	PART_IN_STOCK	PART_UNITS_NEEDED	PART OF PART
▶	AA21-E	2.5 cm. washer, 1.0 mm. rim	432	4 C-130	
▶	AB-121	Cotter pin, copper	1034	2 C-130	
▶	C-130	Rotor assembly	36		
▶	E129	2.5 cm. steel shank	128	1 C-130	
▶	X10	10.25 cm. rotor blade	345	4 C-130	
▶	X34AW	2.5 cm. hex nut	879	2 C-130	

Рис. 3.25. Другая унарная связь: "деталь (PART) входит в деталь (PART)"

Таблица: COMPONENT		База данных: CH3_TEXT	
COMP_CODE	PART_CODE	COMP_PARTS_NEEDED	
▶ C-130	E129	1	
C-130	X10	4	
C-130	AA21-6	4	
C-130	AB-121	2	
C-130	X34AW	2	
C-131A2	E129	1	
C-131A2	X10	1	
C-131A2	X34AW	2	

Таблица: PART		
PART_CODE	PART_DESCRIPTION	PART_IN_STOCK
▶ AA21-6	2.5 cm. washer, 1.0 mm. rim	432
AB-121	Cotter pin, copper	1034
C-130	Rotor assembly	36
E129	2.5 cm. steel shank	128
X10	10.25 cm. rotor blade	345
X34AW	2.5 cm. hex nut	879

Рис. 3.26. Реализация рекурсивной связи M:N
"деталь (PART) входит в деталь (PART)"

Таблица: COURSE		База данных: CH3_TEXT	
CRS_CODE	DEPT_CODE	CRS_DESCRIPTION	CRS_CREDIT
▶ ACCT-211	ACCT	Accounting I	3
ACCT-212	ACCT	Accounting II	3
CIS-220	CIS	Intro. to Microcomputing	3
CIS-420	CIS	Database Design and Implementation	4
MATH-243	MATH	Mathematics for Managers	3
QM-261	CIS	Intro. to Statistics	3
QM-362	CIS	Statistical Applications	4

Таблица: PREREQ		
CRS_CODE	PRE TAKE	
▶ QM-261	MATH-243	
QM-362	MATH-243	
QM-362	QM-261	
CIS-420	CIS-220	

Рис. 3.27. Реализация рекурсивной связи M:N
"курс (COURSE) требует знания курса (COURSE)"

Рекурсивные связи M:N нам более знакомы по колледжу. Например, обратите внимание, как связь M:N "курс (COURSE)" требует знания курса (COURSE)", представленная на рис. 3.23, реализована на рис. 3.27. В этом примере курс MATH-243 предлагает курсы QM-261 и QM-362, в то время как QM-362 требует предварительного прохождения курса MATH-243 и QM-261.

Наконец, рекурсивная связь 1:M "сотрудник (EMPLOYEE) руководит сотрудником (EMPLOYEE)", показанная на рис. 3.23, реализована на рис. 3.28.

Таблица: EMPLOYEE_FIG3_28		База данных: CH3_TEXT	
	EMP-CODE	EMP_LNAME	EMP_MANAGER
▶	01	Waddell	102
	102	Orincona	
	103	Jones	102
	104	Rebolloh	102
	105	Robertson	102
	106	Deltona	102

Рис. 3.28. Реализация рекурсивной связи 1:M
"сотрудник ((EMPLOYEE) руководит сотрудником (EMPLOYEE))"

3.3.9. Составные сущности

В оригинальной ER-модели, описанной Ченом, связи не содержат атрибутов. Вы должны помнить из гл. 2, что реляционная модель требует использования связей 1:M. Если встречается связь типа M:N, то нам придется создать переход между сущностями, соединенными такой связью. Такая переходная *промежуточная сущность (bridge entity)* состоит из первичных ключей каждой из соединяемых сущностей. (Пример такого перехода приведен в гл. 2, на рис. 2.25 и 2.26 и с некоторыми изменениями он воспроизведен на рис. 3.29). Промежуточную сущность называют также *составной сущностью (composite entity)*. В гл. 2 было показано, что составная сущность в модели Чена изображается ромбом в прямоугольнике. Поскольку модель "птичья лапка" ориентирована на реализацию, в ней составная сущность не выделяется.

Если внимательно посмотреть на рис. 3.29, то можно заметить, что составная сущность ENROLL зависит от существования двух других сущностей; в ее основе — первичные ключи сущностей, соединяемые с помощью составной сущности. Составная сущность может также содержать дополнительные атрибуты, не играющие существенной роли в связи. Например, хотя в составную сущность должны, по крайней мере, включаться первичные ключи сущностей STUDENT и CLASS, в нее также могут входить такие дополнительные атрибуты, как оценки, пропуски занятий и другие данные, уникально идентифицирующие успеваемость студента конкретной группы.

Наконец, необходимо помнить, что ключ таблицы ENROLL (CLASS_CODE и STU_NUM) целиком составлен из первичных ключей таблиц CLASS и STUDENT.

Поэтому в ключевых атрибутах таблицы ENROLL недопустимы пустые значения (null).

Таблица: STUDENT_FIG3_29A

STU_NUM	STU_LNAME
321452	Bowser
324257	Smithson

База данных: CH3_TEXT

Таблица: ENROLL

CLASS_CODE	STU_NUM	ENROLL_GRADE
10014	321452	C
10014	324257	B
10018	321452	A
10018	324257	B
10021	321452	C
10021	324257	C

Таблица: PARTCLASS_FIG3_29C

CLASS_CODE	CRS_CODE	CLASS_SECTION	CLASS_TIME	CLASS_ROOM	PROF_NUM
10014	ACCT-211	3	TTh 2:30-3:45 p.m.	BUS252	342
10018	CIS-220	2	MWTF 9:00-9:50 a.m.	KLR211	114
10021	QM-261	1	MWTF 8:00-8:50 a.m.	KLR200	114

Рис. 3.29. Конвертирование связи M:N в две связи 1:M

Реализация небольшой базы данных, представленной на рис. 3.29, требует, чтобы мы четко определили связи. В частности, мы должны знать стороны "1" и "М" каждой связи, а также то, являются ли связи необязательными или обязательными. Например, стоит обратить внимание на следующее:

- ☐ группа может существовать (по крайней мере, в начале регистрации), даже если в ней нет ни одного студента. Поэтому (рис. 3.30) в связи типа M:N между STUDENT и CLASS со стороны сущности STUDENT нужно использовать символ необязательности.

Однако поскольку связь между сущностями STUDENT и CLASS разбивается на две связи 1:M при помощи промежуточной сущности ENROLL, то необязательность переходит в сущность ENROLL (рис. 3.31). Иначе говоря, теперь возможна ситуация, когда группа не входит в ENROLL, если ни один студент в эту группу не записан. Поскольку группа может и не встретиться в ENROLL, сущность ENROLL становится необязательной для сущности CLASS;

- ☐ если некто стал студентом (STUDENT), то он должен быть записан, по крайней мере, в одну группу (CLASS). Говоря кратко, сущность CLASS обязательна для сущности STUDENT. Следовательно, студент должен появиться, по крайней мере, один раз в сущности ENROLL. Естественно, если студент записывается в несколько групп (берет несколько курсов), то он может появиться более одного раза внутри сущности ENROLL. Например, обратите внимание, что STU_NUM = 321452

встречается в таблице ENROLL три раза (см. рис. 3.29). С другой стороны, каждый студент встречается в таблице STUDENT только один раз. (Например, в таблице STUDENT на рис. 3.29 $STU_NUM = 321452$ встречается только один раз). Поэтому между STUDENT и ENROLL на рис. 3.31 имеет место связь типа 1:M, причем "M" относится к ENROLL;

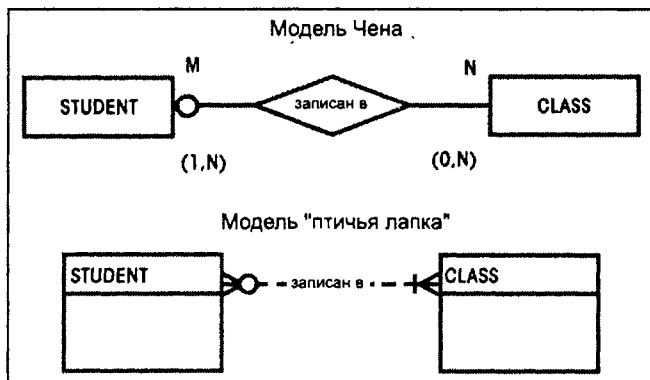


Рис. 3.30. Связь M:N между STUDENT и CLASS

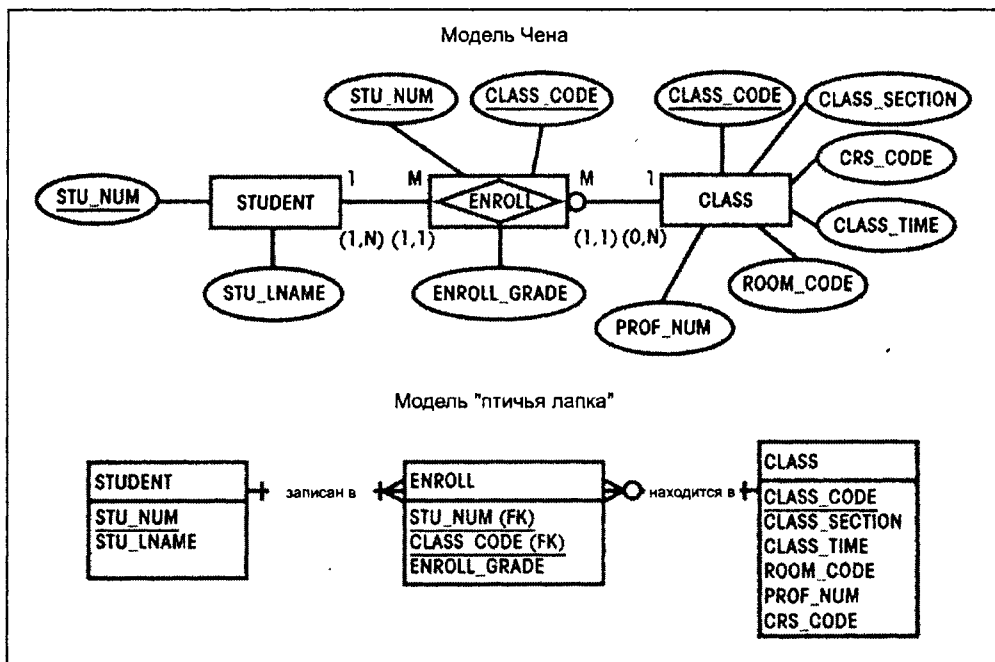


Рис. 3.31. Составная сущность на ER-диаграмме

- из рис. 3.29 следует, что группа может встречаться в таблице ENROLL более одного раза. Например, CLASS_CODE = 10014 дважды встречается в таблице ENROLL. Однако CLASS_CODE = 10014 встречается в таблице CLASS только один раз, поэтому между CLASS и ENROLL имеет место связь 1:M. На рис. 3.31 обозначение "M" размещено на стороне ENROLL, в то время как "1" — на стороне CLASS.

Примечание

На ER-диаграмме на рис. 3.31 представлена вся информация, необходимая для моделирования упрощенного процесса записи студента в группу. Однако включение всех атрибутов вносит некоторую неразбериху в диаграмму, особенно когда ER-модель содержит множество сущностей, и каждая из этих сущностей содержит множество атрибутов. Поэтому, хотя ER-форма, представленная на рис. 3.31, полезна с точки зрения обучения, в действительности проектировщики БД главным образом используют ER-модели, на которых не изображаются атрибуты в виде эллипсов. В последующем мы будем использовать именно такой практический подход.

3.3.10. Супертипы и подтипы сущности

Большинство предприятий принимают на работу людей с различными профессиональными навыками. Например, авиационное предприятие может нанимать пилотов, а также множество служащих других специальностей. Описывать специализацию всех служащих предприятия в одной таблице очень неудобно. С одной стороны, пилоты должны отвечать специальным требованиям (например, минимальный налет, тренировочные полеты и т. д.), которые не нужны для других сотрудников. Поэтому, если сведения обо всех сотрудниках и их квалификации будут описываться единственной таблицей EMPLOYEE и храниться так, как это показано на рис. 3.32, вы или получите много пустых мест в такой таблице, или будете вынуждены вводить большое количество ненужных фиктивных элементов. Например, если сотрудники Lewis, Lange, Williams, Duzak и Travis являются пилотами, то, как показано на рис. 3.32, наличие у них специальных квалификационных характеристик EMP_LICENSE, EMP_RATINGS и EMP_MED_TYPE приведет к появлению большого количества пустых мест (null) при занесении сведений о других сотрудниках, которые не являются пилотами³.

Очевидно, что у пилотов есть некоторые общие атрибуты с другими сотрудниками, например, фамилия (EMP_LNAME) и дата рождения (EMP_HIRE_DATE). С другой стороны, имеется много характеристик, создающих проблемы при попытке хранения атрибутов всех сотрудников в одной таблице. К счастью, эти проблемы легко разрешаются с помощью иерархии обобщенных представлений о наборах сущностей, использующих совместные характеристики.

В основном *иерархия обобщенных представлений (generalization hierarchy)* отображает связи "предок-потомок", о которых мы говорили при обсуждении иерархической

³ Пилот должен иметь периодически обновляемый медицинский сертификат (EMP_MED_TYPE), допускающий его к полетам. Например, медицинский сертификат Type I требуется для выдачи лицензии на пилотирование на транспортных авиалиниях, Type II — на коммерческих авиалиниях, а TYPE III — для частных лицензий.

модели баз данных в гл. 1. В контексте реляционных баз данных иерархия обобщенных представлений отображает связи между супертипами сущности верхнего уровня и подтипами сущности нижнего уровня (рис. 3.33). Другими словами, *супертип* содержит совместно используемые атрибуты, в то время как *подтип* содержит уникальные атрибуты.

Таблица: EMPLOYEE_FIG3_22						База данных: CH3_TEXT	
	EMP_NUM	EMP_LNAME	EMP_LICENSE	EMP_RATINGS	EMP_MED_TYPE	EMP_HIRE_DATE	
▶	100	Kolmycz				15-Mar-85	
	101	Lewis	ATP	SEL/MEL/Instr/CFII	1	25-Apr-86	
	102	Vandam				20-Dec-90	
	103	Jones				28-Aug-00	
	104	Lange	ATP	SEL/MEL/Instr	1	20-Oct-94	
	105	Williams	COM	SEL/MEL/Instr/CFI	2	08-Nov-94	
	106	Duzak	COM	SEL/MEL/Instr	2	05-Jan-02	
	107	Diante				02-Jul-94	
	108	Viesenbach				18-Nov-92	
	109	Travis	COM	SEL/MEL/SES/Instr/CFII	1	14-Apr-98	
	110	Genkazi				01-Dec-01	

Рис. 3.32. Пустые места (null), созданные уникальными атрибутами

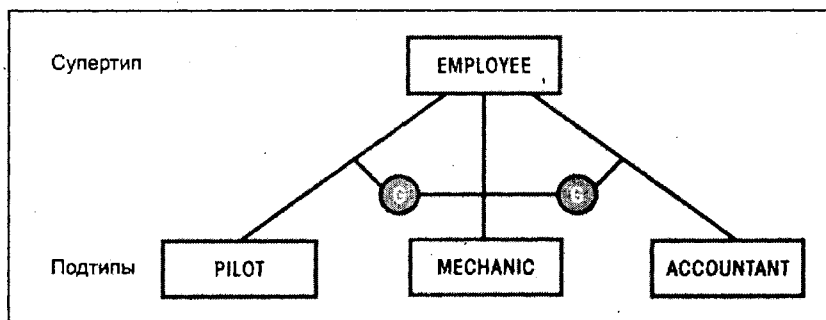


Рис. 3.33. Иерархия обобщенных представлений

Примечание

Вы можете сказать, что пустые места в таблице EMPLOYEE на рис. 3.32 можно удалить с помощью флагов, но такое решение трудно осуществимо, когда таблица расширяется путем включения численных значений, связанных с пилотами, например, общий налет в часах, налет в часах за текущий год и т. д. Если вы используете флаг -999 или 0 (или другой числовой флаг) для "не пилотов", то расчетные значения, такие как общий средний налет, среднегодовой минимальный и максимальный налеты теряют свой смысл. Возможное решение этой проблемы состоит в том, чтобы создать атрибут классификации специальности, а затем выполнить все расчеты с учетом ограничения по этому атрибуту. Такое решение позволит корректно рассчитать численные значения, но также может привести к более сложным форматам запросов, быстрому росту числа атрибутов с

классификацией специальности и возможному увеличению квалификационных атрибутов различных типов. В конечном счете проектировщик должен использовать при выборе подхода профессиональное чутье. Мы признаем, что супертипы и подтипы оказались весьма полезными при работе над проектами.

Из рис. 3.33 можно видеть, что связи наследуются, т. е. подтип сущности наследует атрибуты и связи от супертипа сущности. Например, все пилоты, механики и бухгалтеров имеют имена, домашние адреса и т. д. Однако ранее было показано, что сотрудники могут также иметь атрибуты, уникальные для их специализации. Другими словами, супертип набора сущностей обычно связан с несколькими уникальными и непересекающимися (*disjoint, nonoverlapping*) подтипами набора сущностей. Такие *непересекающиеся* связи обозначаются символом "G".

Супертип и его подтип(ы) поддерживают связь 1:1. Например, иерархия обобщенных представлений позволяет заменить нежелательную структуру таблицы EMPLOYEE (см. рис. 3.32) двумя таблицами, одна из которых представляет супертип EMPLOYEE, а другая — подтип PILOT (рис. 3.34).

Таблица: EMPLOYEE (супертип)			База данных: CH3_TEXT		
EMP_NUM	EMP_LNAME	EMP_HIRE_DATE			
100	Kolmycz	15-Mar-85			
101	Lewis	25-Apr-86			
102	Vandam	20-Dec-90			
103	Jones	28-Aug-00			
104	Lange	20-Oct-94			
105	Williams	08-Nov-94			
106	Duzak	05-Jan-02			
107	Diante	02-Jul-94			
108	Wiesenbach	18-Nov-92			
109	Travis	14-Apr-98			
110	Genkazi	01-Dec-01			

Таблица: PILOT (подтип)			
EMP_NUM	PIE_LICENSE	PIE_RATINGS	PIE_MED_TYPE
101	ATP	SEL/MEL/Instr/CFII	1
104	ATP	SEL/MEL/Instr	1
105	COM	SEL/MEL/Instr/CFI	2
106	COM	SEL/MEL/Instr	2
109	COM	SEL/MEL/SES/Instr/CFII	1

Рис. 3.34. Связь EMPLOYEE/PILOT супертип/подтип

Некоторые супертипы содержат *пересекающиеся (overlapping)* подтипы. Например, какой-то сотрудник может быть преподавателем и, к тому же, администратором, поскольку руководитель факультета может также вести занятия. Точно так же человек может быть и студентом, и сотрудником, поэтому сущности STUDENT и EMPLOYEE являются подтипами сущности PERSON (субъект) в пересекающейся

связи, как сущности PROFESSOR и ADMINISTRATOR являются подтипами сущности EMPLOYEE (сотрудник) в пересекающейся связи. Такие пересекающиеся связи на рис. 3.35 обозначены символами "Gs".

Примечание

Символы G и Gs для супертипов/подтипов предложил Тоби Дж. Теори (Toby J. Teorey) в своей работе "Database Modeling and Design: The Entity Relationship Approach", Morgan Kaufmann, 1990.

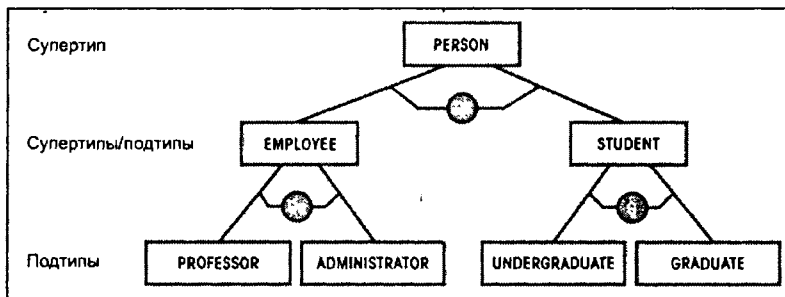


Рис. 3.35. Иерархия обобщенных представлений с пересекающимися подтипами

В сущности, иерархия обобщенных представлений представляет собой связь "является кем-то". Например, "преподаватель является сотрудником", "администратор является сотрудником" и "сотрудник является субъектом".

3.4. Сравнение обозначений в ER-моделировании

Обозначения связей и мощности, которые мы использовали до настоящего времени, типичны для моделей Чена и "птичья лапка". Однако необходимо ознакомиться и с другими ER-стилями, например, Rein85 и IDEF1X. Хотя эти модели основаны на одних и тех же концепциях моделирования, в них были разработаны дополнительные стили, которые проще приспосабливаются к компьютерным инструментальным средствам моделирования, чем оригинальная модель Чена. Весьма вероятно, что вы будете конвертировать даже лучшие модели Чена в эти (или подобные) модели, если собираетесь использовать средства компьютеризированного проектирования систем (*computer-assisted systems engineering, CASE*) в контексте баз данных. В следующем списке суммируются основные свойства каждой модели.

- Модель Чена предложена Питером Ченом (Peter Chen) в основополагающей работе "The Entity-Relationship Model. Toward a Unified View of Data" (ACM, Transactions on Database Systems 1(1), March, 1976). Модель Чена переводит концептуальное моделирование в практическую плоскость проектирования БД, предлагая в качестве основных функциональных блоков сущности и связи. Идеи Чена были развиты Т. Дж. Теори, Д. Янгом и Дж. П. Фраем (Т. J. Teory, D. Yang и J. P. Fry) в опубликованной ими работе "A Logical Design Methodology for Relational Databases Using

the Extended Entity Relationship Model" (ACM Computing Surveys, June, 1986, стр. 197—222). Основные структуры в модели Чена, расширенные в работе Теори, Янга и Фрая, доминировали на рынке средств CASE в конце 1980-х начале 1990-х годов. (См. работу Майка Риччити (Mike Ricciuti) "Database Vendors Make Their CASE", Datamation 38(5), March 1992.) Инструментальное CASE-средство Excelerator, которое использовалось многими проектировщиками БД в начале 1990-х годов, возможно, была наилучшей "чистой" моделью Чена. Хотя модель Чена в настоящее время и не лидирует на рынке генераторов ER-диаграмм, все современные средства своими корнями уходят в модель Чена.

- Модель "птичья лапка" (Crow's Foot model), разработанная К. В. Бахманом (C. W. Bachman), стала популярным инструментом моделирования. Вместо использования символического стиля Чена при обозначении связей (1 и M) и мощности (0,1), (0,N), (1,1) и (1,N), в этой модели используются графические обозначения, показанные на рис. 3.36. (Название "птичья лапка" как раз и отражает графический символ, используемый для обозначения связи M в виде трехпальцевой птичьей лапки). В модели "птичья лапка" совмещается информация о связности и мощности связи в едином наборе символов. В отличие от модели Чена, эта модель позволяет обозначить лишь три значения мощности 0, 1 или N. Например, в модели "птичья лапка" невозможно отобразить мощность (5,25).
- Модель Rein85 разработана Д. Ренером (D. Reiner) в 1985 году. Хотя она основана на тех же концепциях, что и модель "птичья лапка", система обозначений в модели Rein85 другая, она оперирует на более высоком уровне абстракции. В модели Rein85 невозможно явно указать мощность, для ее обозначения используются связи. Система обозначений Rein85 представлена на рис. 3.36.
- Модель IDEF1X стала результатом исследований интегрированных систем автоматизированного производства (integrated computer-aided manufacturing, ICAM), которые проводились в 1970-х годах. ICAM стала источником графических методов определения функций, структур данных и динамики развития промышленных предприятий. Интеграция этих методов получила название IDEF (ICAM Definition, методы ICAM). Оригинальная версия IDEF, разработанная компанией Hughes Aircraft, получила название IDEF1. Расширенная версия этой модели, которая называется IDEF1X, стала стандартом в промышленности военно-воздушных сил США и получила признание как основное промышленное инструментальное средство моделирования данных. Обратите внимание на рис. 3.36, что в IDEF1X используется меньше символов, чем в других моделях данных, — это уменьшает детализацию типов и расширяет возможности изображения связей.

Чтобы проиллюстрировать использование этих четырех методов, рассмотрим модель выписки счетов, которую мы обсуждали в гл. 2. Внимательно исследуйте содержимое таблиц (см. рис. 2.29) и схемы их связей (см. рис. 2.30). Мы будем к тому же следовать следующим бизнес-правилам:

- в моделях учета счетов мы полагаем, что список CUSTOMER (клиент) содержит и потенциальных, и действующих клиентов. Следовательно, клиент в этом списке мог (еще) и не сделать ни одной покупки, что инициировало бы выписку счета. Иначе говоря, сущность INVOICE необязательна для сущности CUSTOMER;
- некоторые товары, хранящиеся в инвентарном списке, никогда не продавались и поэтому никогда не присутствовали в счетах. Следовательно, сущность INVOICE необязательна для сущности PRODUCT (товар). Однако помните, что связь M:N между INVOICE и PRODUCT реализуется через промежуточную сущность LINE

для того, чтобы разбить связь M:N на две связи 1:M. Таким образом, сущность LINE (строка) необязательна для сущности PRODUCT, поскольку непроданный товар никогда не появляется в строке счета.

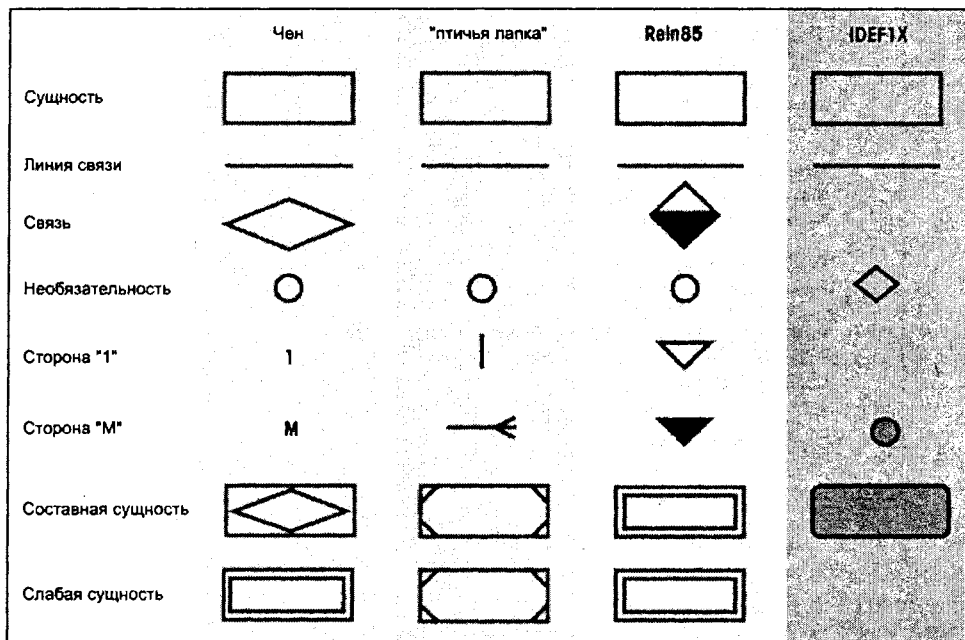


Рис. 3.36. Сравнение обозначений в ER-моделях

На рис. 3.37—3.40 представлены все четыре модели данных, в которых использована информация из гл. 2 (см. разд. 2.7) и приведенные бизнес-правила.

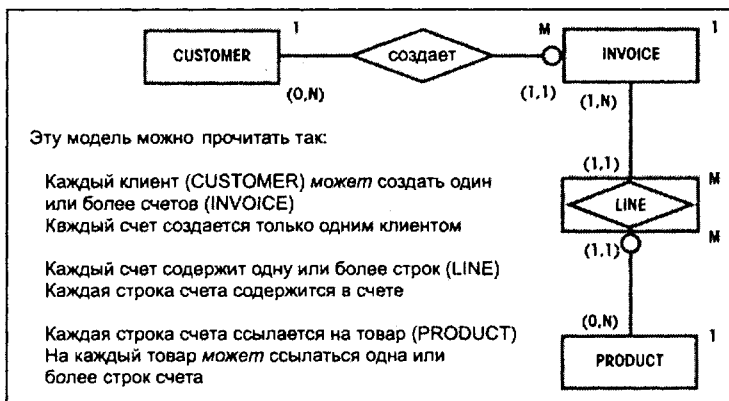


Рис. 3.37. Представление системы учета счетов с помощью модели Чена

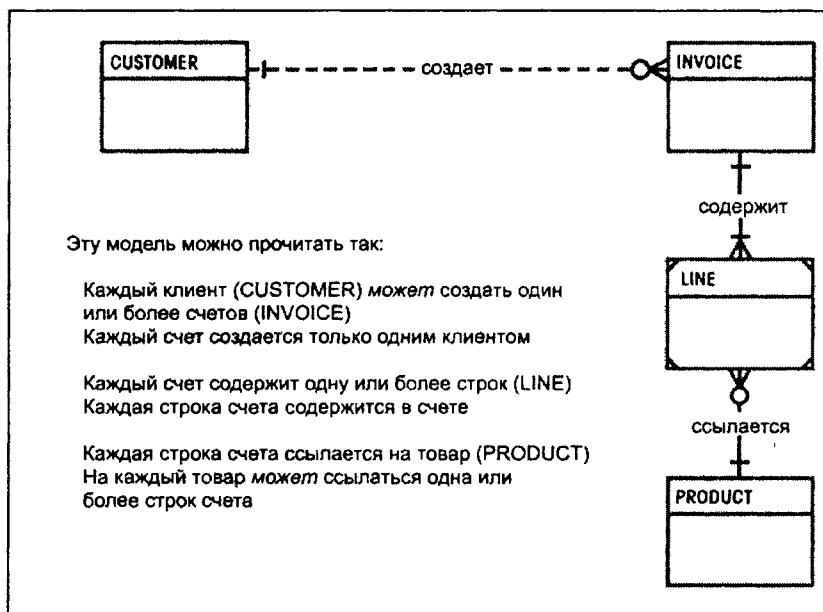


Рис. 3.38. Представление системы учета счетов с помощью модели "птичья лапка"

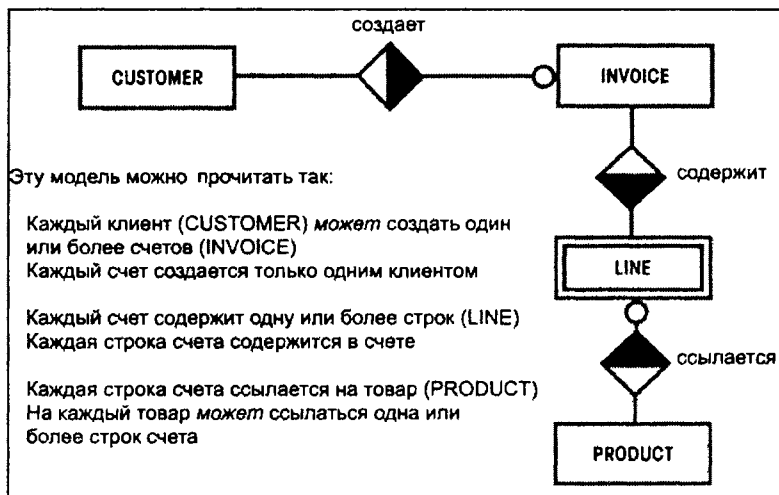


Рис. 3.39. Представление системы учета счетов с помощью модели Rein85

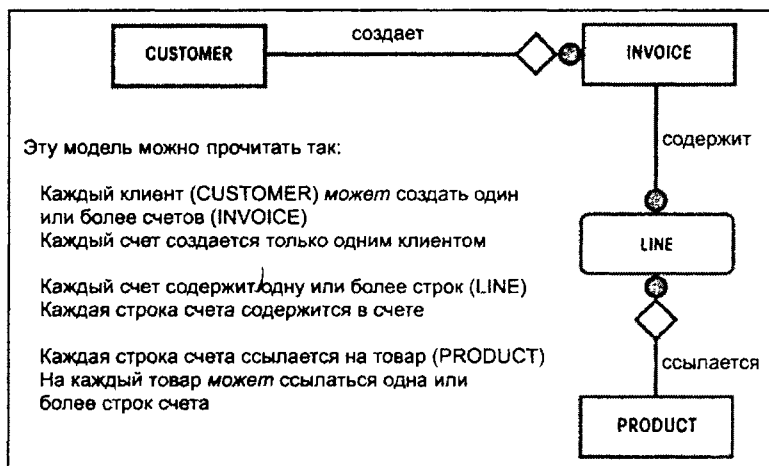


Рис. 3.40. Представление системы учета счетов с помощью модели IDEF1X

3.5. Разработка ER-диаграмм

Процесс проектирования базы данных является итеративным, а не линейным или последовательным. Термин "итеративный" означает "повторяющийся". *Итеративный процесс (iterative process)* основан, таким образом, на повторяющихся операциях и процедурах. Например, построение ER-диаграмм обычно начинается с общего описания операций и процедур предприятия. Затем создается графическое представление базовой ER-модели. При исследовании ER-модели, вероятнее всего, появятся дополнительные объекты, атрибуты и связи. Поэтому базовая ER-модель после включения вновь обнаруженных компонентов будет изменяться. Впоследствии новый цикл исследования может также привести к появлению дополнительных компонентов и видоизменению имеющейся диаграммы. Процесс повторяется до тех пор, пока конечные пользователи и проектировщики не посчитают, что разработанная ER-диаграмма правильно отражает деятельность предприятия и его функции.

В процессе проектирования проектировщики базы данных зависят не только от опросов, помогающих определить сущности, атрибуты и связи. Значительное количество информации может быть получено при исследовании бизнес-форм и отчетов, используемых на предприятии в его повседневной деятельности.

Чтобы проиллюстрировать этот итеративный процесс, который, в конечном счете, должен привести к созданию работоспособной ER-диаграммы, предположим, что мы начинаем опрос администрации колледжа Tiny College. В процессе опроса мы получаем общее описание деятельности колледжа.

- Tiny College подразделяется на несколько факультетов: факультет бизнеса, факультет искусства и сцены, факультет обучения и факультет прикладных наук. Каждым факультетом руководит декан. Отметим, что мощность сущности DEAN (декан) может быть представлена как (1,1), а сущности SCHOOL (факультет) —

тоже как (1,1). (Наименьшее и наибольшее количество деканов факультета равно 1, каждый декан назначается только на один факультет.)

Существует, конечно, несколько способов оценки связи между сущностями DEAN и SCHOOL. Каждый декан является членом группы более высокого уровня, которая называется администрацией, поэтому мы можем включить деканов в сущность ADMINISTRATOR. Однако деканы являются еще и преподавателями и могут вести занятия в группах. Следовательно, деканы могут быть включены в сущность PROFESSOR (преподаватели). Кроме того, администраторы и преподаватели являются сотрудниками. Поэтому в зависимости от требований к информации и представлений проектировщика БД об этих требованиях хорошо было бы оценить возможность создания связей супертипов/подтипов, которые представлены в ER-фрагменте на рис. 3.41.

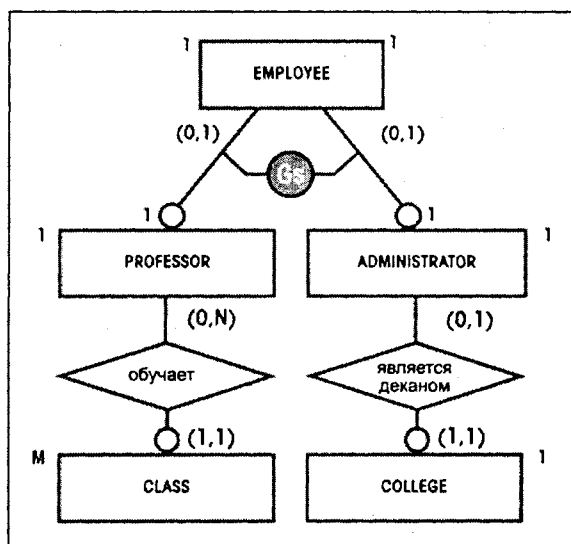


Рис. 3.41. Связи супертипов/подтипов

Всегда помните, что хороший проект БД требует, чтобы проектировщик учитывал требования к информационному обеспечению и бизнес-правила предприятия. В данной операционной среде Tiny College профессиональные качества администрации и преподавателей (ученые степени, звания и т. д.) идентичны. И поскольку ни для кого из остальных сотрудников (секретари, смотрители, механики и т. д.) не требуется такая ученая атрибутика, значимые связи супертипов/подтипов сокращены до одной — преподаватель/служащий. Связь супертип/подтип представлена на уровне реализации с помощью ER-фрагмента, показанного на рис. 3.42.

Поскольку большинство СУБД не поддерживают прямую связь супертип/подтип, проектировщики конвертируют ее в связь 1:1, как показано на ER-диаграмме (см. рис. 3.42), где преподаватель (PROFESSOR) является сотрудни-

ком (EMPLOYEE). В данном случае не все сотрудники являются преподавателями, поэтому сущность PROFESSOR необязательна для сущности EMPLOYEE (другими словами, сотрудник не обязательно должен быть преподавателем).

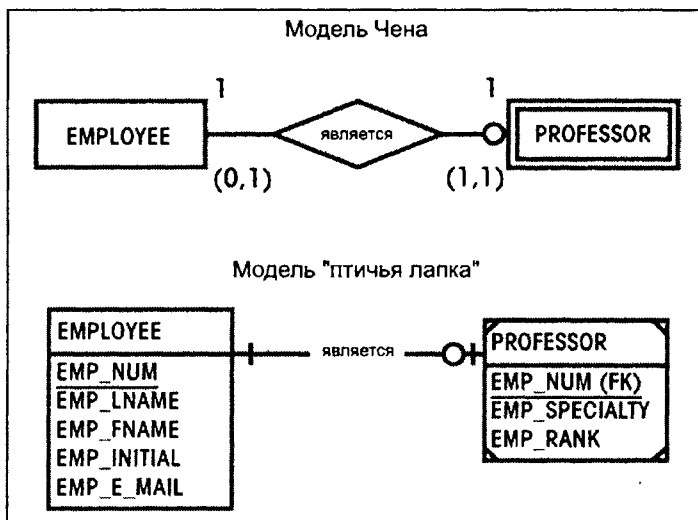


Рис. 3.42. Связь супертип/подтип на ER-диаграмме

Примечание

Существование связей 1:1 часто указывает на ошибочное восприятие атрибутов как сущностей, поэтому необходимо проявлять особую осторожность при оценке необходимости использования связи 1:1. В данном случае связь 1:1 должным образом обусловлена естественной связью супертип/подтип.

- На каждом факультете имеется несколько кафедр. Например, на факультете бизнеса есть кафедры бухгалтерского учета, менеджмента и маркетинга, экономики и финансов, а также компьютерных информационных систем. Еще раз напомним правило мощности связи: наименьшее число кафедр факультета — одна, максимальное число факультетов не определено (N). С другой стороны, каждая кафедра принадлежит только одному факультету, таким образом, мощность выражается как (1:1), т. е. минимальное и максимальное число факультетов, которым принадлежит кафедра, равно 1.

Принимая во внимание это описание деятельности колледжа, мы можем изобразить фрагмент ER-диаграммы (рис. 3.43). (Взгляните на рис. 3.42, где показано, что сущность PROFESSOR необязательна для сущности EMPLOYEE — не все сотрудники являются преподавателями. Также обратите внимание, что сущность PROFESSOR независима от существования сущности EMPLOYEE и наследует ее первичный ключ. Поэтому связь между EMPLOYEE и PROFESSOR является сильной, в то время как сущность PROFESSOR слабая.)

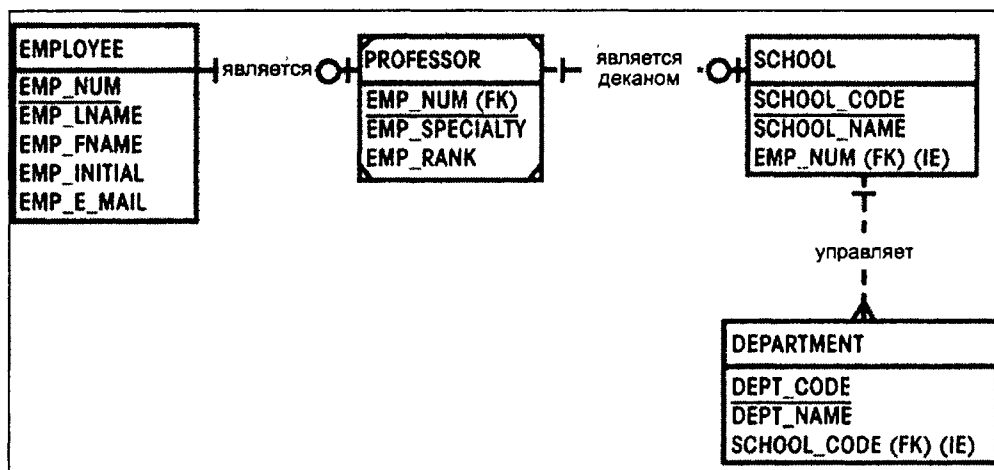


Рис. 3.43. Первый фрагмент ER-диаграммы Tiny College

Примечание

Опять-таки необходимо оценить причины появления связи 1:1 между сущностями PROFESSOR и SCHOOL в связи "является деканом". Стоит повторить, что существование связи 1:1 часто указывает на неверное определение атрибутов и сущностей. В данном случае мы можем легко устранить связь 1:1, сохраняя атрибуты деканов в таблице SCHOOL. Это решение также позволяет легко ответить на вопрос "кто является деканом факультета и каковы полномочия декана?" Отрицательная сторона такого решения состоит в том, что оно приводит к дублированию данных, уже хранящихся в таблице PROFESSOR, создавая, таким образом, почву для возникновения аномалий. Однако поскольку каждым факультетом руководит единственный декан, проблемы дублирования данных практически нет. Выбор какого-то одного подхода часто зависит от требований к информации, скорости транзакций и профессиональной интуиции проектировщика базы данных. Говоря кратко, не стоит необдуманно использовать связь 1:1, всегда необходимо убедиться, что каждая связь 1:1 в проекте БД оправдана.

- Каждая кафедра предлагает несколько курсов. Например, кафедра менеджмента и маркетинга предлагает такие курсы, как "Введение в менеджмент", "Принципы маркетинга", "Управление производством" и т. д. Фрагмент ER-диаграммы для этих условий представлен на рис. 3.44. Обратите внимание, что эта связь основана на бизнес-правилах работы Tiny College. Если, например, в Tiny College есть кафедры, которые можно считать "исследовательскими", они могут вообще не предлагать никаких курсов, следовательно, сущность COURSE (курс) может быть необязательной для сущности DEPARTMENT (кафедра).
- Мы иллюстрировали связь между COURSE и CLASS на рис. 3.14. Тем не менее, стоит повторить, что сущность CLASS (группа) является частью сущности COURSE (курс). Иначе говоря, кафедра может предлагать несколько разделов (групп) одного курса по базам данных. Каждая из этих групп обучается преподавателем в данный момент и в данном месте, т. е. существует связь 1:M между сущностями COURSE (курс) и CLASS (группа). Однако поскольку курс может существо-

вать в каталоге курсов колледжа Tiny College, даже если в нем нет групп, включенных в расписание, сущность CLASS необязательна для COURSE. Поэтому связь между COURSE и CLSS выглядит так, как это показано на рис. 3.45.

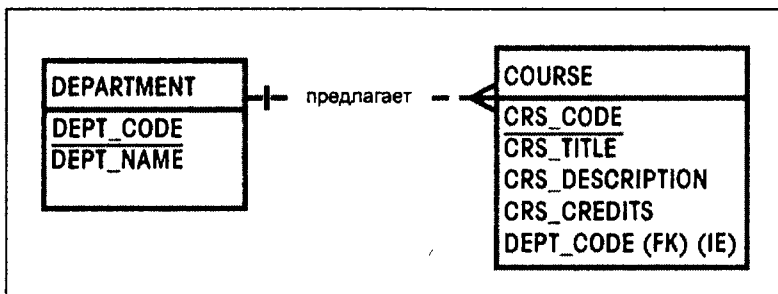


Рис. 3.44. Второй фрагмент ER-диаграммы Tiny College

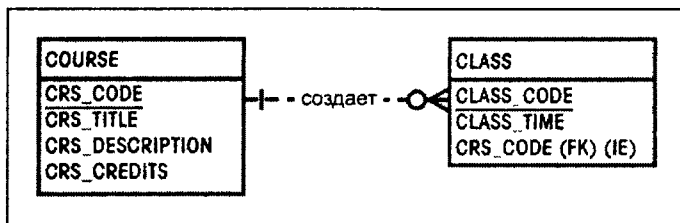


Рис. 3.45. Третий фрагмент ER-диаграммы Tiny College

- На каждой кафедре есть несколько преподавателей. Некоторые из преподавателей руководят кафедрами. Только один из преподавателей может возглавлять кафедру, на которой он работает, и ни один из преподавателей не обязан руководить кафедрой. Следовательно, сущность DEPARTMENT (кафедра) обязательна для сущности PROFESSOR (преподаватель) в связи "руководит". Эти связи суммированы в ER-фрагменте, представленном на рис. 3.46.

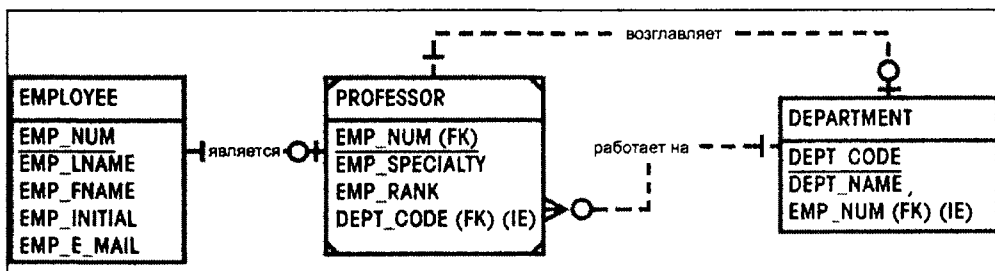


Рис. 3.46. Четвертый фрагмент ER-диаграммы Tiny College

- Каждый преподаватель может обучать до четырех групп, каждая из которых связана с разделом курса. Преподаватель может заниматься только исследовательской деятельностью и вообще не вести занятия в группах. Фрагмент ER-диаграммы, представленный на рис. 3.47, отображает такое положение дел.

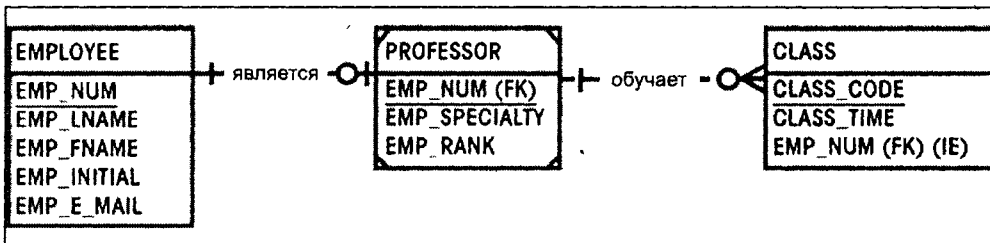


Рис. 3.47. Пятый фрагмент ER-диаграммы Tiny College

- Студент может быть записан в несколько групп, но за данный период обучения он может быть записан в группу только один раз. Например, в течение некоторого периода студент может решить пройти обучение по пяти предметам — "Статистика", "Бухгалтерский учет", "Английский язык", "Базы данных" и "История" — но данный студент не может быть записан в группу по предмету "Статистика" пять раз за период обучения! Каждый студент может записаться не более, чем в 6 классов, а в каждом классе может быть до 35 студентов, таким образом, имеет место связь M:N между сущностями **STUDENT** и **CLASS**. Поскольку группа (**CLASS**) может изначально существовать (в начале периода записи студентов в группы) даже в том случае, если в ней нет (пока) студентов, сущность **STUDENT** обязательна для сущности **CLASS** в связи M:N. Эту связь M:N нужно разделить на две связи 1:M с помощью промежуточной сущности **ENROLL** (список), как показано во фрагменте ER-диаграммы на рис. 3.48. Но обратите внимание, что рядом с сущностью **ENROLL** поставлен символ необязательности: если есть группа, в которой нет студентов, то эта группа не будет встречаться в таблице **ENROLL**. Отметим также, что сущность **ENROLL** слабая: она зависит от существования, и в состав ее первичного ключа входят первичные ключи сущностей **STUDENT** и **CLASS**.

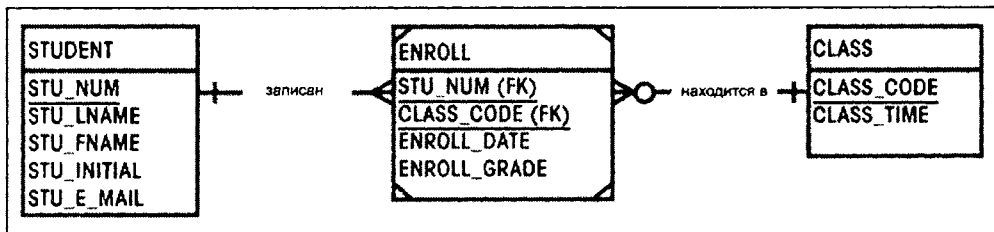


Рис. 3.48. Шестой фрагмент ER-диаграммы Tiny College

- На каждой кафедре обучается несколько (надеемся, что много!) студентов, которым кафедра предоставляет профилирующие предметы. Однако каждый студент имеет единственный профилирующий предмет и, следовательно, связан с единственной кафедрой (рис. 3.49). Однако в колледже Tiny College возможно — по крайней мере, пока, — чтобы студенты не объявляли свой профилирующий предмет. Такой студент не привязан к конкретной кафедре и поэтому сущность DEPARTMENT необязательна для сущности STUDENT. Еще раз повторим, что связи между сущностями и сущности сами по себе отражают деятельность организации, т. е. бизнес-правила предприятия определяют компоненты ER-диаграммы!

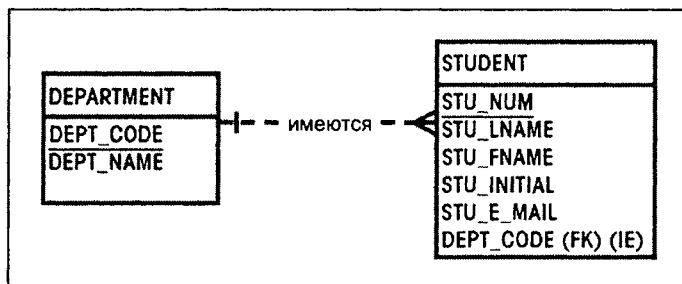


Рис. 3.49. Седьмой фрагмент ER-диаграммы Tiny College

- У каждого студента есть куратор на кафедре, каждый куратор консультирует несколько студентов. Куратор также является преподавателем, но не все преподаватели курируют студентов. Следовательно, сущность STUDENT необязательна для сущности PROFESSOR в связи "преподаватель (PROFESSOR) курирует студента (STUDENT)" (рис. 3.50).

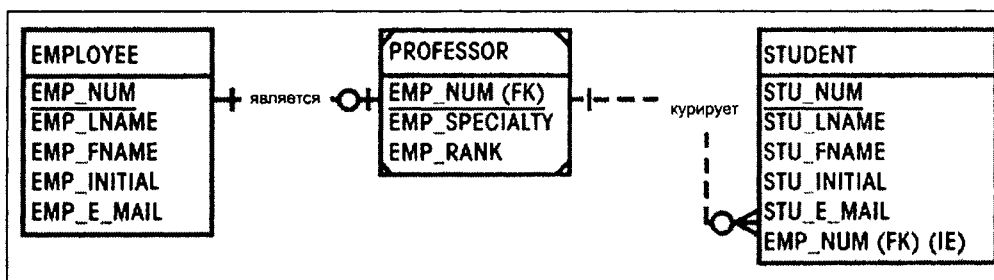


Рис. 3.50. Восьмой фрагмент ER-диаграммы Tiny College

- Если внимательно рассмотреть сущность CLASS на рис. 3.51, то можно заметить, что эта сущность содержит атрибут ROOM_CODE (код аудитории). Исходя из нашего соглашения об именах, понятно, что ROOM_CODE в CLASS является внешним ключом для сущности ROOM (аудитория). В свою очередь, каждая аудитория расположена в здании, поэтому мы создадим последний фрагмент диа-

граммы Tiny College исходя из того, что в здании (BUILDING) может быть несколько аудиторий (ROOM), но каждая аудитория (ROOM) находится в единственном здании (BUILDING) (рис. 3.51). На этом фрагменте ER-диаграммы видно, что в некоторых зданиях нет аудиторий. Например, в складских зданиях может не быть нумерованных аудиторий вообще.

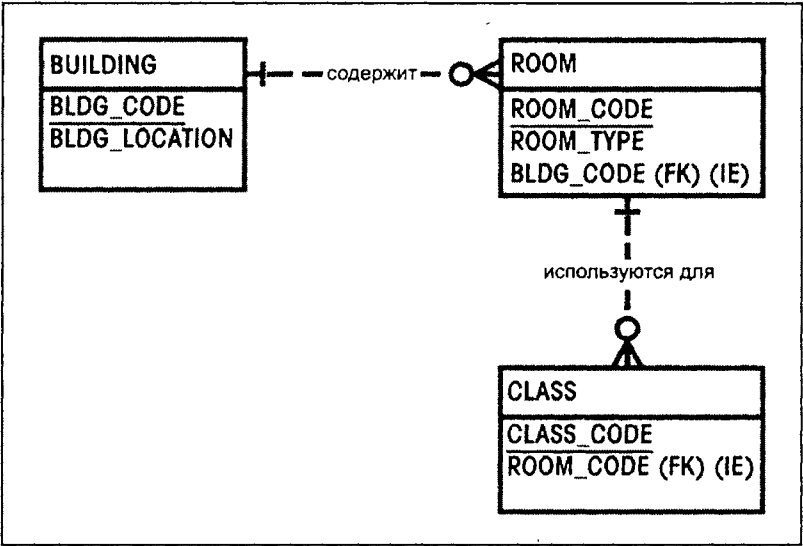


Рис. 3.51. Девятый фрагмент ER-диаграммы Tiny College

- На основе вышеизложенного мы можем определить следующие сущности:
- | | |
|------------|---|
| SCHOOL | COURSE |
| DEPARTMENT | CLASS |
| EMPLOYEE | ENROLL (промежуточная сущность между STUDENT и CLASS) |
| PROFESSOR | STUDENT |
| BUILDING | ROOM |

После того как мы определили значение сущности, мы можем определить предварительные связи между этими сущностями. Затем необходимо описать атрибуты сущностей. Определение атрибутов сущностей поможет нам лучше понять связи сущностей. Компоненты ER-модели, имена сущностей и их связи представлены в табл. 3.2.

Таблица 3.2. Компоненты ER-модели

Сущность	Связь	Связность	Сущность
SCHOOL (факультет)	имеет в своем составе	1:M	DEPARTMENT (кафедра)

Таблица 3.2 (окончание)

Сущность	Связь	Связность	Сущность
DEPARTMENT (кафедра)	обучаются	1:M	STUDENT (студент)
DEPARTMENT (кафедра)	нанимает на работу	1:M	PROFESSOR (преподаватель)
DEPARTMENT (кафедра)	предлагает	1:M	COURSE (курс)
COURSE (курс)	подразделяется на	1:M	CLASS (группа)
PROFESSOR (преподаватель)	является	1:1	EMPLOYEE (сотрудник)
PROFESSOR (преподаватель)	является деканом	1:1	SCHOOL (факультет)
PROFESSOR (преподаватель)	руководит	1:1	DEPARTMENT (кафедра)
PROFESSOR (преподаватель)	ведет занятия	1:M	CLASS (группа)
PROFESSOR (преподаватель)	курирует	1:M	STUDENT (студент)
STUDENT (студент)	записан в	M:N	CLASS (группа)
BUILDING (здание)	имеется	1:M	ROOM (аудитория)
ROOM (аудитория)	используется	1:M	CLASS (группа)

Примечание: ENROLL это составная сущность, реализующая связь "студенты (STUDENT) записаны (enroll) в группу (CLASS)".

Мы должны также определить связность и мощность связи для только что установленных связей путем широкого опроса конечных пользователей. Определив компоненты ER-модели, мы можем нарисовать ER-диаграмму или концептуальную схему (рис. 3.52). Фактически атрибуты сущности и их домены должны также отображаться на ER-диаграмме. Однако чтобы не перегружать диаграмму, атрибуты сущности можно изобразить отдельно.

3.6. Проблемы проектирования базы данных: противоречивые цели

Проектировщики базы данных часто должны идти на компромисс, что связано с противоречивыми целями, такими, например, как приверженность стандартам проектирования (элегантный дизайн), скорость обработки запросов и требования к информационному обеспечению.

- База данных должна проектироваться с учетом стандартов проектирования. Такие стандарты побуждают нас разрабатывать логические структуры, минимизирую-

щие избыточность данных, таким образом уменьшая вероятность деструктивных аномалий данных. Мы должны также изучить, каким образом стандарты определяют способы устранения пустых значений (null), иницируя разработку связей супертип/подтип. На самом деле, вы должны изучить, каким образом стандарты определяют представление всех компонентов внутри проекта БД. Говоря кратко, использование стандартов проектирования позволяет нам иметь дело со строго определенными компонентами и оценивать взаимодействие этих компонентов с определенной точностью. Без таких стандартов практически невозможно обеспечить надлежащий процесс проектирования, оценить имеющийся проект или проследить вероятное влияние изменений в проекте.

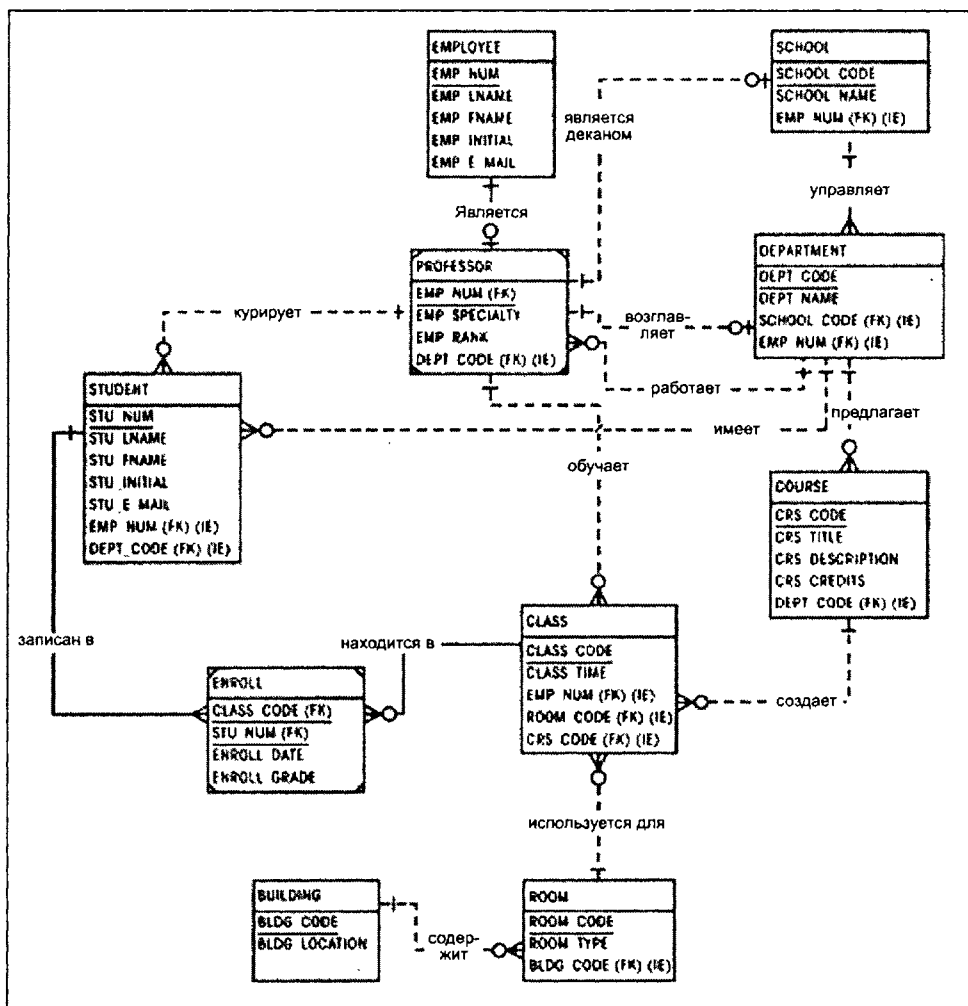


Рис. 3.52. Полная ER-диаграмма Tiny College

- Во многих организациях, особенно там, где выполняется очень много операций, высокая скорость обработки зачастую является самым важным аспектом проектирования БД. Высокая скорость обработки требует минимизации времени доступа, которая может быть достигнута минимизацией количества и сложности логически обусловленных связей. Поэтому требование высокой скорости выполнения транзакций может вынудить хранить данные внутри одной таблицы, в то время как проектировщик предпочитает расположить данные в различных таблицах для соответствия стандартам проектирования. Например, "безупречный" дизайн может использовать связь супертип/подтип 1:1, чтобы избежать пустых мест, в то же время проект, обеспечивающий высокую скорость выполнения транзакций, может вынудить объединить эти две таблицы, чтобы избежать использования дополнительной связи, применяя фиктивные элементы для устранения пустых мест (null).
- Получение регулярной, своевременной информации также может быть целью проектирования БД. Повышенные требования к информации могут стать причиной преобразования данных, приводящего к большому числу атрибутов и сущностей внутри проекта. Поэтому в базе данных, возможно, придется жертвовать некоторыми чисто дизайнерскими структурами и/или высокой скоростью выполнения транзакций, чтобы обеспечить максимальный объем выдаваемой информации.

Очевидно, что создание проекта, удовлетворяющего всем требованиям и стандартам проектирования, — очень важная задача. Однако если такой проект не соответствует требованиям клиента к скорости выполнения транзакций и к представлению информации, то, с точки зрения клиента, проектировщик не выполнил свою работу. Компромиссы в проекте это реальный факт.

Даже если нашей основной целью является надлежащее определение сущностей, атрибутов, связей и ограничений, мы, тем не менее, обязаны учитывать требования конечного пользователя, такие как эффективность, безопасность, совместный доступ, целостность данных и т. д. Проектировщик должен учесть все требования и проконтролировать доступность всех операций обновления, поиска и удаления. Наконец, грош цена проекту, если конечный продукт не может выполнять запросы и создавать необходимые отчеты.

Вы, по всей вероятности, поняли, что даже очень хорошо спроектированные ER-диаграммы будут обязательно изменяться в процессе эксплуатации. Это не должно оттолкнуть вас от ER-проектирования. ER-моделирование — важнейшая часть разработки хорошего проекта, который впоследствии может быть улучшен и расширен. Возможно, наибольшая выгода использования ER-диаграмм состоит в понимании процессов, происходящих на предприятии.

И, наконец, документировать, документировать и еще раз документировать! Записывайте все действия. Затем перечитывайте все то, что вы записали. Документация не только поможет отслеживать собственно проектирование, но и позволит вам (или тем, кто последует за вами) вспомнить весь процесс проектирования, когда придет время модифицировать проект. Хотя необходимость документирования очевидна, одна из самых болезненных проблем анализа баз данных и систем состоит в том, что правило "отражайте все в письменной форме" не выполняется ни на одной из стадий проектирования и разработки. Разработка стандартов на организационную документацию — очень важный аспект обеспечения совместимости и согласованности данных.

Резюме

Модель данных — относительно простая абстракция сложной картины реального мира. Проектировщики баз данных используют ее для взаимодействия с прикладными программистами и конечными пользователями.

Требования к моделированию данных зависят от различных представлений данных (глобальных или локальных) и уровня абстракции данных. Можно определить четыре уровня абстракции, представленные в табл. 3.3.

Таблица 3.3. Уровни абстракции данных

Модель	Уровень абстракции	Модель данных	Область действия	Зависимость	Модель БД
Концептуальная	Высокий	Сущность	Глобальная	Не зависит от аппаратного и программного обеспечения	Реляционная
Внешняя		ER-компоненты	Подмножество	Не зависит от аппаратного обеспечения	
Внутренняя		Реляционная и др.	Глобальная	Не зависит от аппаратного обеспечения	
Физическая	Низкий	Методы физического хранения	Не определена	Всегда зависит от программного и аппаратного обеспечения	Иерархическая и сетевая

В модели "сущность-связь" (ER-модели) для представления концепции базы данных с точки зрения конечного пользователя используются ER-диаграммы. Основными компонентами ER-модели являются сущности, связи и атрибуты. В ER-диаграмму также включены обозначения связности и мощности связи.

Связность обозначает тип связи (1:1, 1:M, M:N). Мощность связи выражает определенное число экземпляров сущности, связанных с экземплярами связанной сущности. Связность и мощность в основном определяются бизнес-правилами. В свою очередь бизнес-правила выводятся из детального описания информационной среды предприятия, деловых операций и информационных потребностей. Если это описание неадекватно или неточно, то выведенные из него бизнес-правила, вероятнее всего, станут причиной неадекватных и неточных моделей данных.

Для графического изображения компонентов и связей в ER-моделях используются специальные ER-обозначения. Такое графическое представление служит неким эквивалентом разрабатываемого проекта базы данных. Как нет смысла возводить здание без предварительной детальной проработки всех его структурных компонентов и

нанесения их на план, так нецелесообразно проектировать базы данных без тщательного изучения моделей данных, представленных ER-диаграммами.

Чтобы избежать избыточности данных и большого количества пустых мест в таблицах, сущности внутри иерархии обобщенных представлений можно подразделять на супертипы и подтипы. В супертипах хранятся совместно используемые свойства сущностей, в подтипах — их уникальные свойства. Фактически в иерархии обобщенных представлений супертип сущности отображается как предок каждого из подтипов сущности, т. е. подтип наследует все основные свойства и связи от родительского супертипа. Между супертипом и подтипом сущности существует связь 1:1, т. е. каждая строка подтипа соответствует только одной строке супертипа и наоборот. Связи супертипов и подтипов представлены на рис. 3.33 и 3.34.

В модели Чена используется алфавитно-цифровое обозначение связности и мощности связи, что позволяет проектировщикам БД моделировать природу экземпляров сущности и ее ограничений точнее, чем в моделях "птичья лапка", Rein85 и IDEF1X. Поэтому с точки зрения концептуального проектирования модели Чена более удобны. Однако модели "птичья лапка", Rein85 и IDEF1X с их более абстрактной символикой для обозначения связности и мощности и нацеленностью на реализацию проекта БД больше подходят для инструментальных средств CASE, чем модель Чена. Поскольку модель "птичья лапка" доступна в большинстве инструментальных средств проектирования баз данных, в последующих главах мы предпочли использовать именно эту модель. Однако независимо от избранной методологии логика моделирования остается одной и той же. Поскольку ни одна из ER-моделей не может точно отобразить все реальные данные и действующие ограничения, для выполнения, по крайней мере, некоторых бизнес-правил необходимо использовать прикладное программное обеспечение.

Проектировщики баз данных, независимо от того, насколько они способны обеспечить выполнение всех соглашений проекта, часто вынуждены идти на компромисс. Такой компромисс необходим, поскольку для конечных пользователей скорость выполнения транзакций или требования к характеру информации могут оказаться важнее "совершенной" логики модели и выполнения всех стандартов проектирования. Следовательно, проектировщики БД должны обладать профессиональным чутьем, чтобы определить, до какой степени можно изменять принятые соглашения моделирования. Чтобы быть уверенными в своих рассуждениях, проектировщики должны очень хорошо изучить существующие соглашения в моделировании данных.

Основные термины

ER-диаграмма — ERD

Бизнес-правила — business-rules

Бинарная связь — binary relationship

Связь — relationship

Внешняя модель — external model

Внутренняя модель — internal model

Домен — domain

Зависимость от существования — existence-dependent

Иерархия обобщенных представлений — generalization hierarchy

Итеративный процесс — iterative process

Компьютеризированное проектирование систем — computer-assisted systems engineering (CASE)

Концептуальная модель — conceptual model

Многозначный атрибут — multivalued attribute

Модель "птичья лапка" — Crow's Foot model

Модель — model

Модель IDEF1X

Модель Rein85

Модель данных — data model

Модель Чена — Chen model

Мощность — cardinality

Независимость от оборудования — hardware independence

Независимость от программного обеспечения — software independence

Независимость от существования — existence-independent

Необязательность — optionality

Необязательный — optional

Непересекающийся — disjoint, nonoverlapping

Обязательный — mandatory

Однозначный атрибут — single-valued attribute

Подтип — subtype

Производный атрибут — derived attribute

Промежуточная сущность — bridge entity

Простой атрибут — simple attribute

Рекурсивная связь (recursive relationship)

Связность — connectivity

Сильная (идентифицируемая) связь — strong (identifying) relationship

Слабая (неидентифицируемая) связь — weak (non-identifying) relationship

Слабая сущность — weak entity

Составная сущность — composite entity

Составной атрибут — composite attribute

Составной ключ — composite key

Супертип — supertype

Тернарная связь — ternary relationship

Унарная связь — unary relationship

Участник — participant

Физическая модель — physical model

Экземпляр сущности — entity instance, entity occurrence

Вопросы

1. Назовите и опишите различные уровни абстракции данных, определенные ANSI/SPARC.
2. Что представляют собой основные функциональные блоки модели "сущность-связь"? Расскажите о каждом.
3. Что такое составная сущность и когда она используется?
4. Почему моделирование данных считается "средством коммуникации"?
5. Предположим, что у вас имеется концептуальная модель, структура которой представлена на рис. 3.53. На базе этой модели создайте две внешних модели.

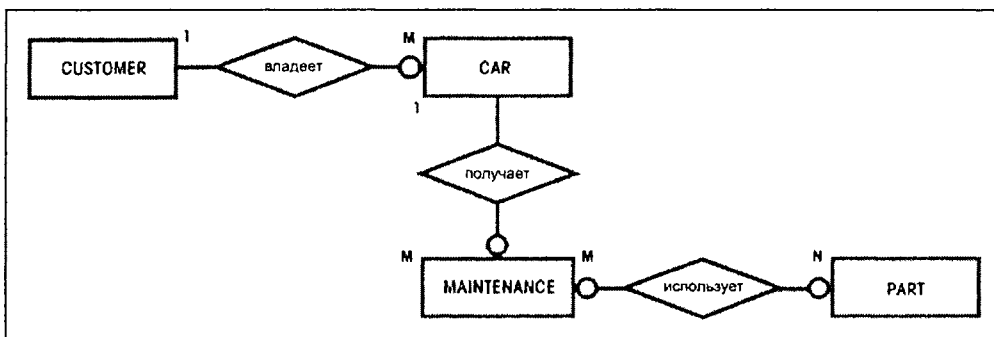


Рис. 3.53. Концептуальная модель к вопросу 5

6. Каким образом модели баз данных связаны с абстракцией данных? (Подсказка: почему проектировщики БД считают, что разработку проектов проще осуществлять с помощью реляционной модели?)
7. Как вы изобразите (графически) каждый из следующих компонентов ER-модели?
 - сущность
 - атрибут
 - связь
8. Компания Hudson Engineering Group (HEG) заключила с вами контракт на создание концептуальной модели БД, приложения которой должны отвечать требо-

ваниям программы обучения этой компании. Администратор НЕГ предоставил вам следующее описание операционной среды группы обучения:

В НЕГ имеется 12 инструкторов, в группе обучения может быть до 30 человек. НЕГ предлагает пять курсов "Высокие технологии", на каждом из которых может быть несколько групп. Если в группу набирается менее десяти человек, то она аннулируется. Следовательно, возможна ситуация, при которой на курсе не будет ни одной группы за семестр. Каждую группу обучает один инструктор. Каждый инструктор может обучать до двух групп или может заниматься только исследовательской работой. Каждый обучающийся может заниматься не более чем в двух группах за семестр.

Имея такие сведения, проделайте следующее.

- нарисуйте ER-диаграмму компании НЕГ;
 - опишите связь между инструктором и группой в терминах связности, мощности связи и зависимости от существования.
9. Поясните различие между составным ключом и составным атрибутом. Как каждый из них изображается на ER-диаграммах?
 10. Какие два возможных варианта действий есть у проектировщика БД, если ему встретился многозначный атрибут?
 11. Что такое производный атрибут? Приведите пример.
 12. Как обозначается связь между сущностями на ER-диаграммах? Приведите пример на основе моделей Чена и "птичья лапка".
 13. Что такое слабая сущность и как она изображается на ER-диаграммах? Приведите пример на основе моделей Чена и "птичья лапка".
 14. Как на ER-диаграммах изображается составная сущность и каковы ее функции? (Проиллюстрируйте на моделях Чена и "птичья лапка".)
 15. Используя следующие бизнес-правила, создайте соответствующие ER-диаграммы на основе моделей Чена и "птичья лапка" для каждой из связей:
 - в компании имеется четыре подразделения;
 - каждое подразделение нанимает сотрудников;
 - у каждого сотрудника может быть (а может и не быть) один или более иждивенцев;
 - каждый сотрудник может иметь (или нет) определенный трудовой стаж.
 16. Используя компоненты ER-диаграммы, разработанной в вопросе 15, создайте ER-диаграммы Чена и "птичья лапка", в которые включаются все эти компоненты.
 17. Какие три (часто взаимоисключающие) требования к БД должны учитываться в проекте базы данных?
 18. Кратко, но, по возможности, точно объясните различие между однозначными и простыми атрибутами. Приведите пример каждого из них.
 19. Что такое многозначные атрибуты и как ими оперируют в процессе проектирования БД?

Последние вопросы базируются на ER-диаграмме, представленной на рис. 3.54.

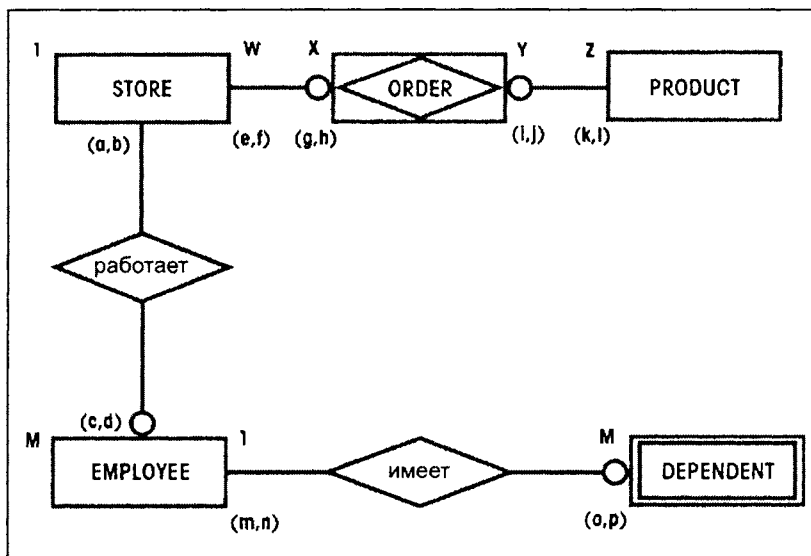


Рис. 3.54. ER-диаграмма к вопросам 20–23

20. Проставьте правильные значения мощностей для: (a,b) _____ (c,d) _____
 _____ (e,f) _____ (g,h) _____ (i,j) _____ (k,l) _____
 _____ (o,p) _____
21. Проставьте правильные связности для: W _____ X _____ Y _____
 Z _____
22. Какие два атрибута должны содержаться в составной сущности? В ответе используйте соответствующую терминологию.
23. Опишите точно состав первичного ключа слабой сущности. В ответе используйте соответствующую терминологию.
24. Конвертируйте ER-диаграмму модели Чена, представленную на рис. 3.54, в модель "птичья лапка".

Задачи

При решении первых трех задач используйте ER-модель Чена, представленную на рис. 3.55.

1. Используйте следующие бизнес-правила, чтобы правильно описать связности на ER-диаграмме:
 - отдел принимает на работу множество сотрудников, но каждый сотрудник работает в одном отделе;
 - некоторые сотрудники, которых называют "блуждающими", не принадлежат какому-то отделу;

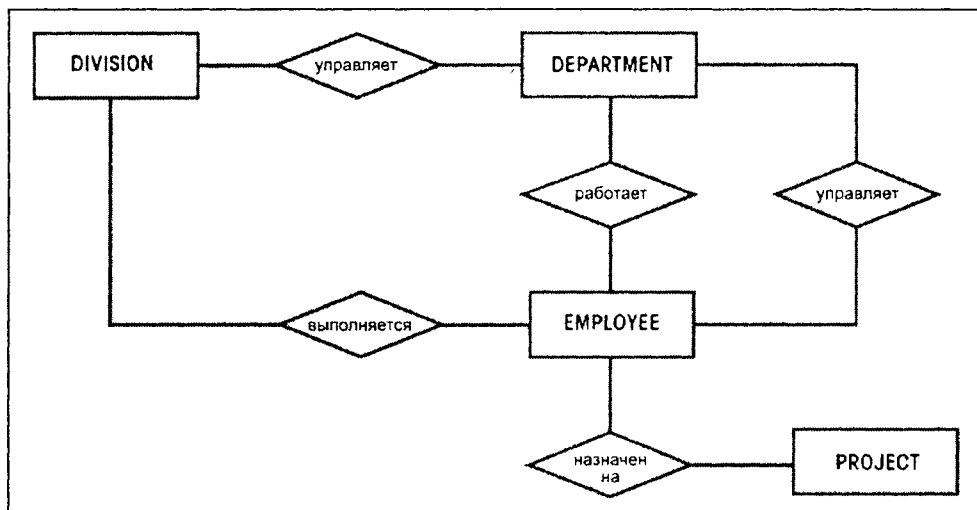


Рис. 3.55. ER-диаграмма к задачам 1—3

- в секторе может быть несколько отделов, но каждый отдел принадлежит только одному сектору;
 - сотрудник может работать в нескольких проектах, а проект может выполняться несколькими сотрудниками;
 - над проектом должен работать по крайней мере один сотрудник;
 - каждым отделом руководит кто-то из сотрудников, и каждым отделом руководит только один сотрудник;
 - каждый сектор возглавляется одним из сотрудников, и каждым сектором руководит только один сотрудник.
2. Укажите все мощности связи для данной модели.
 3. Модифицируйте ER-модель, разбив связь M:N на две связи 1:M, соединенные через составную сущность. Затем перепишите связности и мощности связи в соответствии со сделанными изменениями.
 4. Конвертируйте модель Чена, которую вы разработали в задачах 1—3, в модель "птичья лапка". Включите хотя бы минимальное число атрибутов, необходимых для реализации этой модели.
 5. Компания Temporary Employment Corporation (TEC) обеспечивает временной работой сотрудников на период максимальной загрузки. Менеджеры TEC предоставили следующее описание предприятия:
 - у TEC есть список кандидатов, готовых начать работу;
 - если кандидат до этого работал, то у него есть определенный трудовой стаж, зафиксированный в трудовой книжке. (Естественно, если кандидат до этого никогда не работал, то стаж отсутствует. Каждый раз после выполнения работ по найму в трудовой книжке создается одна запись.);

- каждый кандидат имеет несколько специальностей. Каждая специальность может иметься более чем у одного кандидата. (Например, возможно, что имеется более одного бакалавра в области бизнеса или несколько сертифицированных специалистов Microsoft. Не исключается вариант, когда кандидат может иметь обе таких специальности.);
- у ТЕС есть список компаний, которым требуются временные сотрудники;
- каждый раз, когда какой-то компании требуется временный сотрудник, ТЕС делает запись в папке Openings (вакансии). Эта папка содержит номер вакансии, название компании, требуемую специальность, дату начала работы, предполагаемую дату окончания работы и размер почасовой оплаты;
- на каждую вакансию требуется только одна основная специальность;
- если кандидат соответствует требуемой специальности, он получает эту работу, и при этом делается запись в папке Placement Record (регистрация приема на работу). В этой папке содержатся номер вакансии, номер кандидата, общее число отработанных часов и т. д. Кроме того, для кандидата делается запись в трудовой книжке;
- в списке вакансий ТЕС использует специальные коды для описания специальностей (список кодов приведен в табл. 3.4).

Таблица 3.4. Коды специальностей

Код	Описание
SEC-45	Секретарь, скорость набора текста не менее 45 слов в минуту
SEC-60	Секретарь, скорость набора текста не менее 60 слов в минуту
CLERK	Служащий
PRG-VB	Программист, Visual Basic
PRG-C++	Программист, C++
DBA-ORA	Администратор БД, Oracle
DBA-DB2	Администратор БД, DB2
SYS-1	Системный аналитик, уровень 1
SYS-2	Системный аналитик, уровень 2
NW-NOV	Сетевой администратор, опыт работы в Novell

Руководство ТЕС предполагает отслеживать следующие сущности:

COMPANY (компания)

OPENING (вакансия)

QUALIFICATION (специальность)

CANDIDATE (кандидат)

JOB_HISTORY (трудовой стаж)

PLACEMENT (место работы)

На основе этой информации проделайте следующее:

- нарисуйте ER-диаграмму "птичья лапка" этой компании;
 - определите возможные связи;
 - определите связность для каждой связи;
 - определите необязательность зависимости для этих связей;
 - раскройте связи M:N.
6. Организация Jonesburgh County Basketball Conference (JCBC) представляет собой любительскую ассоциацию баскетбола. В каждом городе (CITY) округа имеется одна команда (TEAM). В состав каждой команды входит не более 12 и не менее 9 игроков. В каждой команде может быть до трех тренеров (тренер нападения, тренер защиты и тренер с неполным рабочим днем). Каждая команда в течение сезона играет две игры (дома и на выезде) против каждой команды. Принимая во внимание эти условия, выполните следующее:
- определите связность для каждой связи;
 - определите тип зависимости между CITY и TEAM;
 - определите мощность связи "команда-игрок", а также "команда-город";
 - определите зависимость в связях "тренер-команда" и "команда-игрок";
 - создайте диаграммы Чена и "птичья лапка", представляющие базу данных JCBC.
7. Корпорация Automata Inc. производит по контракту специализированные автомобили. В компании имеется несколько подразделений, каждое из которых изготавливает отдельное транспортное средство, например, лимузин, трейлер, минивэн или RV (recreational vehicle, автомобили для семейного отдыха).

Когда новый автомобиль изготовлен, подразделение размещает в отделе закупок заказ на покупку отдельных компонентов. Отдел закупок корпорации Automata заинтересован в создании базы данных для отслеживания заказов и ускорения доставки оборудования.

Заказ, полученный отделом закупок, может содержать несколько различных позиций. Склад устроен так, что наиболее часто запрашиваемые позиции поставляются практически сразу же. Когда поступает заказ, он проверяется на наличие заказанных позиций на складе. Если позиция отсутствует, то ее необходимо заказать у поставщика. Каждая позиция может поставляться от нескольких поставщиков.

Используя это функциональное описание деятельности отдела закупок корпорации Automata, проделайте следующее:

- определите основные сущности;
- определите все связи и связности между сущностями;
- определите тип существующей зависимости во всех связях;
- дайте несколько примеров отчетов, которые могут быть получены с помощью такой БД.

8. Создайте ER-диаграмму на основе модели Чена, исходя из следующих требований:
- счет (INVOICE) выписывается торговым агентом (SALESREP). Каждый торговый агент может выписать несколько счетов, но каждый счет выписывается единственным агентом;
 - счет (INVOICE) выписывается для единственного клиента (CUSTOMER). Однако у каждого клиента может иметься несколько выписанных счетов;
 - в счет может включаться несколько строк (LINE), в которых описывается товар, купленный клиентом;
 - информация о товарах хранится в таблице PRODUCT (товар);
 - информация о поставщике товара хранится в таблице VENDOR (поставщик).

Примечание

В ER-диаграмме должны отображаться определенные вами (исходя из каких-то соображений) бизнес-правила. Убедитесь, что в вашей ER-диаграмме нашли отражение необходимые условия. Также убедитесь, что вы включили все атрибуты, которые позволяют успешно реализовать вашу модель.

9. На основе краткого описания бизнес-правил ресторанного сервиса фирмы ROBCOR и используя методологию "птичья лапка", нарисуйте ER-диаграмму с полным набором обозначений. Убедитесь, что включены все необходимые сущности, связи, связности и мощности.

Примечание

Ограничьте вашу ER-диаграмму сущностями и связями, основанными на представленных здесь бизнес-правилах. Иначе говоря, НЕ добавляйте "реализма" в ваш проект, расширяя его или облагораживая экзотическими бизнес-правилами! Однако убедитесь, что вы включили все атрибуты, позволяющие вам успешно реализовать проект.

В каждый обед входит одно первое блюдо, но каждое первое блюдо может входить в меню разных обедов. Гость может присутствовать на нескольких обедах, и на каждом обеде может быть несколько приглашенных. Каждое приглашение на обед может быть послано множеству гостей, и каждый гость может получить несколько приглашений.

10. Используя методологию "птичья лапка", создайте ER-диаграмму, которую можно применить для некоторой клиники, используя, например, такие бизнес-правила:
- пациенту может быть назначено несколько приемов у одного или более врачей клиники, а врач может принимать несколько пациентов. Однако каждый назначенный прием делается только одним врачом, и каждый назначенный прием относится к единственному пациенту;
 - экстренные случаи не требуют предварительно назначенного приема. Однако экстренный случай заносится в книгу как "прием вне расписания", чтобы можно было контролировать назначенные визиты;
 - если все идет своим чередом, то прием состоится в назначенное время. Результатом посещения будут диагноз и, при необходимости, назначенное лечение;

- при каждом визите обновляется запись в истории болезни пациента;
 - при каждом визите пациенту выписывается чек на оплату. Каждый чек за визит выписывается одним врачом, и каждый врач может выписать чек нескольким пациентам;
 - каждый чек должен быть оплачен. Однако чек может быть оплачен несколькими способами, а один платеж может погашать сразу более одного чека;
 - пациент может оплатить чек напрямую или на основе соглашения о медицинском страховании на своем предприятии;
 - если чек оплачивается страховой компанией, пациент освобождается от платы в соответствии с его страховой программой.
11. В Tiny College так понравились выполненные вами проект и реализация системы учета студентов, что они захотели расширить проект, включив в него свой небольшой гараж. Краткое описание операций таково:
- преподаватели могут использовать транспортные средства, принадлежащие Tiny College, для официально санкционированных поездок. Например, преподаватели могут использовать транспорт для поездки к студентам, обучающимся в центрах, находящихся за территорией кампуса, для поездки за необходимыми документами, для доставки студентов в заранее оговоренные места и для поездок, связанных с производственной необходимостью. Автомобили, используемые для этих целей, обслуживаются подразделением Travel Far But Slowly Center (TFBS) Tiny College;
 - используя регистрационные формы, каждый факультет может зарезервировать автомобиль для своей службы, отвечающей за заполнение путевого листа в конце каждой поездки. В регистрационную форму включаются предполагаемое время выезда, необходимый тип транспортного средства, пункт назначения и имя ответственного преподавателя. Когда ответственный преподаватель получает автомобиль, он должен подписать контрольную форму и взять заполненный путевой лист. (Сотрудник TFBS, который выдает автомобиль, также должен подписать контрольную форму.) Путевой лист преподавателя включает в себя идентификационный код преподавателя, идентификационный код транспортного средства, показания спидометра в начале и конце поездки, жалобы на обслуживание (если требуется), объем купленного топлива (если требуется), номер корпоративной кредитной карточки Tiny College, использовавшейся при оплате топлива. Если покупалось топливо, то квитанция об оплате по кредитной карте должна быть прикреплена к путевому листу. На основе этой квитанции и типа автомобиля (седан, минивэн, трак, микроавтобус) службы факультета рассчитывают пробег. (*Совет: НЕ используйте больше сущностей, чем необходимо. Помните разницу между сущностью и атрибутом!*);
 - обслуживание автомобилей осуществляет TFBS. Каждый раз, когда необходимо техническое обслуживание, в регистрационную форму техосмотра вносятся соответствующая запись, в которой приводятся краткое описание ремонта, даты начала и конца ремонта, а также идентификационный номер механика, выполнявшего ремонт (только авторизованный механик может ремонтировать автомобиль);

- как только создается регистрационная форма, ее номер заносится в подробную форму техобслуживания, а также передается руководителю отдела техобслуживания, который заполняет форму техобслуживания для регистрационной формы с данным номером. Подробная форма техобслуживания содержит различные строки для каждой выполненной работы, использованные запчасти и идентификационный номер механика, выполнявшего ремонт. Когда все позиции заполнены, подробная форма прикрепляется к регистрационной форме, проставляется дата, а механик, выполнивший ремонт, подписывает форму. Скрепленные формы затем подшиваются в папку, которая впоследствии используется для составления отчетов;
- в TFBS имеется склад запчастей (масло, масляные фильтры, воздушные фильтры и ремни различных видов). На складе ежедневно проверяется наличие запчастей и переоформляются заказы на запчасти, чтобы под рукой был всегда минимальный необходимый запас. Чтобы отслеживать использование запчастей, заведующий складом записывает номер регистрационной формы, для которой использовалась данная запчасть;
- каждый месяц TFBS выпускает отчеты: по пробегу автомобилей (информация по факультетам и по преподавателям на факультете); по расходам в расчете на каждое транспортное средство и факультет; об использовании запчастей; сводный отчет о техническом обслуживании автомобилей.

Используя это краткое описание операций, разработайте соответствующую (со всеми обозначениями) ER-диаграмму. Используйте модель Чена для обозначения сущностей, связей, связности и мощности.

12. Используя следующую информацию, составьте работоспособную ER-диаграмму (на основе модели Чена). Убедитесь, что включены все необходимые сущности, связи, связности и мощности связей.
 - Фирма EverFail выполняет быструю смену машинного масла и смазку. Хотя сюда пригоняют свои машины с тем, чтобы действительно быстро сделать замену масла, к вящему удовольствию клиента фирма EverFail также заменяет дворники, масляные фильтры и воздушные фильтры. В счет включаются смена масла и все замененные запчасти, а также стандартная смазка. Когда счет представлен к оплате, клиент платит наличными, использует кредитную карточку или выписывает чек. EverFail не предоставляет кредит на услуги. База данных EverFail создается для того, чтобы отслеживать все компоненты сделок.
 - Учитывая столь интенсивное использование материалов, фирма EverFail должна тщательно контролировать свой склад (расход масла, щетки, фильтры). Поэтому если количество материалов достигает некоторого минимума, их необходимо заказать у соответствующего поставщика. У EverFail есть список, в котором указаны как фактические, так и потенциальные поставщики.
 - Периодически фирма EverFail посылает уведомления своим клиентам на основе даты обслуживания автомобиля, а также отслеживает пробег автомобилей своих клиентов.

Примечание

Задачи 13 и 14 можно использовать в качестве основы для целого класса проектов. В этих задачах демонстрируются проблемы перевода описания операций в набор бизнес-правил, определяющих компоненты ER-диаграммы, которую впоследствии можно успешно реализовать. Эти задачи можно также использовать как основу для обсуждения компонентов и содержимого надлежащего описания операций. При этом нужно понять одну важную вещь (если вы действительно хотите создать БД, которую можно успешно реализовать): необходимо уметь в приведенном описании "фонový", несущественный материал отделять от существенных деталей, которые могут повлиять на проект БД в целом. Также надо иметь в виду, что многие ограничения не могут быть включены в проект БД, а должны учитываться в прикладных программах.

Хотя описание операций в задаче 13 связано с Web-бизнесом, целью выполнения упражнений являются именно аспекты *проектирования базы данных*, а не интерфейс и подробности управления транзакциями. На самом деле, можно утверждать, что сегодняшний Web-бизнес значительно повысил внимание к проектированию БД. (Возможно, вы и не столкнетесь с неприятностями, связанными с плохим проектом БД, если объемы продаж на вашем предприятии невелики, но при увеличении объема сделок — что и имеет место в Web-бизнесе — такой риск возрастает в значительной степени.)

13. Для выполнения этих упражнений используйте следующее описание операций компании RC_Model Company.

Компания RC_Model продает свои товары — пластмассовые модели (самолеты, корабли и автомобили) и дополнительные наклейки для них — через Web-сайт (www.rc-model.com). Модели (и наклейки) могут выполняться в масштабе от 1/144 до 1/32.

Клиенты используют Web-сайт для выбора товара и оплаты по кредитной карте. Если товар в настоящее время недоступен, то по желанию клиента можно оформить предварительный заказ. (Предварительный заказ не оплачивается до тех пор, пока он не будет доставлен клиенту.) После завершения транзакции печатается счет, а товары, перечисленные в нем, переносятся со склада в отдел доставки (стоимость доставки включается в счет). Отпечатанный счет помещается в упаковочную коробку. Оплата по кредитной карте клиента производится через CC Bank, в котором у RC_Model открыт счет. (*Примечание:* CC Bank не является подразделением компании RC_Model!)

Компания RC_Model отслеживает покупки клиентов и периодически рассылает им рекламные материалы. Поскольку руководство компании требует детальной информации об операциях, требуется создавать много отчетов. Эти отчеты включают в себя, помимо прочего, покупки клиентов по категориям и количеству, товарооборот, доходы по товарам и клиентам и т. д. Если товар не продавался в течение нескольких недель после его выпуска, он удаляется со склада и утилизируется.

Многие клиенты из списка компании RC_Model покупают товары компании. Но RC_Model также купила подписку на журнал *FineScale Modeler* для того, чтобы представить свой товар клиентам, еще не покупавшим у компании. Кроме того, данные о клиентах регистрируются даже в том случае, если клиент только запросил информацию о товарах у компании. Компания RC_Model получает товар непосредственно от изготовителей. Например, пластмассовые мо-

дели заказываются в компаниях Tamiya, Academy, Revell/Monogram и т. д. Наклейки заказываются в компаниях Aeromaster, Tauro, WaterMark и др. (*Внимание:* не все производители в БД RC_Model уже получили заказы.) Все заказы размещаются через Web-сайты производителей, а оплата заказов автоматически производится через счет RC_Model в банке CC Bank. Заказы делаются автоматически, когда на складе компании запас товара снижается до некоторого уровня. (Количество заказываемых единиц товара зависит от такого минимального количества товара на складе, определенного для каждой позиции.)

- Используя это (краткое и неполное) описание операций компании RC_Model, опишите все бизнес-правила для определения сущностей, связей, необязательности, связности и мощности связей. (*Совет:* используйте следующие три бизнес-правила в качестве примера и запишите остальные бизнес-правила в той же форме:
 - ◊ клиент может инициировать несколько счетов;
 - ◊ каждый счет инициируется только одним клиентом;
 - ◊ некоторые клиенты еще не инициировали ни одного счета.)
- Нарисуйте ER-диаграмму (со всеми обозначениями) с помощью модели "птичья лапка", основанной на бизнес-правилах, выработанных в первой части задачи 13. Включите все сущности, связи, необязательность, связность и мощность связи.

14. Для выполнения этих упражнений используйте следующее описание операций компании RC_Charter2.

Компания RC_Charter2 владеет флотом самолетов, сертифицированных по правилам FAR часть 135 (воздушные такси, или чартеры), принятых Federal Aviation Administration (FAA). Эти самолеты используются в качестве воздушных такси (чартеров) в США и Канаде. Чартерные компании выполняют так называемые "внеплановые" операции, т. е. чартерные полеты выполняются после того, как клиент зарезервировал самолет на определенные дату и время для перевозки пассажиров, груза, либо и того и другого вместе. Клиент может, конечно, зарезервировать несколько различных чартерных рейсов за некоторый промежуток времени. Однако для упорядочивания выписки счетов каждый чартерный рейс резервируется одним и только одним клиентом. Некоторые из клиентов RC_Charter2 не пользуются чартерными операциями компании, вместо этого они покупают топливо, используют обслуживающий персонал или другие службы компании. (*Внимание:* данный проект БД нацелен только на чартерные операции.)

Каждый чартерный маршрут приносит компании RC_Charter2 прибыль. Такие доходы образуются после оплаты клиентами выполненных рейсов. Чартерный рейс зависит от модели самолета, расстояния, времени ожидания, особых условий заказчика и опыта экипажа. Оплачиваемое расстояние рассчитывается путем умножения общей протяженности замкнутого маршрута на стоимость выполнения мили полета на данной модели. Длина замкнутого маршрута рассчитывается по навигационным полетным картам. Пример трассировки маршрута приведен на рис. 3.56. Обратите внимание, что длина маршрута рассчитывается так: $130 + 200 + 180 + 390 = 900$.

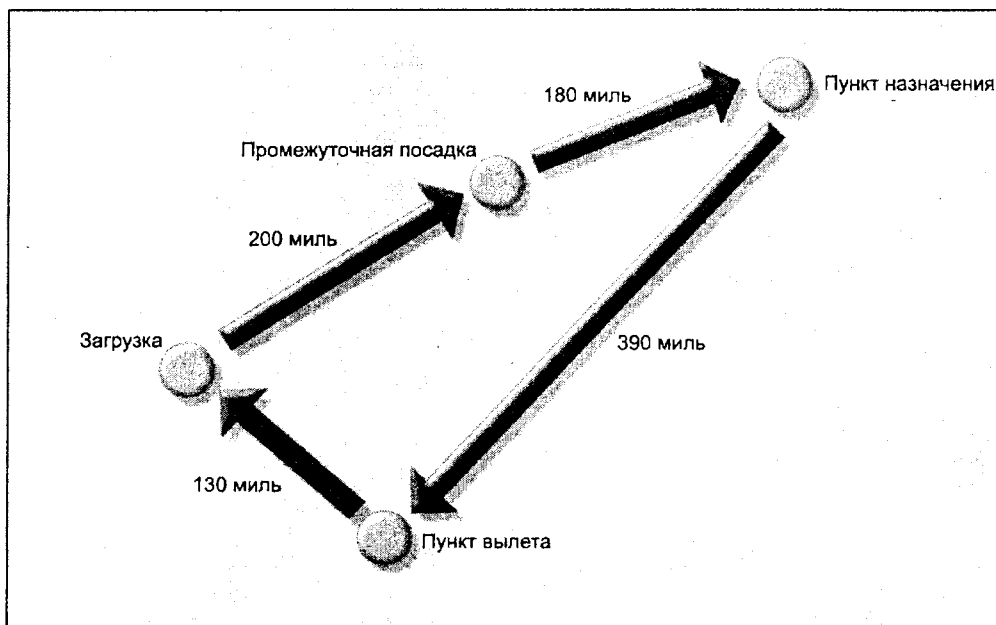


Рис. 3.56. Определение длины замкнутого маршрута

В зависимости от того, имеется или нет у клиента кредит в компании RC_Charter2, он может:

- оплатить выставленный счет по завершении полета;
- оплатить часть выставленного счета и занести остаток на свой счет. Размер остатка определяется размером кредита клиента;
- записать весь выставленный счет на свой счет. Размер оплаты не может превышать допустимый кредит клиента.

Клиенты могут оплатить всю стоимость имеющегося баланса по предыдущим рейсам. Такие платежи можно выполнить в любое время, нет необходимости привязывать это к определенному рейсу.

В стоимость чартерного рейса включается оплата пилотов и других членов экипажа, которая определяется правилами FAR, глава 135. Однако если клиенты потребуют увеличить экипаж, что не определено в требованиях FAR, глава 135, то такие клиенты должны оплачивать работу дополнительных членов экипажа по почасовому тарифу. Почасовая оплата зависит от квалификации дополнительного члена экипажа.

База данных должна позволять изменять назначение экипажа на полет. Для выполнения чартерного рейса необходим самолет, и каждым самолетом управляет экипаж. Экипаж самого маленького поршневого самолета состоит из одного пилота. В экипаж больших самолетов, т. е. большегрузного самолета, взлетный вес которого превышает 12 500 фунтов (≈ 5600 кг), и реактивных самолетов дол-

жен обязательно входить второй пилот, а на больших пассажирских самолетах требуется включить в экипаж стюардесс. В некоторых самолетах старого типа в составе экипажа должен находиться бортинженер, а для большегрузных самолетов требуется специалист по размещению груза. Говоря кратко, экипаж может состоять более чем из одного человека, и не все члены экипажа обязательно являются пилотами.

Оплата простоя самолета чартерного рейса рассчитывается умножением числа часов простоя на стоимость простоя для данной модели. Расходы экипажа определяются расходами на питание, ночлег (гостиницу, мотель) и транспорт до гостиницы и обратно.

База данных RC_Charter2 должна быть спроектирована для создания ежемесячных сводок обо всех чартерных рейсах, затратах и всех доходах, полученных на основе отчетов о выполненных рейсах. Такие отчеты основаны на данных, предоставляемых каждым пилотом экипажа после каждого рейса: дата и время рейса, пункт назначения, бортовой номер, данные о пилоте (и других членах экипажа), дистанция полета, расход топлива и другая необходимая информация о чартерном полете. Такая информация затем используется для создания ежемесячных отчетов о доходах и эксплуатационных расходах по клиентам, самолетам и пилотам. Все пилоты и другие члены экипажа являются сотрудниками компании RC_Charter2, т. е. компания не использует контрактных пилотов и членов экипажа.

Все операции проводятся в соответствии с требованиями FAR, глава 135, регламентирующими тренировку и лицензирование пилотов и членов экипажа. Например, пилоты должны иметь или коммерческую лицензию, или лицензию на пилотирование транспортных самолетов. Обе лицензии требуют соответствующего рейтинга. Рейтинги это специфические требования к компетенции. Например:

- для управления посадкой и взлетом многомоторных самолетов или только посадкой, подходящим рейтингом будет MEL, или многомоторные самолеты (Multi-Engine Landplane). Если многомоторные самолеты могут садиться или взлетать с поверхности воды, то приемлемым рейтингом будет MES (многомоторные гидросамолеты, Multi-Engine Seaplane);
- приборный рейтинг основан на демонстрации способности выполнять все операции полета только по показаниям приборов. Приборный рейтинг необходим для управления самолетом в условиях IMC (Instrument Meteorological Conditions, сложные метеорологические условия), а это управление подчиняется правилам IFR (Instrument Flight Rules, правила полета в сложных метеорологических условиях). В отличие от этого управление при "хорошей погоде" или при полете в условиях хорошей видимости основаны на правилах VFR (Visual Flight Rules, правила полета в условиях хорошей видимости);
- типовой рейтинг необходим для всех самолетов с взлетным весом более 12 500 фунтов (≈ 5600 кг) или для чисто реактивных самолетов. (Если самолет использует реактивную тягу для управления винтами, то его называют турбовинтовым. Такие самолеты не требуют типового рейтинга, даже если их взлетный вес превышает 12 500 фунтов).

Хотя действие лицензий и рейтингов пилотов не ограничено по времени, они все же требуют периодического медицинского освидетельствования пилотов в соответствии с правилами FAR, глава 135. Важно обратить внимание на следующее.

- Медицинский сертификат может иметь Класс I или Класс II (Class I или Class II). Класс I — более жесткий, чем Класс II, и должен обновляться каждые полгода. Класс II должен обновляться каждый год. Если Класс I не обновляется в течение шести месяцев, он автоматически переходит в Класс II. Если Класс II не обновляется в течение года, он автоматически переходит в Класс III, который не допускает пилотов к выполнению коммерческих рейсов.
- Глава 135 FAR по контролю полетов требует успешного прохождения контрольных проверок каждые шесть месяцев. Проверка на готовность к выполнению полетов включает в себя выполнение всех маневров и процедур, определенных в главе 135 FAR.

Члены экипажа, не являющиеся пилотами, тоже должны быть сертифицированы надлежащим образом. Например, мастер по закреплению груза должен иметь соответствующий сертификат, как и стюардессы. Кроме того, члены экипажа (например, мастер по закреплению груза), работающие на большегрузных самолетах (взлетный вес более 12 500 фунтов, и более 19 пассажиров), тоже должны проходить периодическую проверку, а также письменные и практические экзамены. Компания RC_Charter2 требует хранить все записи обо всех типах тестов, датах их проведения и результатах для каждого члена экипажа, а также дату проведения медицинского обследования пилотов.

Кроме того, члены экипажа должны периодически проходить тест на употребление наркотиков, результаты этих тестов также должны отслеживаться. (Обратите внимание, что члены экипажа, не являющиеся пилотами, не обязаны проходить тесты, специфичные для пилотов, которые требует FAR. Точно так же и пилоты не обязаны проходить тесты, например, те, которые требуются для специалистов по закреплению груза и стюардесс.) Однако многие члены экипажа имеют лицензии и/или сертификаты в нескольких областях. Например, пилот может иметь сертификат ATP и специалиста по закреплению груза. Если такой пилот назначается специалистом по закреплению груза на данный рейс, то ему потребуется соответствующий сертификат. Точно так же стюардесса может иметь сертификат на выполнение коммерческих рейсов. Примерные форматы данных приведены в табл. 3.5—3.8.

Таблица 3.5. Тесты

Код теста	Описание теста	Периодичность тестирования
1	Проверка навыков пилотирования в соответствии с FAR, глава 135	6 месяцев
2	Медицинский Класс 1	6 месяцев
3	Медицинский Класс 2	12 месяцев
4	Практика специалиста по закреплению груза	12 месяцев

Таблица 3.5 (окончание)

Код теста	Описание теста	Периодичность тестирования
5	Практика стюардессы	12 месяцев
6	Тест на наркотики	Случайная
7	Письменные экзамены	6 месяцев
...

Таблица 3.6. Результаты тестирования

Сотрудник	Код теста	Дата теста	Результаты тестирования
101	1	12 ноября 2001	Пройден
103	6	23 декабря 2001	Пройден
112	4	23 декабря 2001	Пройден
103	7	11 января 2002	Пройден
112	7	16 января 2002	Пройден
101	7	16 января 2002	Пройден
101	6	11 февраля 2002	Пройден
125	2	15 февраля 2002	Пройден
...

Таблица 3.7. Лицензии и сертификаты

Лицензия/сертификат	Описание
ATP	Airline Transport Pilot (пилот транспортных самолетов)
Comm	Commercial Pilot (пилот коммерческих рейсов)
Med-1	Class-1 Medical (медицинский Класс I)
Med-2	Class-2 Medical (медицинский Класс II)
Instr	Instrument rating (полет по приборам)
MEL	Multiengine rating (управление многомоторными самолетами)
LM	Loadmaster (специалист по закреплению грузов)
FA	Flt. Attendant (стюард, стюардесса)

Таблица 3.8. Лицензии и сертификаты сотрудников

Сотрудник	Лицензия/сертификат	Дата получения
101	ATP	12 ноября 1992
103	Comm	23 декабря 1990
112	FA	23 декабря 1998
103	Instr	11 января 1991
112	LM	16 января 1996
101	Comm	16 января 1987
101	Med-1	11 февраля 2001
101	Instr	15 февраля 1988
...

Пилоты и другие члены экипажа должны проходить периодическую тренировку в соответствии с их квалификацией. Периодические тренировки основаны на учебных планах FAA. Например, периодическая тренировка пилотов включает в себя обзор всех правил полетов в соответствии с FAR, глава 135, интерпретацию метеорологических данных, требования к полетным операциям компании, специфические полетные операции и т. д. Компания RC_Charter2 требует хранения всех записей обо всех периодических тренировках каждого члена экипажа, который проходит такие тренировки.

Компания RC_Charter2 требует подробной регистрации всех полномочий всех членов экипажа и всех обязательных тренировок, определенных FAR, глава 135. Полная запись должна содержать все требования и всю необходимую информацию.

Для выполнения чартерного рейса компания должна иметь соответствующий самолет. Этот самолет должен пилотировать пилот-командир (Pilot in Command, PIC) с лицензией, отвечающей требованиям FAA и соответствующими документами. Такой самолет оборудован поршневыми или турбовинтовыми двигателями и имеет взлетный вес менее 12 500 фунтов, допускает управление одним пилотом, имеющим разрешение в соответствии с FAR, и оборудован автопилотом. Однако даже несмотря на то что FAR допускает, чтобы таким самолетом управлял один пилот, многие клиенты требуют наличия второго пилота, который также может выполнять полет в соответствии с требованиями FAR.

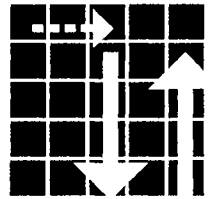
Директор-распорядитель компании RC_Charter2 собирается получить в лизинг турбореактивный самолет, а такому самолету требуется экипаж из двух пилотов. Оба пилота должны иметь лицензию в соответствии с FAR, глава 135, рейтинг и соответствующую подготовку.

Компания также взяла в лизинг большегрузный самолет взлетным весом более 12 500 фунтов. Такой самолет может перевозить большое число пассажиров, что требует наличия одной и более стюардесс. Такой самолет также требует наличия в экипаже, при перевозке груза свыше 12 500 фунтов, специалиста по безопасно-

сти и закреплению грузов. База данных должна разрабатываться так, чтобы в нее можно было включать дополнительные чартерные экипажи.

- Используя это (неполное) описание операций компании, опишите все прикладные бизнес-правила для выявления сущностей, связей, необязательности, связности и мощности. (*Совет:* используйте следующие пять бизнес-правил в качестве примера и запишите остальные правила в таком же виде:
 - ◇ клиент может заказать несколько чартерных рейсов;
 - ◇ каждый чартерный рейс заказывается только одним клиентом;
 - ◇ некоторые клиенты не сделали (пока) заказов на чартерные рейсы;
 - ◇ сотрудник может быть назначен на несколько чартерных рейсов;
 - ◇ в каждом чартерном рейсе в качестве членов экипажа может участвовать несколько сотрудников.)
- Нарисуйте ER-диаграмму с полным набором обозначений в соответствии с моделью "птичья лапка", основываясь на бизнес-правилах, которые вы только что записали. Включите все сущности, связи, необязательность, связность и мощность связи.

Глава 4



Нормализация таблиц базы данных

В этой главе мы обсудим следующие вопросы:

- ☐ что такое нормализация и какую роль она играет в процессе проектирования баз данных?
- ☐ нормальные формы 1НФ, 2НФ, 3НФ, НФБК и 4НФ;
- ☐ как преобразовать нормальную форму более низкого уровня к более высокому?
- ☐ как совместно использовать ER-моделирование и нормализацию для получения хорошего проекта базы данных?
- ☐ как с помощью денормализации можно повысить эффективность обработки информации?

Обзор

Невозможно создать хороший проект реляционной базы данных без правильно построенных таблиц. В этой главе мы изучим способы создания правильной структуры таблиц, позволяющей контролировать избыточность данных и, следовательно, предотвратить их аномалии. Процесс, который позволяет добиться таких результатов, называется нормализацией.

Для того чтобы распознавать и понимать свойства правильной табличной структуры, полезно понять, что же такое плохая структура. Поэтому мы начнем с изучения свойств таблицы с плохой структурой и проблем, создаваемых такой таблицей. Затем мы покажем, как можно исправить структуру такой таблицы. Эта методология дает большие преимущества: вы узнаете, как проектировать таблицы с хорошей структурой и как можно исправить плохо структурированные таблицы¹.

Вы узнаете, что нормализация позволяет устранить аномалии данных. Кроме того, работа с нормализованными таблицами будет гораздо эффективнее, чем с таблич-

¹ Часть материала этой главы основана на работе Питера Роба, Карлоса Коронела и Натана Адамса (C. Nathan Adams) "Relational Database Design at a Construction Company: A Problem or a Solution?" Journal of Systems Management (Cleveland, Ohio), т. 42, н. 8, вып. н. 362, August, 1991, стр. 23—36. Напечатано с разрешения Journal of Systems Management. В этой статье исследуются некоторые проблемы, возникающие у неопытных проектировщиков БД. Доступность простых в использовании РСУБД позволяет новичкам быстро создавать плохие решения. В связи с этим умение корректировать плохие табличные структуры является полезным навыком.

ными структурами, не подвергшимися нормализации. Вы увидите, что набор нормализованных табличных структур точнее отражает реальную деятельность предприятия.

4.1. Таблицы базы данных и нормализация

Только хорошего программного обеспечения базы данных недостаточно, чтобы избежать проблем с избыточностью данных, которые мы обсуждали в *гл. 1*. Если таблицы базы данных рассматривать всего лишь как файлы некоей системы файлов, то РСУБД никогда не сможет продемонстрировать все свои уникальные возможности обработки данных.

Таблица — основной строительный элемент в процессе проектирования реляционной БД. Следовательно, структура таблицы представляет для нас особый интерес. В идеальном случае в процессе проектирования, который мы изучали в *гл. 3*, мы получаем хорошо сконструированные таблицы. Но вероятность получить плохие табличные структуры есть даже при правильном проектировании БД. Так как же нам научиться распознавать плохие табличные структуры и как научиться строить хорошие таблицы? Ответ на эти вопросы кроется в нормализации таблиц. *Нормализация (normalization)* это процесс назначения атрибутов сущностям. Нормализация уменьшает избыточность данных и в более широком смысле помогает устранить аномалии данных, появляющихся вследствие избыточности. Нормализация не устраняет избыточность данных вообще, вместо этого она дает возможность контролировать избыточность, что позволяет корректно связывать таблицы базы данных.

Процесс нормализации состоит из нескольких этапов, на каждом из которых определяются так называемые *нормальные формы*. Результатом первых трех этапов являются соответственно первая нормальная форма, 1НФ (First Normal Form — 1NF), вторая нормальная форма, 2НФ (Second Normal Form — 2NF) и третья нормальная форма, 3НФ (Third Normal Form — 3NF). С точки зрения структуры таблиц 2НФ лучше, чем 1НФ, а 3НФ лучше, чем 2НФ. В большинстве проектов БД третья нормальная форма (3НФ) завершает процесс нормализации. Однако в *разд. 4.3* будет показано, почему иногда приходится иметь дело и с четвертой нормальной формой (4НФ), а в некоторых специальных приложениях могут потребоваться формы и более высокого уровня, чем четвертая. Такие приложения обычно связаны со статистическими исследованиями, которые выходят за пределы контекста большинства бизнес-операций. Поэтому в нашей книге мы не будем рассматривать нормальные формы выше четвертой.

Хотя нормализация является очень важной составляющей процесса проектирования БД, не надо полагать, что всегда предпочтительнее более высокая нормальная форма. Вообще говоря, чем выше нормальная форма, тем больше потребуется операций соединения для получения результата и, следовательно, тем медленнее система будет реагировать на запросы конечного пользователя. Успешный проект должен учитывать требования конечного пользователя к производительности базы данных. Поэтому вы должны быть готовы и к *денормализации* некоторых фрагментов проекта БД для удовлетворения требованиям конечного пользователя. (Денормализация приводит к понижению уровня формы, т. е. в процессе денормализации форма 3НФ конвертируется в форму 2НФ.) Однако *надо отдавать себе отчет, что повышение*

производительности посредством денормализации непременно приведет к повышению уровня избыточности данных.

4.1.1. Необходимость нормализации

Для того чтобы продемонстрировать процесс нормализации, рассмотрим простейшее бизнес-приложение. В данном случае мы будем исследовать работу с упрощенной базой данных строительной компании, которая занимается проектированием зданий. У каждого проекта имеются свои уникальный номер, название, штат сотрудников и т. д. У каждого сотрудника есть идентификационный номер и специальность, например, инженер или специалист по компьютерам.

Компания выставляет своим клиентам счета в соответствии с трудозатратами в человеко-часах на выполнение данного контракта. (Например, стоимость одного часа работы специалиста по компьютерам отличается от стоимости часа работы инженера.) Периодически компания составляет отчеты, содержащие информацию, приведенную в табл. 4.1.

Общий итог (total charge) в табл. 4.1 — это производный атрибут и поэтому он не хранится в БД.

Может показаться, что самый простой и быстрый способ составить необходимый отчет состоит в том, чтобы создать таблицу, содержимое которой соответствовало бы форме самого отчета (рис. 4.1).

Поскольку общая стоимость работ получается умножением отработанных часов на стоимость часа работы, эта позиция отражена в таблице на рис. 4.1. К сожалению, структура таблицы, представленная на рис. 4.1, не соответствует требованиям, которые мы обсуждали в гл. 2, и поэтому не может считаться удобным средством манипулирования данными.

- ❑ Номер проекта (PROJ_NUM), несомненно, можно было бы использовать в качестве первичного ключа или, по крайней мере, части первичного ключа, однако в нем имеются пустые места (null).
- ❑ Вид элементов таблицы стимулирует противоречивость данных. Например, значение JOB_CLASS (специализация) "Elect. Engineer" (инженер-электрик) в одном случае введена как "Elect. Eng", в другом "El. Eng" или даже "EE".
- ❑ В таблице имеется очевидная избыточность данных, что приводит к следующим аномалиям.
 - *Аномалии обновления (update anomalies).* Изменение JOB_CLASS для сотрудника с номером (EMP_NUM) 105 требует выполнения этого действия в нескольких строках таблицы, где встречается EMP_NUM = 105.
 - *Аномалии включения (insertion anomalies).* Для занесения информации в любую строку необходимо вводить в проект какого-нибудь сотрудника. Если сотрудник еще не включен в какой-либо проект, то для того, чтобы занести в таблицу сведения о сотруднике, придется создавать фиктивный проект.
 - *Аномалии удаления (deletion anomalies).* Если сотрудник с номером 103 увольняется, то необходимо будет удалить все строки, где EMP_NUM = 103. При этом может быть утеряна и важная информация.

Таблица 4.1. Примерное содержимое отчета

Proj. Number	Project Name	Employee Number	Employee Name	Job Class.	CHG/HR	Hours Billed	Total Charge
15	Evergreen	103	June E. Arbough	Инженер-электрик	\$84.50	23.8	\$2,011.10
		101	John J. News	Проектировщ. БД	\$105.00	19.4	\$2,037.00
		105	Alice K. Johnson*	Проектировщ. БД	\$105.00	35.7	\$3,748.50
		106	William Smithfield	Программист	\$35.75	12.6	\$450.45
		102	David H. Senior	Сист. аналитик	\$96.75	23.8	\$2,302.65
Subtotal							\$10,549.70
18	Amber Wave	114	Annelise Jones	Инж. проектировщ.	\$48.10	24.6	\$1,183.26
		118	James J. Frommer	Общие работы	\$18.36	45.3	\$831.71
		104	Anne K. Ramoras*	Сист. аналитик	\$96.75	32.4	\$3,134.70
		112	Darlene M. Smithson	Аналитик по принятию решений	\$45.95	44.0	\$2,021.80
		Subtotal					
22	Rolling Tide	105	Alice K. Johnson	Проектировщ. БД	\$105.00	64.7	\$6,793.50
		104	Anne K. Ramoras	Сист. аналитик	\$96.75	48.4	\$4,682.70
		113	Delbert K. Joenbrood*	Инж. проектировщ.	\$48.10	23.6	\$1,135.16
		111	Geoff B. Wabash	Канцелярия	\$26.87	22.0	\$591.14
		106	William Smithfield	Программист	\$35.75	12.8	\$457.60
Subtotal							\$13,660.10

Таблица 4.1 (окончание)

Proj. Number	Project Name	Employee Number	Employee Name	Job Class.	CHG/HR	Hours Billed	Total Charge
25	Starflight	107	Maria D. Alonzo	Программист	\$35.75	24.6	\$879.45
		115	Travis B. Bawang*	Сист. аналитик	\$96.75	45.8	\$4,431.15
		101	John J. News*	Проектировщ. БД	\$105.00	56.3	\$5,911.50
		114	Annelise Jones	Инж. проектировщ.	\$48.10	33.1	\$1,592.11
		108	Ralph B. Washington	Сист. аналитик	\$96.75	23.6	\$2,283.30
		118	James J. Frommer	Общие работы	\$18.36	30.5	\$559.98
		112	Darlene M. Smithson	Аналитик по принятию решений	\$45.95	41.4	\$1,902.33
Subtotal							\$17,559.82
Total							\$48,941.09

Примечание: Звездочкой обозначены руководители проекта, где:

Proj Number — номер проекта
 Project Name — название проекта
 Employee Number — номер сотрудника
 Job Class — спецификация
 CHG/HR — стоимость часа работы
 Hours Billed — отработанные часы
 Total Charge — общая стоимость работ
 Subtotal — промежуточный итог
 Total — общий итог

	PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
▶	15	Evergreen	103	June E. Arbough	Elect. Engineer	\$84.50	23.8
			101	John G. News	Database Designer	\$105.00	19.4
			105	Alice K. Johnson *	Database Designer	\$105.00	35.7
			106	William Smithfield	Programmer	\$35.75	12.6
			102	David H. Senior	Systems Analyst	\$96.75	23.8
	18	Amber Wave	114	Annelise Jones	Applications Designer	\$48.10	24.6
			118	James J. Frommer	General Support	\$18.36	45.3
			104	Anne K. Ramoras *	Systems Analyst	\$96.75	32.4
			112	Darlene M. Smithson	DSS Analyst	\$45.95	44.0
	22	Rolling Tide	105	Alice K. Johnson	Database Designer	\$105.00	64.7
			104	Anne K. Ramoras	Systems Analyst	\$96.75	48.4
			113	Delbert K. Joenbrood *	Applications Designer	\$48.10	23.6
			111	Geoff B. Wabash	Clerical Support	\$26.87	22.0
			106	William Smithfield	Programmer	\$35.75	12.8
	25	Starflight	107	Maria D. Alonzo	Programmer	\$35.75	24.6
			115	Travis B. Bawangi	Systems Analyst	\$96.75	45.8
			101	John G. News *	Database Designer	\$105.00	56.3
			114	Annelise Jones	Applications Designer	\$48.10	33.1
			108	Ralph B. Washington	Systems Analyst	\$96.75	23.6
			118	James J. Frommer	General Support	\$18.36	30.5
			112	Darlene M. Smithson	DSS Analyst	\$45.95	41.4

Рис. 4.1. Таблица со структурой, соответствующей формату отчета

Может показаться, что структура таблицы работает правильно, и выполнить отчет очень просто. Но предположим, что сотрудницу Darlene M. Smithson переводят в проект Evergreen. Оператор, отвечающий за ввод информации, должен внести в файл PROJECT следующую строку:

```
15 Evergreen 112 Darlene M. Smithson DSS Analyst $45.95 0.0
```

чтобы заполнить соответствующие атрибуты PROJ_NUM, PROJ_NAME, EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOUR и HOURS²

Каждый раз, когда на проект назначается новый сотрудник, некоторые элементы данных (PROJ_NAME, EMP_NAME, CHG_HOUR) приходится вводить без всякой на то необходимости. Представьте себе объем ненужного ввода данных, если потребуется ввести 200—300 элементов таблицы. Ведь достаточно ввести соответствующий идентификационный номер сотрудника, чтобы можно было идентифицировать Darlene M. Smithson, ее специализацию и стоимость часа ее работы. Поскольку лишь один сотрудник имеет номер 112, его персональные данные (имя, специальность и т. д.) не должны набираться всякий раз при обновлении файла

² Еще раз обратите внимание, что применяемое соглашение об именовании показывает, что означает каждый атрибут и откуда он происходит. Например, в атрибуте PROJ_NAME использован префикс PROJ, обозначающий, что этот атрибут связан с таблицей PROJECT, в то время как вторая часть — NAME (имя) — говорит сама за себя. Однако необходимо помнить, что длина названия также является потенциальным источником ошибок, особенно при обозначении префиксов. Например, предпочтительнее использовать префикс CHG, а не CHARGE (стоимость). (В контексте этой БД трудно предположить, что этот сокращенный префикс может означать что-нибудь другое.) — Прим. пер.

таблицы. К сожалению, структура, представленная на рис. 4.1, не предоставляет такой возможности.

Избыточность данных, имеющая место в таблице на рис. 4.1, может привести к неэкономному использованию дискового пространства. К тому же избыточность данных является причиной аномалий. Предположим, например, что оператор вводит следующую информацию:

15 Evergreen 112 Darla Smithson DCS Analyst \$45.95 0.0

На первый взгляд эти данные корректны. Но скажите, Evergreen это тот же проект, что и Evergreen? А DCS Analyst — то же самое, что и DSS Analyst? И Darla Smithson — та же личность, что и Darlene M. Smithson? Подобная путаница вызвана проблемой целостности данных, и произошла она потому, что при вводе информации не соблюдалось правило, предписывающее, что все копии избыточных данных должны быть идентичны.

При проектировании БД проблемы целостности данных, которые могут быть вызваны их избыточностью, обязательно должны приниматься во внимание. В среде реляционных баз данных такие проблемы можно разрешить достаточно легко.

4.1.2. Приведение к первой нормальной форме

Поскольку в реляционной модели данные рассматриваются как часть таблицы или набора таблиц, именно в таблицах и должны быть определены все основные значения. Обратите внимание, что на рис. 4.1 имеются *повторяющиеся группы*, поскольку в проекте с данным номером (PROJ_NUM) может содержаться целая группа, состоящая из нескольких значений, т. е. один экземпляр сущности (PROJ_NUM) имеет несколько значений. Посмотрите, например, как на рис. 4.2 представлены компоненты проекта Evergreen.

	PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
▶	5	Evergreen	103	June E. Arbough	Elect. Engineer	\$84.50	23.8
■			101	John G. News	Database Designer	\$105.00	19.4
■			105	Alice K. Johnson *	Database Designer	\$105.00	35.7
■			106	William Smithfield	Programmer	\$35.75	12.6
■			102	David H. Senior	Systems Analyst	\$96.75	23.8

Рис. 4.2. Данные по проекту Evergreen

В реляционной таблице не должно быть повторяющихся групп. Если же такие группы существуют, то от них необходимо избавиться с тем, чтобы каждая строка определяла единственное значение сущности. Процедура очень проста: просто добавьте соответствующий компонент, по крайней мере, в столбец (или столбцы) первичного ключа. Это означает, что табличные данные, представленные на рис. 4.1 и 4.2, должны быть преобразованы к виду, изображенному на рис. 4.3. Удалив повторяющиеся группы, мы получаем таблицу, приведенную к первой нормальной форме.

Вид, представленный на рис. 4.3, это более чем просто косметические изменения. Теперь даже неопытный наблюдатель обратит внимание, что атрибут PROJ_NUM

нельзя использовать в качестве первичного ключа, поскольку номер проекта не идентифицирует уникально все остальные атрибуты сущности (строки). Например, значение PROJ_NUM, равное 15, может определять одного из пяти сотрудников. Первичный ключ, уникально определяющий любое значение атрибута, должен быть комбинацией атрибутов PROJ_NUM и EMP_NUM.

	PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
▶	5	Evergreen	103	June E. Arbough	Elect. Engineer	\$84.50	23.8
	15	Evergreen	101	John G. News	Database Designer	\$105.00	19.4
	15	Evergreen	105	Alice K. Johnson *	Database Designer	\$105.00	35.7
	15	Evergreen	106	William Smithfield	Programmer	\$35.75	12.5
	15	Evergreen	102	David H. Senior	Systems Analyst	\$96.75	23.9
	18	Amber Wave	114	Annelise Jones	Applications Designer	\$48.10	24.6
	18	Amber Wave	118	James J. Frommer	General Support	\$18.36	45.3
	18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	\$96.75	32.1
	18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	\$45.95	44.0
	22	Rolling Tide	105	Alice K. Johnson	Database Designer	\$105.00	64.7
	22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	\$96.75	48.9
	22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	\$48.10	23.6
	22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	\$26.87	22.5
	22	Rolling Tide	106	William Smithfield	Programmer	\$35.75	12.1
	25	Starflight	107	Maria D. Alonzo	Programmer	\$35.75	24.7
	25	Starflight	115	Travis B. Bawangi	Systems Analyst	\$96.75	45.8
	25	Starflight	101	John G. News *	Database Designer	\$105.00	56.3
	25	Starflight	114	Annelise Jones	Applications Designer	\$48.10	33.1
	25	Starflight	108	Ralph B. Washington	Systems Analyst	\$96.75	23.9
	25	Starflight	118	James J. Frommer	General Support	\$18.36	30.2
	25	Starflight	112	Darlene M. Smithson	DSS Analyst	\$45.95	41.4

Рис. 4.3. Организация данных: первая нормальная форма (1НФ)

Зависимости можно установить с помощью *диаграммы зависимостей (dependency diagram)*, представленной на рис. 4.4.

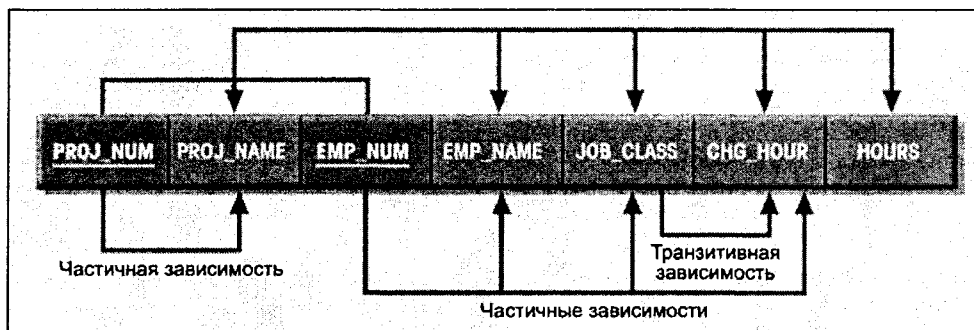


Рис. 4.4. Диаграмма зависимостей: первая нормальная форма (1НФ)

На рис. 4.4 необходимо обратить внимание на следующее:

- атрибуты первичного ключа выделены жирным шрифтом и подчеркнуты, а также закрашены другим оттенком;
- линии со стрелками над сущностями указывают все возможные желательные зависимости, т. е. зависимости, основанные на первичном ключе. В данном случае отметим, что атрибуты сущности зависят от комбинации PROJ_NUM и EMP_NUM. Иначе говоря, если вы знаете PROJ_NUM и EMP_NUM для какой-то строки, то вы сможете идентифицировать все остальные значения в этой строке. Например, исходя из рис. 4.3, если вы знаете, что PROJ_NUM = 15, а EMP_NUM = 103, то значения атрибутов PROJ_NAME, EMP_NAME, JOB_CLASS, CHG_HOUR и HOURS будут соответственно равны Evergreen, June E. Arbough, Elect. Engineer, \$84.50 и 23.8;
- линии со стрелками в нижней части диаграммы зависимостей указывают на не-обязательные зависимости. Имеются два типа таких зависимостей:
 - *частичная зависимость (partial dependency)*. Вам необходимо знать только значение PROJ_NUM для определения PROJ_NAME, т. е. PROJ_NAME зависит только от части первичного ключа. А для определения EMP_NAME, JOB_CLASS и CHG_HOUR необходимо знать только EMP_NUM. Зависимость, определяемая только частью составного первичного ключа, называется частичной зависимостью;
 - *транзитивная зависимость (transitive dependency)*. На рис. 4.4 можно заметить, что CHG_HOUR зависит от JOB_CLASS. Поскольку ни CHG_HOUR, ни JOB_CLASS не являются первичными атрибутами — т. е. ни один из этих атрибутов не является, по крайней мере, частью ключа — в этом случае говорят, что имеет место транзитивная зависимость. (Другими словами, транзитивная зависимость это зависимость одного непервичного атрибута от другого непервичного атрибута.) Проблема здесь состоит в том, что такие зависимости тоже могут стать причиной аномалии данных.

Зависимости, представленные на рис. 4.4, можно записать в текстовом формате следующим образом:

PROJ_NUM, EMP_NUM → PROJ_NAME, EMP_NAME, JOB_CLASS,
CHG_HOUR, HOURS

PROJ_NUM → PROJ_NAME

EMP_NUM → EMP_NAME, JOB_CLASS, CHG_HOUR

JOB_CLASS → CHG_HOUR

(Формат текстовой записи был описан в гл. 2.)

Как мы уже отмечали в гл. 3, табличную структуру можно представить в текстовом формате следующим образом:

ИМЯ ТАБЛИЦЫ (АТРИБУТ(Ы)_ПЕРВИЧНОГО_КЛЮЧА,
ЗАВИСИМЫЕ_АТРИБУТЫ)

Например, если предыдущую таблицу назвать CHARGE (расходы), то ее структуру можно представить в таком виде:

CHARGE (PROJ_NUM, EMP_NUM, PROJ_NAME, EMP_NAME, JOB_CLASS,
CHG_HOUR, HOURS)

Обратите внимание, что в таблице CHARGE имеется составной ключ, в который входят атрибуты PROJ_NUM и EMP_NUM. Любой атрибут, который, по меньшей мере, является частью ключа, называется *первичным атрибутом* или *ключевым атрибутом*. Следовательно, и PROJ_NUM, и EMP_NUM являются первичными (ключевыми) атрибутами. Соответственно, непервичный атрибут (или неключевой атрибут) не является частью ключа.

Примечание

Говорят, что таблица приведена к *первой нормальной форме (1NF)*, если в ней:

- определены все ключевые атрибуты;
- отсутствуют повторяющиеся группы. Иначе говоря, на пересечении каждого столбца и каждой строки содержится только одно (атомарное) значение, а не множество значений;
- все атрибуты зависят от первичного ключа.

Все реляционные таблицы удовлетворяют требованиям, предъявляемым к 1NF. Проблема таблицы, приведенной к первой нормальной форме (см. рис. 4.4), состоит, однако, в том, что в ней имеются частичные зависимости — т. е. зависимости, основанные только на части первичного ключа.

Несмотря на то что частичные зависимости иногда используются в целях повышения производительности, все же их нужно использовать чрезвычайно осторожно³. Такое предостережение абсолютно оправданно, поскольку таблица, в которой имеются частичные зависимости, все еще содержит избыточные данные и, следовательно, подвержена различным аномалиям данных. Причиной избыточности данных является тот факт, что в каждой вводимой строке требуется повторный ввод одних и тех же данных. Например, если пользователь в течение дня вводит 20 элементов таблицы для EMP_NUM = 105, то каждый раз он должен вводить значения атрибутов EMP_NAME, JOB_CLASS и CHG_HOUR, несмотря на то, что эти атрибуты для EMP_NUM = 105 одни и те же. Такое дублирование весьма неэффективно. К тому же повторяющиеся действия могут стать причиной аномалии данных, поскольку ничто не мешает оператору набирать немножко отличающиеся в каждой записи имена пользователя, его должности или тарифной ставки. Например, имя сотрудника EMP_NUM = 102 может быть введено как Dave Senior или как D. Senior. Точно так же название проекта может быть введено правильно (Evergreen) или с ошибкой (Evegreen). Такие аномалии нарушают целостность реляционной базы данных и правила непротиворечивости.

4.1.3. Приведение ко второй нормальной форме

К счастью, проект реляционной базы данных можно легко усовершенствовать, приведя таблицы БД к виду, называемому второй нормальной формой (2НФ). Процесс

³ Если сформулированные заказчиком требования к информации вынуждают использовать частичные зависимости, то самое время рассмотреть возможность разработки проекта информационного хранилища (Data Warehouse), особенности которого будут обсуждаться в гл. 13.

преобразования 1НФ в 2НФ очень прост. Взяв за основу форму 1НФ, представленную на рис. 4.4, необходимо проделать следующее:

1. Записать каждый ключевой компонент в отдельной строке, а затем в последней строке записать исходный (составной) ключ:

PROJ_NUM

EMP_NUM

PROJ_NUM EMP_NUM

2. Каждый компонент станет ключом в новой таблице. Иначе говоря, исходную таблицу необходимо разделить на три таблицы. Назовем эти таблицы PROJECT (проект), EMPLOYEE (сотрудник) и ASSIGN (назначение) соответственно.

3. После каждого нового ключа записать зависимые атрибуты. (Для определения зависимости атрибутов используйте рис. 4.4.) Зависимости для компонентов исходного ключа можно выявить, прослеживая линии со стрелками в нижней части диаграммы зависимостей, представленной на рис. 4.4. Другими словами, новые три таблицы PROJECT, EMPLOYEE и ASSIGN могут быть описаны следующим образом:

PROJECT (PROJ_NUM, PROJ_NAME)

EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOUR)

ASSIGN (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)

4. Поскольку количество часов, затраченных на каждый проект каждым сотрудником, зависит в таблице ASSIGN как от атрибута PROJ_NUM, так и от атрибута EMP_NUM, в таблицу ASSIGN мы их поместили под названием ASSIGN_HOURS.

Результат этой операции представлен на рис. 4.5. Теперь большинство аномалий, которые мы ранее обсуждали, устранено. Например, если мы захотим добавить/изменить/удалить запись в проекте PROJECT, то необходимо лишь обратиться к таблице PROJECT и добавить/изменить/удалить только одну строку.

Примечание

Говорят, что таблица приведена ко *второй нормальной форме (2НФ)*, если:

- она приведена к первой нормальной форме;
- в ней нет частичных зависимостей, т. е. нет атрибутов, зависящих от части первичного ключа.

(Но, тем не менее, в таблицах 2НФ может иметь место транзитивная зависимость, т. е. один или более атрибутов могут функционально зависеть от неключевых атрибутов).

Поскольку частичная зависимость может иметь место только в том случае, если первичный ключ таблицы состоит из нескольких атрибутов, таблицы, в которых первичный ключ содержит всего лишь один атрибут, автоматически будут иметь 2НФ, если они имеют 1НФ.

На рис. 4.5 все еще видна транзитивная зависимость, которая может стать причиной аномалий. Например, если стоимость часа работы для данной специализации, кото-

рую имеют несколько сотрудников, измененится, то это изменение необходимо будет проделать для *каждого* такого сотрудника. Если вы забудете обновить какие-то записи, связанные с изменением стоимости часа, то разные сотрудники, имеющие одну и ту же специализацию, получат разную зарплату.

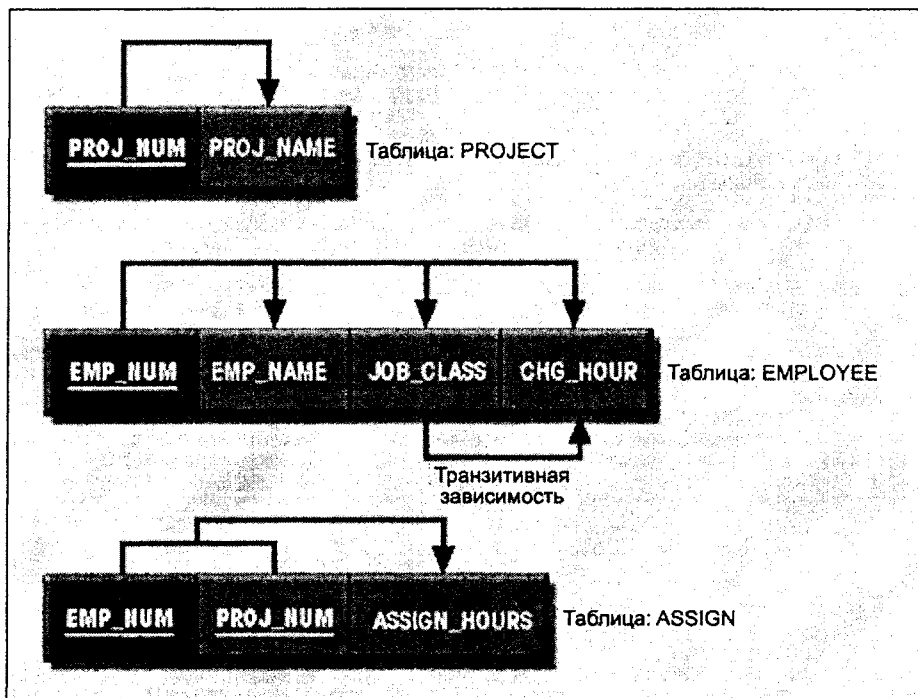


Рис. 4.5. Результат приведения ко второй нормальной форме (2НФ)

4.1.4. Преиведение к третьей нормальной форме

Аномалии данных, созданные в базе данных, представленной на рис. 4.5, легко можно устранить, разбив фрагменты или фрагмент, на который указывает линия со стрелкой транзитивной зависимости (в нижней части диаграммы зависимостей) и поместив их в отдельную таблицу. Однако атрибут `JOB_CLASS` должен остаться в исходной таблице (имеющей 2НФ) в качестве внешнего ключа для того чтобы, установить связь между исходной таблицей и вновь созданной. Иначе говоря, после завершения преобразования в нашей базе данных должно быть четыре таблицы:

PROJECT (`PROJ_NUM`, `PROJ_NAME`)

ASSIGN (`PROJ_NUM`, `EMP_NUM`, `ASSIGN_HOURS`)

EMPLOYEE (`EMP_NUM`, `EMP_NAME`, `JOB_CLASS`)

JOB (`JOB_CLASS`, `CHG_HOUR`)

Отметим, что выполнив это преобразование, мы устранили транзитивную зависимость оригинальной таблицы EMPLOYEE, и теперь таблица приведена к третьей нормальной форме (3НФ).

Примечание

Говорят, что таблица приведена к *третьей нормальной форме (3НФ)*, если:

- она приведена ко второй нормальной форме (2НФ);
- в ней отсутствуют транзитивные зависимости.

После того как мы навели порядок в структурах таблиц и устранили проблемы, вызванные частичной и транзитивной зависимостями, можно подумать о том, как улучшить возможности манипулирования данными и эксплуатационные характеристики нашей БД. Необходимо выполнить следующие действия.

- ❑ По мере приема новых сотрудников значение атрибута JOB_CLASS нужно вводить всякий раз, когда данные о новом служащем вводятся в таблицу EMPLOYEE. К сожалению, при этом могут возникать ошибки ввода данных, которые приведут к нарушению ссылочной целостности. Например, если в таблице EMPLOYEE ввести для атрибута JOB_CLASS значение DB Designer (проектировщик БД) вместо Database Designer (проектировщик баз данных), то как раз и возникнет такое нарушение, поскольку в таблице JOB нет записей для DB Designer. Добавив атрибут JOB_CODE, мы создадим такую зависимость:

JOB_CODE → JOB_CLASS, CHG_HR

Новый атрибут приводит к транзитивной зависимости (если вы считаете, что JOB_CODE является подходящим первичным ключом), поскольку при этом получается, что:

JOB_CLASS → CHG_HR

а это и есть транзитивная зависимость, поскольку неключевой атрибут JOB_CLASS определяет значение другого неключевого атрибута CHG_HR. Однако такая зависимость для нас не очень опасна, поскольку наличие атрибута JOB_CODE значительно уменьшает вероятность нарушения ссылочной целостности.

- ❑ Лучше все же придерживаться принятого соглашения об именовании. Следовательно, мы должны переименовать атрибут CHG_HOUR в JOB_CHG_HOUR, чтобы подчеркнуть его связь с таблицей JOB. Кроме того, имя атрибута JOB_CLASS не вполне точно описывает такие элементы, как System Analyst (системный аналитик), Database Designer и т. д.; для этого атрибута лучше подойдет название JOB_DESCRIPTION (описание рода деятельности). Точно так же использование названия ASSIGNE_HOURS вместо HOURS (часы) позволит нам увязать отработанные часы с таблицей ASSIGN (назначения).
- ❑ Вообще-то практику следования условию *атомарности* атрибутов — т. е. использования только таких атрибутов, которые невозможно разбить на составляющие, — можно считать очень полезной. Очевидно, атрибут EMP_NAME таблицы EMPLOYEE не является атомарным, поскольку его можно разделить на фамилию, имя и отчество (инициал в английском варианте). Разбивая атрибут на более мелкие, мы тем самым увеличиваем гибкость модели. Например, если мы

введем атрибуты EMP_LNAME (фамилия), EMP_FNAME (имя) и EMP_INITIAL (инициал или отчество), то нам легче будет создать список телефонов, отсортированный по фамилии, имени и отчеству (инициалу). Такая задача практически невыполнима, если все компоненты имени хранятся внутри одного атрибута.

- ❑ Если таблица EMPLOYEE используется в реальной конфигурации, в нее желательно добавить еще несколько атрибутов. Например, общий размер выплаченной заработной платы на сегодняшний день, платежи в фонд социального страхования, платежи по медицинскому страхованию. В нашем примере необходимо добавить дату приема сотрудника на работу. Атрибут EMP_HIREDAY (дата приема на работу) впоследствии может быть использован для отслеживания стажа работы сотрудника, предоставления привилегий сотрудникам, имеющим большой стаж работы на предприятии, и других моральных поощрений.
- ❑ Возможность получения подробных сведений о каждом руководителе проекта гарантируется в системе тем, что атрибут EMP_NUM используется в качестве внешнего ключа в таблице PROJECT. Это обеспечивает нам доступ к сведениям о каждом руководителе проекта без ненужного дублирования данных.
- ❑ Хотя на первый взгляд первичным ключом может служить комбинация атрибутов EMP_NUM и PROJ_NUM, использование в этом качестве кода операции назначения сотрудника ASSIGN_NUM предпочтительнее и обеспечивает большую гибкость. Например, если первичным ключом будет комбинация атрибутов EMP_NUM и PROJ_NUM и сотрудник дважды занесет "количество отработанных часов" в таблицу ASSIGN, то это приведет к нарушению целостности на уровне сущности. Даже если мы добавим в составной первичный ключ атрибут ASSIGN_DATE, целостность на уровне сущности все же будет нарушена, когда какой-либо сотрудник сделает одну или более записей для одного и того же проекта в течение одного дня. (Сотрудник может работать над проектом в течение нескольких часов утром, а затем снова работать над этим же проектом в тот же день в более позднее время.) А вот если в качестве первичного ключа использовать атрибут ASSIGN_NUM, то такие действия не вызовут нарушений целостности, особенно, если прикладное программное обеспечение разработано таким образом, что новое значение ASSIGN_NUM генерируется автоматически каждый раз при выполнении операции назначения сотрудника на проект. Однако оставить атрибуты EMP_NUM и PROJ_NUM в качестве внешних ключей необходимо, поскольку эти атрибуты реализуют связи между таблицами ASSIGN, EMPLOYEE и PROJECT. Таким образом, мы сможем отслеживать сведения обо всех работах, выполняя суммирование значений атрибута ASSIGN_HOUR по проекту, по сотрудникам, по дате и — используя связь между таблицами EMPLOYEE и JOB — по роду деятельности.
- ❑ Размещение стоимости часа работы в таблице ASSIGN является ключевым моментом для обеспечения ретроспективной достоверности данных в таблице ASSIGN. Неплохо было бы назвать этот атрибут ASSIGN_CHG_HOUR. Казалось бы, этот новый атрибут имеет тот же смысл, что и JOB_CHG_HOUR, но это справедливо лишь в том случае, если атрибут JOB_CHG_HOUR всегда остается постоянным. Однако разумнее предположить, что почасовой тариф может со временем изменяться. Что произойдет, если мы будем рассчитывать затраты на проект, умножая отработанные часы в таблице ASSIGN на почасовой тариф, который извлекается из таблицы JOB? Эти затраты всегда основаны на текущем

значении почасового тарифа, хранящемся в таблице JOB, вместо того чтобы использовать значение почасового тарифа, действующего на момент назначения сотрудника на данный проект (см. гл. 2, где подробно рассматривалась проблема ретроспективной достоверности в базе данных).

- Наконец, для хранения фактических затрат на проект мы можем использовать производный атрибут в таблице ASSIGN. Этот производный атрибут с именем ASSIGN_CHARGE представляет собой результат перемножения значений атрибутов ASSIGN_HOURS и ASSIGN_CHG_HOUR. Строго говоря, такой атрибут может быть рассчитан непосредственно при выполнении отчета или выписке счета. Однако хранение производного атрибута в таблице упрощает разработку прикладных программ. К тому же если имеется много транзакций, которые должны найти отражение в отчетах, наличие производного атрибута сэкономит время на выполнение отчета. (Если расчет выполняется во время ввода данных, он будет выполнен сразу после нажатия конечным пользователем клавиши <Enter>, что сэкономит время.)

Таблицы и диаграммы зависимостей после внесения всех этих улучшений представлены на рис. 4.6.

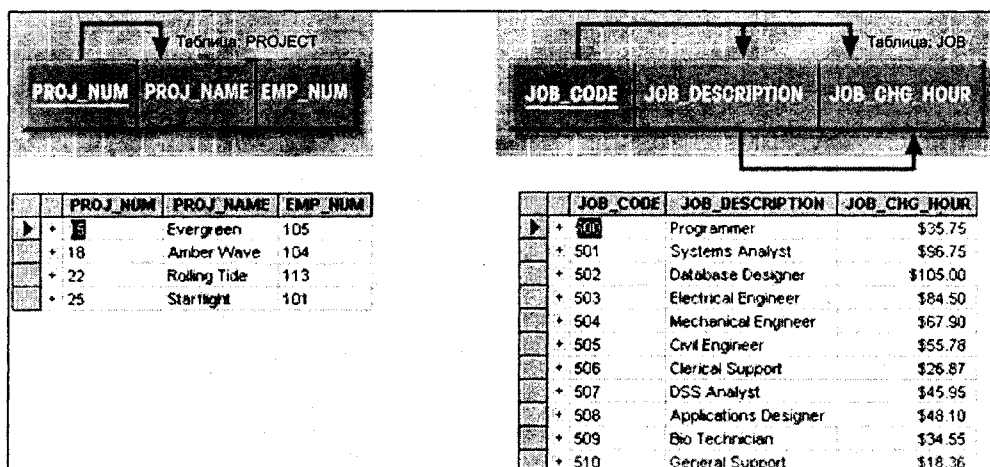


Рис. 4.6. Окончательный вид базы данных

После всех преобразований база данных на рис. 4.6 содержит три таблицы в третьей нормальной форме (3НФ) (PROJECT, ASSIGN и EMPLOYEE) и одну таблицу во второй нормальной форме (2НФ). На рис. 4.6 представлены многочисленные доработки, сделанные в исходном проекте БД. Отметим, что в этой базе данных отсутствуют аномалии данных. Поскольку для каждого атрибута JOB_CODE имеются единственные (и уникальные) значения атрибутов JOB_DESCRIPTION и JOB_CHG_HOUR, нет вероятности использовать разные значения для описания одних и тех же объектов или предметов. Точно так же для каждого атрибута в таблице EMPLOYEE есть только одно значение. Избыточность данных также минимальна.

Таблица: ASSIGN

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-02	15	103	2.6	\$84.50	\$219.70
1002	04-Mar-02	18	118	1.4	\$18.36	\$25.70
1003	05-Mar-02	15	101	3.6	\$105.00	\$378.00
1004	05-Mar-02	22	113	2.5	\$48.10	\$120.25
1005	05-Mar-02	15	103	1.9	\$84.50	\$160.55
1006	05-Mar-02	25	115	4.2	\$96.75	\$408.35
1007	05-Mar-02	22	105	5.2	\$105.00	\$546.00
1008	05-Mar-02	25	101	1.7	\$105.00	\$178.50
1009	05-Mar-02	15	105	2.0	\$105.00	\$210.00
1010	06-Mar-02	15	102	3.8	\$96.75	\$367.65
1011	06-Mar-02	22	104	2.6	\$96.75	\$251.55
1012	06-Mar-02	15	101	2.3	\$105.00	\$241.50
1013	06-Mar-02	25	114	1.8	\$48.10	\$86.58
1014	06-Mar-02	22	111	4.0	\$26.87	\$107.48
1015	06-Mar-02	25	114	3.4	\$48.10	\$163.54
1016	06-Mar-02	18	112	1.2	\$45.95	\$55.14
1017	06-Mar-02	18	119	2.0	\$18.36	\$36.72
1018	06-Mar-02	18	104	2.6	\$96.75	\$251.55
1019	06-Mar-02	15	103	3.0	\$84.50	\$253.50
1020	07-Mar-02	22	105	2.7	\$105.00	\$283.50
1021	08-Mar-02	25	108	4.2	\$96.75	\$406.35
1022	07-Mar-02	25	114	5.8	\$48.10	\$278.98
1023	07-Mar-02	22	106	2.4	\$35.75	\$85.80

Таблица: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
01	News	John	O	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-94	503
104	Ramoras	Anne	K	15-Nov-85	501
105	Johnson	Alice	K	01-Feb-91	502
106	Smithfield	William		22-Jun-01	500
107	Alonzo	Maria	D	10-Oct-91	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-95	501
110	Olenko	Gerald	A	11-Dec-93	505
111	Webash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-92	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Arnelise		20-Aug-91	509
115	Bawangi	Treviz	B	25-Jan-90	501
116	Proff	Gerald	L	05-Mar-95	510
117	Williamson	Anjie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-92	510

Рис. 4.6. Окончательный вид базы данных (окончание)

Если прикладная программа разработана надлежащим образом, то наиболее активная таблица (ASSIGN) требует занесения в нее только значений PROJ_NUM, EMP_NUM и ASSIGN_HOURS. Значения атрибутов ASSIGN_NUM и ASSIGN_DATE должны генерироваться приложением. Например, атрибут

ASSIGN_NUM может быть создан с помощью счетчика, а в атрибут ASSIGN_DATE можно автоматически заносить текущую системную дату. Кроме того, прикладная программа может автоматически подставлять корректное значение атрибута ASSIGN_CHG_HOUR, извлекая его из атрибута JOB_CHG_HOUR таблицы JOB. (Таблицы JOB и ASSIGN связаны через JOB_CODE!) Если значение атрибута JOB_CHG_HOUR в таблице JOB изменяется, то при следующем занесении данных в таблицу ASSIGN эти изменения будут отражены автоматически. Структура таблиц, таким образом, минимизирует необходимость человеческого участия при вводе данных. На самом деле, если в системе требуется, чтобы сотрудник сам вводил свои отработанные часы по проекту, то значение атрибута EMP_NUM может сканироваться в таблицу ASSIGN с помощью считывающего устройства с идентификационной магнитной карты сотрудника. Тем самым структура таблицы ASSIGN обеспечивает еще и некоторый уровень безопасности.

Ограничения на ключи, назначаемые системой

Несмотря на то что наш проект соответствует реальному объекту и установленным требованиям, все же остаются еще некоторые вопросы к проектировщику. Вспомните, что мы создали атрибут JOB_CODE и сделали его первичным ключом таблицы JOB для обеспечения сущностной целостности в этой таблице. После этого можно использовать средства СУБД для установки значений первичного ключа системой. Однако надо понимать, что JOB_CODE не предохранит нас от такого рода записей в таблице JOB:

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
511	Programmer	\$35.75
512	Programmer	\$35.75

Очевидно, что такие записи неприемлемы, поскольку они дублируют друг друга (несмотря на то что они не нарушают ни сущностной, ни ссылочной целостности). Проблема "многократного дублирования записей" возникла после добавления атрибута JOB_CODE в качестве первичного ключа. (Первоначально, когда в качестве первичного ключа использовался атрибут JOB_DESCRIPTION, средства СУБД гарантировали уникальность значений всех записей рода деятельности, если требовалось обеспечить сущностную целостность. Но это создавало проблемы, которые заставили нас ввести атрибут JOB_CODE.) Как бы то ни было, даже если ключом является атрибут JOB_CODE, мы все равно должны обеспечить единственность значений JOB_DESCRIPTION посредством использования уникального индекса.

Такая же проблема есть и в таблице EMPLOYEE. Используя в качестве первичного ключа атрибут EMP_NUM, мы можем сделать несколько записей для одного и того же сотрудника. Чтобы избежать этого, мы должны создать уникальный индекс для EMP_LNAME, EMP_FNAME и EMP_INITIAL. Но как быть с двумя сотрудниками, имеющими одинаковые имена Joe B. Smith? В подобном случае нужно использовать другой (предпочтительнее какой-то внешний) атрибут, например, номер социального страхования (social security number), который может служить основой для уникального индекса.

Наверное, имеет смысл повторить, что при проектировании баз данных зачастую требуется идти на компромиссы и полагаться на профессиональную интуицию.

В реальных условиях мы должны находить баланс между работоспособностью и гибкостью проекта. Например, если необходимо ограничиться только одной записью в атрибут ASSIGN_HOURS (время работы над проектом) в день для одного сотрудника, то придется спроектировать таблицу ASSIGN таким образом, чтобы в атрибутах PROJ_NUM, EMP_NUM и ASSIGN_DATE использовался уникальный индекс. Такое ограничение гарантирует, чтобы сотрудники не смогли вводить одно и то же количество часов несколько раз в день. К сожалению, вероятно, это ограничение будет нежелательным с административной точки зрения. Все-таки если сотрудник работал над данным проектом несколько раз за день и в разное время, то необходимо иметь возможность делать несколько записей для данного сотрудника в один и тот же проект в течение данного дня. В этом случае лучшим решением может стать ввод какого-нибудь внешнего атрибута (например, номера контрольного талона, номер билета и т. п.) для обеспечения уникальности. В любом случае необходим частый аудит данных.

4.1.5. Нормальная форма Бойса—Кодда (БКНФ)

Таблица приведена к *нормальной форме Бойса—Кодда (БКНФ)*, если каждый детерминант таблицы является потенциальным ключом. (*Детерминантом* называется любой атрибут, значения которого определяют другие значения внутри данной строки.) Очевидно, если в таблице имеется только один потенциальный ключ, то формы 3НФ и БКНФ эквивалентны. Рассуждая иначе, можно сказать, что форма БКНФ может быть нарушена, только в том случае, если в таблице имеется более одного потенциального ключа.

Большинство проектировщиков рассматривают нормальную форму Бойса—Кодда (БКНФ) как особый случай 3НФ. На самом деле, если вы используете представленную нами ранее технологию, то большинство таблиц соответствуют требованиям БКНФ, если они соответствуют требованиям 3НФ. Так как же таблица, приведенная к нормальной форме 3НФ, может не быть приведенной к нормальной форме БКНФ? Чтобы ответить на этот вопрос, необходимо вспомнить, что транзитивная зависимость имеет место, когда один первичный атрибут зависит от другого первичного атрибута.

Иначе говоря, таблица приведена к 3НФ, если она приведена к 2НФ, и в ней отсутствуют транзитивные зависимости. А что произойдет в том случае, если неключевой атрибут является детерминантом ключевого атрибута? Такое обстоятельство не нарушает условий 3НФ, но не отвечает правилам БКНФ, поскольку БКНФ требует, чтобы каждый детерминант в таблице был потенциальным ключом.

Ситуацию, которую мы только что описали (таблица, приведенная к 3НФ, не соответствует требованиям БКНФ), проще объяснить с помощью рис. 4.7.

Обратите внимание на следующие функциональные зависимости (рис. 4.7):

$A + B \rightarrow C, D$

$C \rightarrow B$

В структуре таблицы, представленной на рис. 4.7, нет ни частичных зависимостей, ни транзитивных зависимостей. (Условие $C \rightarrow B$ указывает на то, что неключевой атрибут определяет часть первичного ключа — а эта зависимость не транзитивна!).

Очевидно, что структура таблицы, представленная на рис. 4.7, отвечает требованиям ЗНФ. А вот условие $C \rightarrow B$ нарушает требования БКНФ.

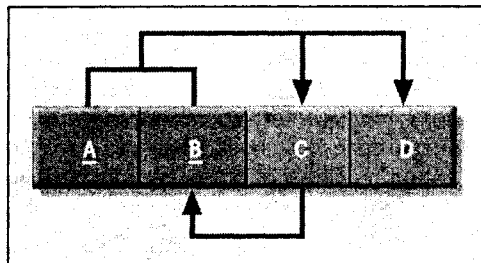


Рис. 4.7. Структура таблицы, приведенной к ЗНФ, но не к БКНФ

Чтобы преобразовать структуру, представленную на рис. 4.7, в таблицу, отвечающую требованиям как ЗНФ, так и БКНФ, прежде всего, необходимо изменить первичный ключ на $A + C$. Это целесообразно сделать, поскольку зависимость $C \rightarrow B$ означает, что C , по сути дела, является подмножеством B . После этого можно считать, что таблица приведена к 2НФ, поскольку в ней имеется частичная зависимость $C \rightarrow B$. Затем, с помощью знакомых процедур декомпозиции, приведем таблицу к виду, представленному на рис. 4.8.

Чтобы посмотреть, как эта процедура применяется на практике, давайте исследуем данные, представленные в табл. 4.2.

Таблица 4.2. Табличная структура для демонстрации использования операции приведения к БКНФ

STU_ID	STAFF_ID	CLASS_CODE	ENROLL_GRADE
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

В табл. 4.2 нашли отражение следующие условия:

- каждый CLASS_CODE (код группы) уникально идентифицирует группу. Это условие иллюстрирует пример, который мы использовали в гл. 2 и 3, когда на курсе могло создаваться несколько групп. Например, курс с названием ИФФС 420 может читаться в двух группах, каждая из которых идентифицируется уникальным кодом для упрощения регистрации студентов. Таким образом, CLASS_CODE 32456 определяет группу 1 курса ИФФС 420, в то время как CLASS_CODE 32457 идентифицирует группу 2 курса ИФФС 420, а CLASS_CODE 28458 определяет группу 4 курса QM 362;

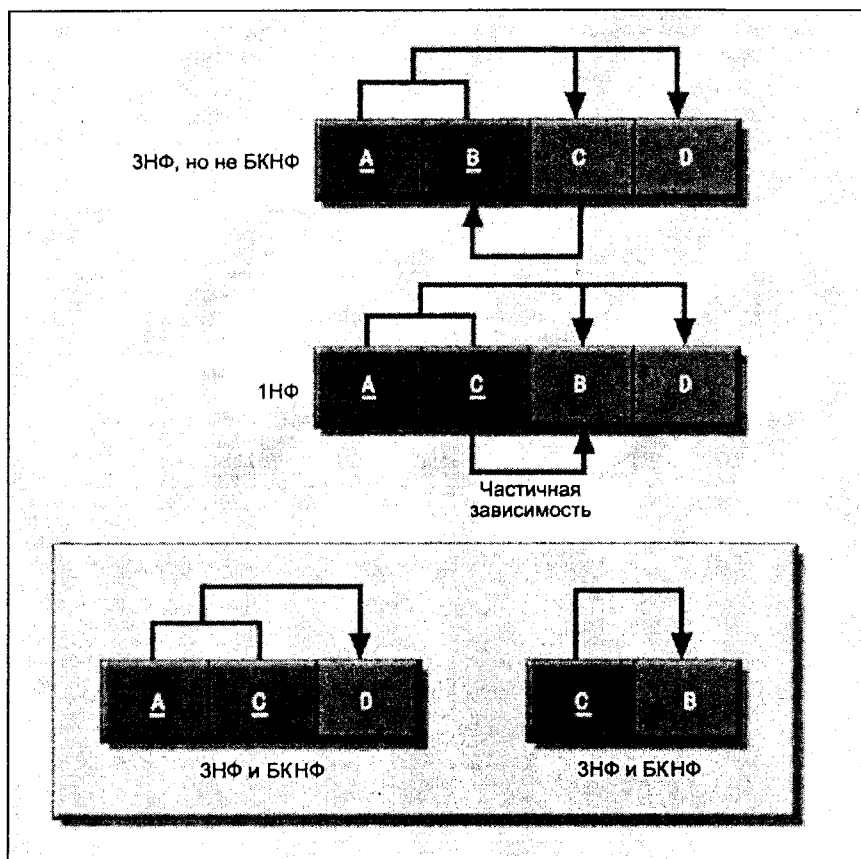


Рис. 4.8. Декомпозиция структуры таблицы для выполнения требований БКНФ

- ❑ студенты могут заниматься в нескольких группах. Обратите внимание, например, что студент 125 посещает группы с кодами 21334 и 32456, имея по ним оценки А и С соответственно;
- ❑ штатный преподаватель может вести занятия в нескольких группах, но каждую группу ведет только один преподаватель. Отметим, что преподаватель с кодом 20 ведет занятия в группах с кодами 32456 и 28458.

Структура, представленная в табл. 4.2, отражена на панели А рис. 4.9:

$STU_ID + STAFF_ID \rightarrow CLASS_CODE, ENROLL_GRADE$

$CLASS_CODE \rightarrow STAFF_ID$

На панели А рис. 4.9 показана структура, которая очевидно приведена к 3НФ, но в таблице, представленной этой структурой, имеется явный недостаток, поскольку она пытается описать сразу два предмета, что может стать причиной аномалий. Например, если различные штатные сотрудники будут вести занятия в группе 32456, то придется обновлять две строки, что приводит к аномалии обновления. А если сту-

дент 135 покинет группу 28458, то мы потеряем информацию о том, кто вел занятия в этом классе, что приводит к аномалии удаления. Решение этой проблемы состоит в декомпозиции структуры таблицы в соответствии с ранее описанными процедурами. Отметим, что декомпозиция, результаты которой представлены на панели В рис. 4.9, дает нам две таблицы, которые соответствуют как требованиям ЗНФ, так и требованиям БКНФ.

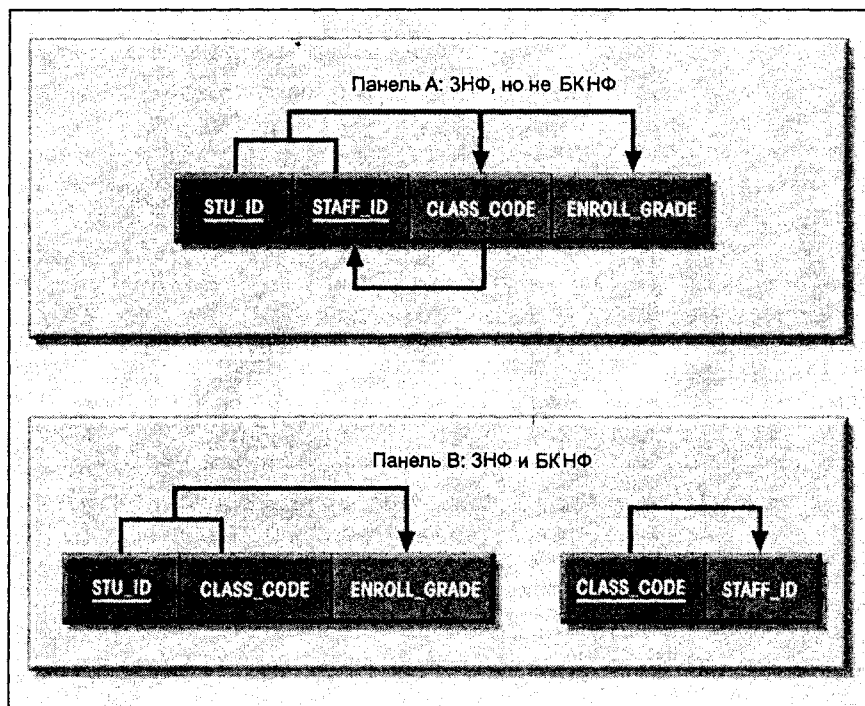


Рис. 4.9. Приведение к БКНФ

Примечание

Таблица приведена к форме БКНФ, если каждый детерминант в этой таблице является потенциальным ключом. Если таблица содержит только один потенциальный ключ, то формы ЗНФ и БКНФ эквивалентны.

4.2. Нормализация и проектирование базы данных

Таблицы, представленные на рис. 4.6, были созданы для демонстрации получения правильных таблиц из таблиц с плохой структурой с помощью процедуры нормализации.

зации. Предполагается, что у вас будет достаточно возможностей применить эти навыки на практике, когда вы приступите к реальному проектированию БД. Нормализация должна быть частью процесса проектирования. Поэтому убедитесь, что выявленные сущности соответствуют нормальной форме *перед* созданием структуры таблиц. Имейте в виду, если вы будете следовать процедурам проектирования, которые мы обсуждали в гл. 2 и 3, то вероятность аномалий будет очень невелика. (Но даже лучшие из нас, к сожалению, не застрахованы от случайных ошибок проектирования, выявляемых при нормализации.) Однако многие базы данных, с которыми вы встретитесь в жизни, будут или спроектированы не совсем корректно, или перегружены аномалиями, являющимися следствием неправильной модификации. При этом вам, возможно, будет предложено перепроектировать и модифицировать такие БД. Следовательно, вы должны быть знакомы как с принципами и методами квалифицированного проектирования, так и с процедурами нормализации.

ER-диаграмма, создание которой является итерационным процессом, обеспечивает макропредставление об информационных потребностях предприятия и его деятельности. Начинайте с определения значимых сущностей, их атрибутов и связей. Затем используйте эти результаты при выявлении дополнительных сущностей и атрибутов.

Процедура нормализации направлена на выявление свойств отдельных сущностей, т. е. нормализация обеспечивает представление о сущностях внутри ER-диаграммы на микроуровне. И как было показано в предыдущих разделах этой главы, процесс нормализации может привести к появлению новых сущностей и атрибутов, которые придется ввести в ER-диаграмму. Поэтому трудно отделить процесс нормализации от ER-моделирования — обе эти технологии должны использоваться совместно.

Чтобы продемонстрировать благотворную роль нормализации в процессе проектирования, вновь исследуем деятельность компании-подрядчика, таблицы которой мы нормализовали в предыдущих разделах. Деятельность компании определяется следующими бизнес-правилами:

- ☐ компания выполняет несколько проектов;
- ☐ в каждом проекте задействовано несколько сотрудников;
- ☐ сотрудник может работать над несколькими проектами;
- ☐ некоторые сотрудники не имеют отношения к конкретному проекту и выполняют текущую работу, не связанную с проектом. Некоторые сотрудники образуют подразделения, которые участвуют в нескольких проектах. Например, исполнительный секретариат компании не принадлежит какому-нибудь отдельному проекту;
- ☐ каждый сотрудник имеет единственную основную специальность. Для каждой специальности определяется стоимость часа работы;
- ☐ многие сотрудники могут иметь одну и ту же специальность. Например, компания может принять на работу несколько инженеров-электриков.

Используя это упрощенное описание деятельности компании, изначально можно определить две сущности вместе с соответствующими атрибутами:

PROJECT (PROJ_NUM, PROJ_NAME)

EMPLOYEE (EMP_NUM, EMP_LNAME, EMP_FNAME, EMP_INITIAL,
JOB_DESCRIPTION, JOB_CHG_HOUR)

Эти две сущности входят в начальную ER-диаграмму, представленную на рис. 4.10.

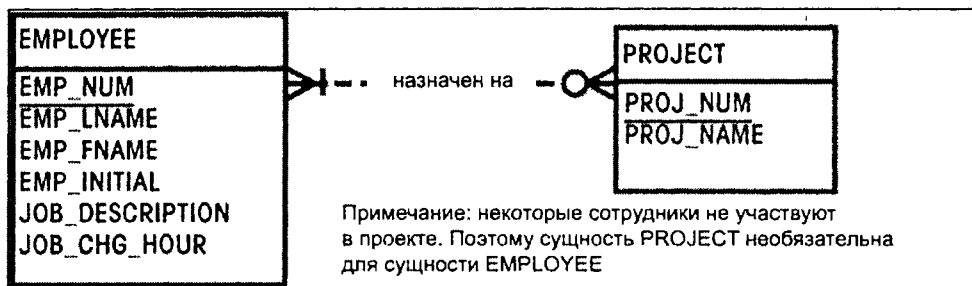


Рис. 4.10. Начальная ER-диаграмма для компании-подрядчика

После создания начальной ER-диаграммы, представленной на рис. 4.10, уровень нормализации выглядит следующим образом:

- ☐ таблица PROJECT приведена к 3НФ и модифицировать ее пока нет необходимости;
- ☐ на таблицу EMPLOYEE следует обратить внимание. Атрибут JOB_DESCRIPTION определяет специальность (например, System Analyst, Database Designer или Programmer). В свою очередь эти специальности определяют тарифную сетку (JOB_CHG_HOUR). Поэтому в таблице EMPLOYEE имеет место транзитивная зависимость.

Устранив транзитивную зависимость в таблице EMPLOYEE, в результате мы получим три сущности:

PROJECT (PROJ_NUM, PROJ_NAME);

EMPLOYEE (EMP_NUM, EMP_LNAME, EMP_FNAME, EMP_INITIAL, JOB_CODE);

JOB (JOB_CODE, JOB_DESCRIPTION, JOB_CHG_HOUR).

Поскольку в процессе нормализации мы ввели новую сущность (JOB), ER-диаграмму необходимо модифицировать (рис. 4.11).

Поскольку связь типа M:N между сущностями EMPLOYEE и PROJECT реализовать невозможно, на ER-диаграмме (см. рис. 4.11) необходимо ввести промежуточную (составную) сущность, и тогда ER-диаграмма будет выглядеть так, как это представлено на рис. 4.12. Составная сущность с именем ASSIGN наследует ключи от сущностей PROJECT и EMPLOYEE (если вы не помните, как создается составная сущность и для чего она нужна, обратитесь к гл. 3).

Из рис. 4.12 следует, что атрибут ASSIGN_HOURS входит в составную сущность с именем ASSIGN. Поскольку скорее всего нам потребуются сведения о каждом руководителе проекта, желательно создать связь "руководит". Эта связь реализуется с помощью внешнего ключа PROJECT. Наконец, с целью повышения возможностей системы по извлечению дополнительной информации могут создаваться некоторые дополнительные атрибуты. Например, можно включить дату приема сотрудника на работу (EMP_HIREDATE), чтобы отслеживать стаж его работы на предприятии.

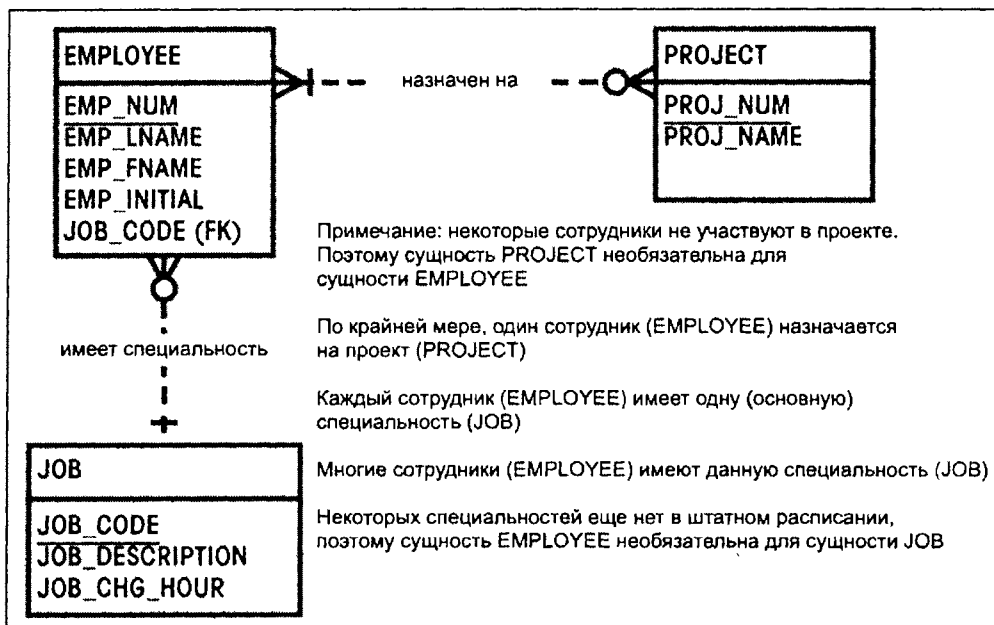


Рис. 4.11. Модификация ER-диаграммы для компании-подрядчика

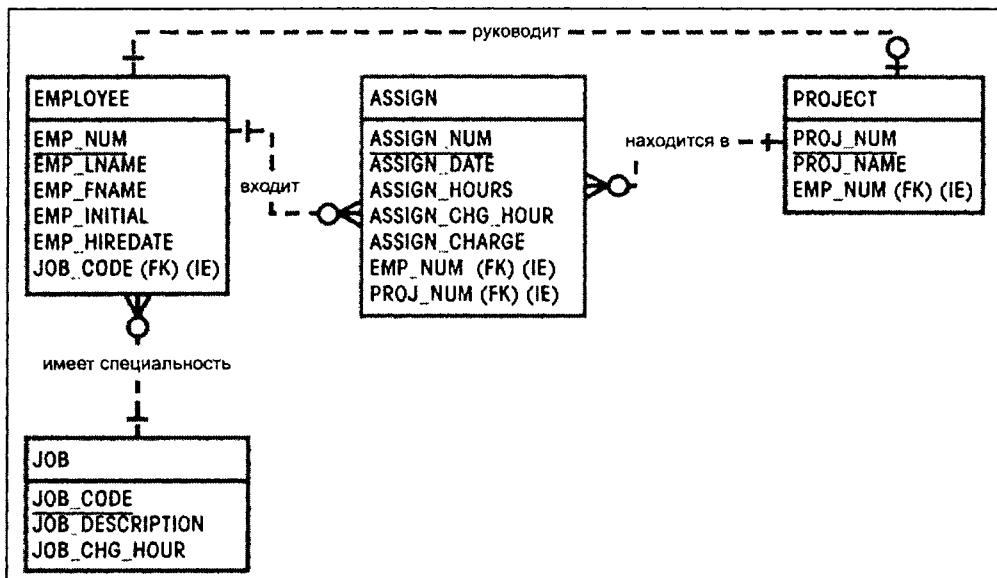


Рис. 4.12. Окончательный вид ER-диаграммы для компании-подрядчика

Таблица: EMPLOYEE						База данных: CONSTRUCTION_CO	
EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	Таблица: JOB	
101	News	John	G	08-Nov-98	502	JOB_CODE	JOB_DESCRIPTION
102	Senior	David	H	12-Jul-87	501		JOB_CHG_HOUR
103	Arbough	June	E	01-Dec-94	503		
104	Ramoras	Anne	K				
105	Johnson	Alice	K				
106	Smithfield	William					
107	Alonzo	Maria	D				
108	Washington	Ralph	B				
109	Smith	Larry	W				
110	Olenko	Gerald	A				
111	Webash	Geoff	B				
112	Smithson	Dariene	M				
113	Joebrood	Delbert	K				
114	Jones	Annelise					
115	Bawangi	Travis	B				
116	Pratt	Gerald	L	05-Mar-95	510		
117	Williamson	Angie	H	19-Jun-94	509		
118	Frommer	James	J	04-Jan-02	510		

PROJ_NUM	PROJ_NAME	EMP_NUM
8	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-02	15	103	2.6	\$84.50	\$219.70
1002	04-Mar-02	18	118	1.4	\$18.36	\$25.70
1003	05-Mar-02	15	101	3.6	\$105.00	\$378.00
1004	05-Mar-02	22	113	2.5	\$48.10	\$120.25
1005	05-Mar-02	15	103	1.9	\$84.50	\$160.55
1006	05-Mar-02	25	115	4.2	\$96.75	\$406.35
1007	05-Mar-02	22	105	5.2	\$105.00	\$546.00
1008	05-Mar-02	25	101	1.7	\$105.00	\$178.50
1009	05-Mar-02	15	105	2.0	\$105.00	\$210.00
1010	06-Mar-02	15	102	3.8	\$96.75	\$367.65
1011	06-Mar-02	22	104	2.6	\$96.75	\$251.55
1012	06-Mar-02	15	101	2.3	\$105.00	\$241.50
1013	06-Mar-02	25	114	1.8	\$48.10	\$86.58
1014	06-Mar-02	22	111	4.0	\$26.87	\$107.48
1015	06-Mar-02	25	114	3.4	\$48.10	\$163.54
1016	06-Mar-02	18	112	1.2	\$45.95	\$55.14
1017	06-Mar-02	18	118	2.0	\$18.36	\$36.72
1018	06-Mar-02	18	104	2.6	\$96.75	\$251.55
1019	06-Mar-02	15	103	3.0	\$84.50	\$253.50
1020	07-Mar-02	22	105	2.7	\$105.00	\$283.50
1021	08-Mar-02	25	108	4.2	\$96.75	\$406.35
1022	07-Mar-02	25	114	5.8	\$48.10	\$278.98
1023	07-Mar-02	22	106	2.4	\$35.75	\$85.80

Рис. 4.13. Реализация базы данных для предприятия-подрядчика

С учетом этих дополнений модель будет содержать четыре сущности вместе с их атрибутами:

PROJECT (PROJ_NUM, PROJ_NAME, EMP_NUM)

EMPLOYEE (EMP_NUM, EMP_LNAME, EMP_FNAME, EMP_INITIAL,
EMP_HIREDATE, JOB_CODE)

JOB (JOB_CODE, JOB_DESCRIPTION, JOB_CHG_HOUR)

ASSIGN (ASSIGN_NUM, ASSIGN_DATE, ASSIGN_HOURS,

ASSIGN_CHG_HOUR, ASSIGN_CHARGE, EMP_NUM, PROJ_NUM)

Процесс проектирования теперь находится на правильном пути. В ER-диаграмме точно представлена деятельность предприятия, а сущности соответствуют требованиям ЗНФ. Сочетание ER-моделирования и нормализации позволяет получить ER-диаграмму, на основе которой сущности могут быть трансформированы в соответствующие табличные структуры. Обратите внимание, что на рис. 4.12 сущность PROJECT необязательна для сущности EMPLOYEE в связи "руководит". Эта необязательность связана с тем, что не все сотрудники руководят проектами. Окончательный вид базы данных представлен на рис. 4.13.

4.3. Нормальные формы более высокого уровня

Вы можете натолкнуться на плохо спроектированные БД, или вам могут предложить конвертировать в формат базы данных большие таблицы, в которых имеется множество многозначных атрибутов. Рассмотрим случай, когда сотрудник работает в нескольких проектах, а также привлекается к работе в различные государственные учреждения.

Например, служащий с идентификационным номером 10123 сотрудничает в Красном Кресте и в благотворительной организации United Way. Кроме того, этот же сотрудник принимает участие в работе над тремя проектами 1, 5 и 12. На рис. 4.14 показано, как эти факты можно записать разными способами.

Такое представление данных создает некоторые проблемы. Каждый из атрибутов EMP_SERVICE и EMP_ASSIGN может иметь различные значения. То есть таблицы содержат два набора многозначных зависимостей (один сотрудник может работать на нескольких государственных службах и принимать участие в разработке нескольких проектов). Представленные на рис. 4.14 варианты многозначных зависимостей означают следующее: если реализованы версии 1 или 2, то в таблицах появятся пустые места (null) и, кроме того, в этом случае отсутствуют даже потенциальные ключи. (EMP_NUM не является уникальным атрибутом, поэтому он не может быть первичным ключом. Ни одна из комбинаций ключей в версиях 1 и 2 не может быть использована в качестве первичного ключа, поскольку в некоторых из них имеются пустые места.) Такое положение дел совершенно неприемлемо, особенно если сотрудников тысячи, и многие из них, будучи занятыми в различных государственных и благотворительных организациях, в то же время работают над различными проек-

тами. Наконец, в версии 3 имеется первичный ключ, но он является комбинацией всех атрибутов таблицы. На самом деле, версия 3 удовлетворяет требованиям 3НФ, но все же содержит большое количество избыточной информации, что, безусловно, нежелательно.

Версия 1 Таблица: FIG4_14A			
	EMP_NUM	EMP_SERVICE	EMP_ASSIGN
▶	10123	Red Cross	1
	10123	United Way	5
	10123		12
Версия 2 Таблица: FIG4_14B			
	EMP_NUM	EMP_SERVICE	EMP_ASSIGN
▶	10123	Red Cross	
	10123	United Way	
	10123		1
	10123		5
	10223		12
Версия 3 Таблица: FIG4_14C			
	EMP_NUM	EMP_SERVICE	EMP_ASSIGN
▶	10123	Red Cross	1
	10123	Red Cross	5
	10123	United Way	12

Рис. 4.14. Таблицы с многозначными зависимостями

Решение состоит в устранении проблемы, вызванной многозначными зависимостями. Это можно сделать, создав две таблицы, представленные на рис. 4.15, на котором видно, что ни одна из таблиц не содержит многозначных зависимостей. Говорят, что такая таблица приведена к четвертой нормальной форме, 4НФ (Fourth Normal Form — 4NF).

Примечание

Говорят, что таблица приведена к *четвертой нормальной форме (4НФ)*, если:

- она приведена к 3НФ;
- в ней отсутствуют многозначные зависимости.

Версия 1 Таблица: FIG4_15A		
	EMP_NUM	EMP_SERVICE
▶	1023	Red Cross
	1023	United Way

Версия 2 Таблица: FIG4_15B		
	EMP_NUM	EMP_ASSIGN
▶	1023	1
	1023	5
	1023	12

Рис. 4.15. Таблицы в 4НФ

Если вы будете следовать принципам проектирования, которые мы приводим в этой книге, то вы никогда не столкнетесь с только что описанными проблемами. Четвертая нормальная форма (4НФ) будет представлять для вас лишь чисто академический интерес, особенно если вы будете следить за тем, чтобы ваши таблицы соответствовали следующим двум правилам.

- ☐ Все атрибуты должны зависеть от первичного ключа, но не должны зависеть друг от друга.
- ☐ Ни в одной из строк не должно содержаться двух или более многозначных сведений о данной сущности.

Существуют нормальные формы и более высоких уровней. Однако такие нормальные формы, как 5НФ и *нормальная форма доменного ключа (ДКНФ, domain-key normal form)*, вряд ли встретятся в реальных бизнес-конфигурациях и представляют по большей части теоретический интерес. Поскольку книга посвящена практическому приложению технологии проектирования БД, мы не будем рассматривать нормальные формы более высоких порядков.

4.4. Денормализация

Создание нормализованных отношений является важнейшей частью проектирования БД, но все же это только одна из задач. В хорошем проекте БД должны учитываться различные требования к обработке информации. По мере того как таблицы преобразуются в соответствии с требованиями нормализации, их число растет. Большее число таблиц влечет за собой увеличение количества операций обмена данными с диском (операции ввода/вывода) и увеличение времени на обработку логических связей, что приводит к снижению скорости всей системы в целом. Следовательно, возможны обстоятельства, при которых потребуется некоторое снижение уровня нормализации, что позволит увеличить скорость обработки данных.

Помните, что достоинства быстрой обработки могут иметь большее значение, чем недостатки некоторой аномалии данных. Кроме того, некоторые аномалии могут иметь чисто теоретическое значение. Например, должны ли люди беспокоиться о том, что в реальной базе данных ZIP_CODE (почтовый индекс) определяет CITY (город) в таблице CUSTOMER (клиент), где первичным ключом является номер клиента? Практично ли на самом деле создавать отдельную таблицу

ZIP (ZIP_CODE, CITY)

для устранения транзитивной зависимости в таблице CUSTOMER? (Может быть, это и будет необходимо, если ваш бизнес предполагает создание списка почтовых адресатов.) Наш совет прост: при выполнении нормализации необходимо руководствоваться еще и здравым смыслом.

В современных базах данных безукоризненную нормализацию зачастую очень трудно обеспечить. Противоречие между эффективностью проекта, информационными потребностями и скоростью обработки информации чаще всего разрешается поиском компромисса, который, как правило, состоит в некоторой денормализации. Иногда использование нормальной формы более низкого уровня помогает устранить серьезные структурные проблемы. В сущности, после некоторой дискуссии, в таблице JOB мы использовали структуру 2НФ (см. рис. 4.6) для того, чтобы уменьшить возможное нарушение ссылочной целостности. Впоследствии (см. гл. 13) мы покажем, что нормальные формы низких уровней имеют место (а иногда они просто необходимы) в специализированных БД, называемых *информационными хранилищами (data warehouses)*. В таких специализированных БД находят отражение все возрастающие потребности широты охвата и глубины поиска информации, на которых основаны системы поддержки решений. Мы покажем, что в информационных хранилищах, как правило, используются структуры 2НФ в условиях сложной, многоуровневой информационной среды, питающейся от множества источников. Короче говоря, несмотря на то что тщательная и глубокая нормализация очень важна, особенно в так называемых *рабочих (operational)* базах данных, которые мы и исследовали до настоящего времени, нельзя сбрасывать со счетов и вторую нормальную форму (2НФ).

Это не означает, что таблицы рабочих баз данных, в которых содержатся частичные или транзитивные зависимости, можно просто оставить в таком виде. Ненормализованные таблицы в рабочей БД, кроме возможных аномалий данных, имеют и другие недостатки:

- ☐ невысокая эффективность обновления информации, поскольку программы, которые считывают и обновляют данные, работают с таблицами больших размеров;
- ☐ затруднен процесс индексирования. Просто непрактично формировать все индексы для множества атрибутов, расположенных в единственной ненормализованной таблице;
- ☐ ненормализованные таблицы затрудняют стратегии создания представлений.

Помните: хороший проект создается не прикладными программами. Также имейте в виду, что ненормализованные таблицы БД чаще всего являются причиной избы-

точности данных в рабочих БД. Другими словами, денормализацию нужно использовать с большой осторожностью.

Резюме

Нормализация — это технология, применяемая при проектировании таблиц с целью минимизации избыточности информации. Первые три нормальные формы (1НФ, 2НФ и 3НФ) встречаются чаще всего. С точки зрения структуры нормальные формы более высокого уровня предпочтительнее нормальных форм низкого уровня, поскольку более высокая степень нормализации минимизирует избыточность данных. Другими словами, 3НФ лучше, чем 2НФ, которая в свою очередь лучше, чем 1НФ. Почти во всех бизнес-проектах 3НФ считается идеальной нормальной формой (существует также специальная, более строгая 3НФ, называемая нормальной формой Бойса—Кодда, БКНФ).

Таблица приведена к 1НФ, когда определены все ключевые атрибуты и когда все остальные атрибуты зависят от первичного ключа. Однако таблица в первой нормальной форме (1НФ) может все еще содержать как частичные, так и транзитивные зависимости. (При частичной зависимости один из атрибутов функционально зависит от части составного первичного ключа. Транзитивная зависимость имеет место, когда один из неключевых атрибутов зависит от другого неключевого атрибута.) Естественно, в таблице, где первичный ключ состоит из единственного атрибута, частичные зависимости отсутствуют.

Таблица приведена ко второй нормальной форме (2НФ), если она приведена к первой нормальной форме (1НФ) и в ней отсутствуют частичные зависимости. Следовательно, таблица 1НФ автоматически находится в 2НФ, если ее первичный ключ состоит из одного атрибута. Таблица, приведенная ко второй нормальной форме, может содержать транзитивные зависимости.

Таблица приведена к третьей нормальной форме (3НФ), если она приведена к 2НФ и не содержит транзитивных зависимостей. Принимая во внимание такое определение 3НФ, можно сказать, что нормальная форма Бойса—Кодда (БКНФ) есть специальный случай формы 3НФ, в которой все детерминантные ключи являются потенциальными ключами. Естественно, если в таблице имеется единственный потенциальный ключ, то таблица, приведенная к 3НФ, автоматически приведена и к БКНФ.

Таблицу, не соответствующую требованиям 3НФ, можно разбивать на новые таблицы, до тех пор, пока все новые таблицы не будут приведены к 3НФ. Этот процесс проиллюстрирован на рис. 4.16—4.18.

Нормализация является частью процесса проектирования. По мере определения в процессе ER-моделирования сущности и атрибуты подвергаются процессу нормализации и при этом формируются (если это необходимо) новые сущности. Нормализованные сущности вводятся в ER-диаграмму, и итеративный процесс ER-

моделирования продолжается до тех пор, пока все сущности и их атрибуты не будут определены, а все таблицы не будут приведены к 3НФ.

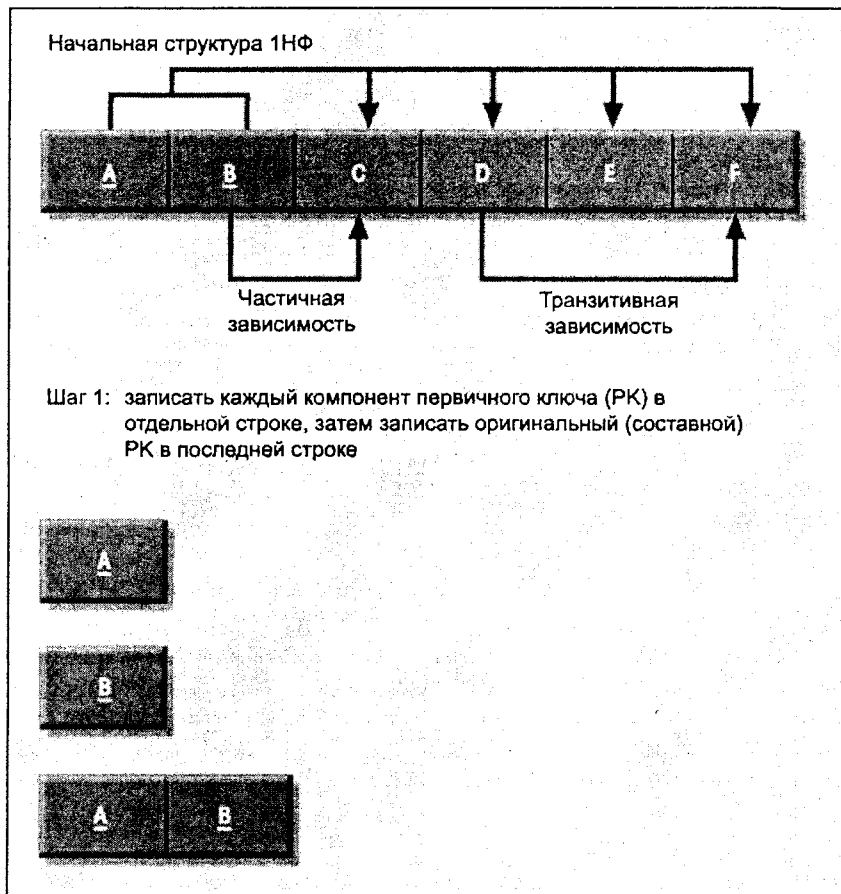


Рис. 4.16. Первоначальная структура 1НФ

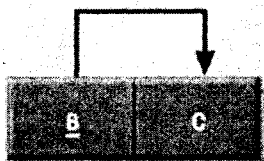
Таблица, приведенная к 3НФ, может содержать многозначные зависимости, которые могут привести либо к большому числу пустых значений, либо к избыточности данных. Поэтому, возможно, придется привести таблицу, приведенную к 3НФ, к четвертой нормальной форме (4НФ), разбивая ее на несколько таблиц с целью устранения многозначных зависимостей. Таким образом, таблица приведена к четвертой нормальной форме (4НФ), если она приведена к третьей нормальной форме (3НФ) и в ней отсутствуют многозначные зависимости.

Большее число таблиц требует большего количества операций обмена с диском и большего времени, затрачиваемого на обработку логики. Поэтому таблицы иногда приходится денормализовать, чтобы уменьшить число операций обмена и увеличить скорость обработки информации. К сожалению, увеличивая таким способом скорость обработки данных, мы снижаем эффективность обновления таблиц (они становятся больше), индексирование усложняется и появляется угроза избыточности данных, которая может стать причиной аномалий данных. Вывод — денормализацию следует применять с осторожностью.

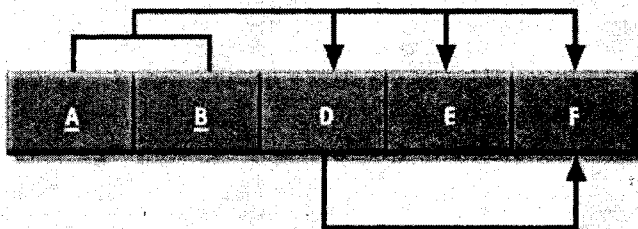
Шаг 2: Расставить все атрибуты, зависящие от первичного ключа (РК), определенного на Шаге 1



Нет атрибутов, зависящих от А. Поэтому А не будет первичным ключом (РК) для новой табличной структуры



Эта таблица приведена к 3НФ (отсутствуют частичные зависимости) и не содержит транзитивных зависимостей



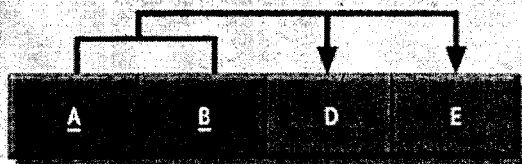
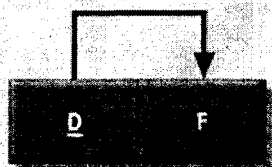
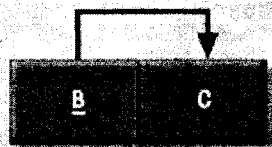
Эта таблица приведена к 2НФ, поскольку в ней имеется транзитивная зависимость

Транзитивная
зависимость

Рис. 4.17. Идентификация возможных атрибутов первичного ключа

Шаг 3: Удалить все транзитивные зависимости, выявленные на Шаге 2, и оставить все структуры 3НФ

Все таблицы приведены к 3НФ, поскольку они приведены к 2НФ (отсутствуют частичные зависимости) и не содержат транзитивных зависимостей



Атрибут D остается в этой табличной структуре в качестве внешнего ключа (FK) для второй таблицы

Рис. 4.18. Структуры таблиц, основанные на выбранных первичных ключах

Основные термины

Атомарность — atomicity

Вторая нормальная форма (2НФ) — Second Normal Form (2NF)

Денормализация — denormalization

Детерминант — determinant

Диаграмма зависимостей — dependency diagram

Ключевой атрибут — key attribute

Неключевой атрибут — nonkey attribute

Непервичный атрибут — nonprime attribute

Нормализация — normalization

Нормальная форма Бойса-Кодда (БКНФ) — Boyce-Codd normal form (BCNF)

Первая нормальная форма (1НФ) — First Normal Form (1NF)

Первичный атрибут — prime attribute

Повторяющиеся группы — repeating groups

Транзитивная зависимость — transitive dependency

Третья нормальная форма (3НФ) — Third Normal Form (3NF)

Частичная зависимость — partial dependency

Четвертая нормальная форма (4НФ) — Fourth Normal Form (4NF)

Вопросы

1. Что такое нормализация?
2. Когда таблица приведена к 1НФ?
3. Когда таблица приведена к 2НФ?
4. Когда таблица приведена к 3НФ?
5. Когда таблица приведена к БКНФ?
6. Используя диаграмму зависимостей, представленную на рис. 4.19, выполните следующие задания:
 - идентифицируйте и опишите каждую из найденных зависимостей;
 - создайте базу данных, таблицы которой приведены, по крайней мере, к 2НФ, представив для каждой из них диаграммы зависимостей;
 - создайте базу данных, таблицы которой приведены, по крайней мере, к 3НФ, представив для каждой из них диаграммы зависимостей.

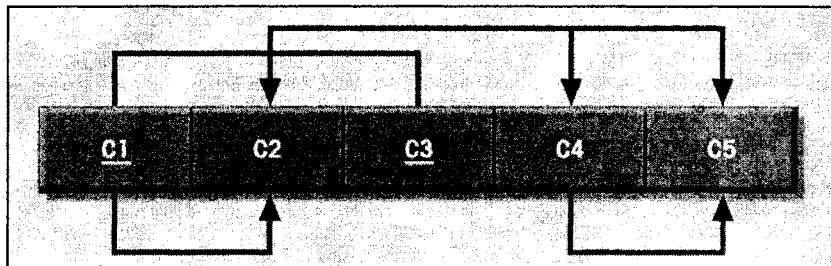


Рис. 4.19. Диаграмма зависимостей к вопросу 6

7. Что такое частичная зависимость? С какой нормальной формой она связана?
8. Какие три аномалии данных вероятнее всего станут следствием избыточности данных? Как устранить эти аномалии?
9. Определите и опишите концепцию транзитивной зависимости.

Задачи

10. Используя следующую структуру таблицы INVOICE (табл. 4.3), постройте для нее диаграмму зависимостей и определите все зависимости (включая все частичные и транзитивные). Вы можете предположить, что таблица не содержит повторяющихся групп, и что каждый номер счета может относиться более, чем к одному товару. (*Подсказка:* в этой таблице используется составной первичный ключ.)

Таблица 4.3. Структура таблицы к задаче 1

Имя атрибута	Пример
INV_NUM	211347
PROD_NUM	AA_E3422QW
SALE_DATE	06/25/1999
PROD_DESCRIPTION	B&D rotary sander, 6 дюймов, диск
VEND_CODE	211
VEND_NAME	NeverFail, Inc.
NUMBER_SOLD	2
PROD_PRICE	\$49.95

2. Используя начальную диаграмму зависимости задачи 1, удалите все частичные зависимости, постройте новую диаграмму и идентифицируйте нормальные формы для каждой созданной вами структуры таблицы.

Примечание

Вы можете считать, что данный товар поставляется одним поставщиком (vendor), но поставщик может поставлять несколько товаров. Следовательно, правильным будет предположить, что имеет место следующая зависимость:

PROD_NUM > PROD_DESCRIPTION, PROD_PRICE, VEND_CODE, VEND_NAME

(*Подсказка:* в результате ваших преобразований должны получиться три диаграммы зависимостей.)

3. Используя структуры таблиц, созданных в задаче 2, удалите все транзитивные зависимости, постройте новые диаграммы зависимостей и определите нормальные формы для каждой созданной таблицы.
4. Используя результаты задачи 3, постройте ER-диаграмму.
5. Используя табличную структуру STUDENT (табл. 4.4), постройте диаграмму зависимости и определите все зависимости (включая транзитивные).

Таблица 4.4. Структура таблицы к задаче 5

Имя атрибута	Пример
STU_NUM	211343
STU_LNAME	Stephanos
STU_MAJOR	Accounting
DEPT_CODE	ACCT
DEPT_NAME	Accounting
DEPT_PHONE	4356
COLLEGE_NAME	Business Administration
ADV_LASTNAME	Grastrand
ADV_OFFICE	HKB201
ADV_BUILDING	Howard Kallenberger Building
ADV_PHONE	2115
STU_GPA	2.87
STU_HOURS	96
STU_CLASS	Junior

6. Используя исходную диаграмму зависимости, построенную в задаче 5, нормализуйте эту структуру в соответствии с требованиями 3НФ. Если вы считаете, что на практике лучше использовать структуру 2НФ, то поясните, почему вы решили оставить 2НФ. Если необходимо, добавьте или модифицируйте атрибуты для получения необходимых детерминантов и соответствия соглашению об именовании.

Примечание

Хотя общее количество часов обучения студента (STU_HOURS) определяет (детерминирует) классификацию студента (STU_CLASS), данная зависимость не столь очевидна, как это можно предположить. Например, студент классифицируется как Junior (предпоследний курс), если он набрал от 61 до 90 часов. Следовательно, студент, который классифицируется как Junior, может набрать 66, 72, 87 или любое другое количество часов в диапазоне от 61 до 90 часов. Говоря кратко, эту классификацию определяет любое количество часов внутри определенного диапазона.

7. Используя результаты задачи 6, постройте ER-диаграмму.

Примечание

ER-диаграмма определяет небольшие фрагменты полного проекта университета. Например, этот фрагмент может использоваться совместно с описанием Tiny College, представленным в гл. 3 и задаче 11 гл. 3.

8. Для учета офисной мебели, компьютеров, принтеров и т. д. в компании FOUNDIT используется табличная структура, представленная в табл. 4.5

Таблица 4.5. Структура таблицы к задаче 8

Имя атрибута	Пример
ITEM_ID	2311345-678
ITEM_DESCRIPTION	HP DeskJet 660C принтер
BLDG_ROOM	325
BLDG_CODE	DEL
BLDG_NAME	Dawn's Early Light
BLDG_MANAGER	I. B. Rightonit

Используя эту информацию, постройте диаграмму зависимостей. Отметьте транзитивные и частичные зависимости.

9. Приняв за основу диаграмму зависимости из задачи 8, создайте набор диаграмм зависимостей, отвечающих требованиям ЗНФ. Переименуйте атрибуты в соответствии с соглашением об именовании и создайте при необходимости новые сущности и атрибуты.
10. Используя результаты задачи 9, постройте ER-диаграмму.

Примечание

Задачи 11—13 могут послужить основой разработки небольшого проекта.

11. В табличной структуре (табл. 4.6) содержится множество некорректных компонентов и свойств (например, имеется несколько многозначных атрибутов, нарушается соглашение об именовании, некоторые атрибуты неатомарны).

Таблица 4.6. Структура таблицы к задаче 11

Имя атрибута	Пример
EMP_CODE	1003
LAST_NAME	Willaker
EDUCATION	HS (школа), BBA (бакалавр), MBA (магистр)
DEPT_CODE	MKTG
DEPARTMENT	Marketing
DEPT_MANAGER	Jill H. Martin
JOB_CLASS	23
TITLE	Sales Agent (торговый агент)

Таблица 4.6 (окончание)

Имя атрибута	Пример
DEPENDENTS	Gerald (супруг), Mary (дочь), John (сын)
BIRTH_DATE	12/23/1968
HIRE_DATE	10/14/1997
TRAINING	Level 1, level 2
BASE_SALARY	\$32,255

Используя эту структуру, постройте диаграмму зависимости. Обозначьте все транзитивные и/или частичные зависимости.

12. Разбейте диаграмму зависимости, построенную в задаче 11, так, чтобы получить диаграмму, отвечающую требованиям ЗНФ. (Подсказка: возможно, потребуется создать несколько новых атрибутов. Убедитесь, что новые диаграммы зависимостей содержат атрибуты, отвечающие основным критериям проектирования, т. е. атрибуты должны не быть многозначными, отвечать соглашению об именовании и т. д.)
13. Используя результаты задачи 12, постройте ER-диаграмму.
14. Используя результаты задачи 13, постройте реляционную схему.

Примечание

Задачи 15—17 могут послужить основой при разработке небольшого проекта.

15. Предположим, что необходимо использовать следующие бизнес-правила при разработке основы проекта БД. База данных используется в управлении клубом-рестораном компании, для рассылки приглашений на обед членам клуба, планирования меню, отслеживания приглашенных на обед гостей и т. д.:
 - каждый обед сервируется на несколько членов клуба, а каждый член клуба может быть приглашен на несколько обедов;
 - член клуба получает много приглашений, каждое приглашение рассылается многим членам клуба;
 - обед основывается на одном блюде, но данное блюдо может быть основой для нескольких обедов. Например, обед может состоять из рыбного блюда, риса и кукурузы, или из рыбы, отварного картофеля и стручковой фасоли;
 - член клуба может быть приглашен на несколько обедов, на каждом обеде может присутствовать несколько членов клуба.

Поскольку менеджер не является специалистом по базам данных, первая попытка создания структуры для БД привела к таким результатам (табл. 4.7).

Таблица 4.7. Структура таблицы к задаче 15

Имя атрибута	Пример
MEMBER_NUM	214
MEMBER_NAME	Alice B. VanderVoort
MEMBER_ADDRESS	325 Meadow Park
MEMBER_CITY	Murkywater
MEMBER_ZIPCODE	12345
INVITE_NUM	8
INVITE_DATE	2/23/02
ACCEPT_DATE	2/27/02
DINNER_DATE	3/2/02
DINNER_ATTEND	Y
DINNER_CODE	5
DINNER_DESCRIPTION	Дары моря
ENTREE_CODE	3
ENTREE_DESCRIPTION	Фаршированный краб
DESSERT_CODE	8
DESSERT_DESCRIPTION	Шоколадный мусс с малиновым соусом

Используя приведенную выше структуру, постройте диаграмму зависимостей. Обозначьте все транзитивные и/или частичные зависимости. (*Подсказка:* в этой структуре используется составной первичный ключ.)

16. Разбейте диаграмму зависимостей, построенную в задаче 15, так, чтобы новые диаграммы были приведены к ЗНФ. (*Подсказка:* возможно, потребуется создать несколько новых атрибутов. Убедитесь, что новые диаграммы зависимостей содержат атрибуты, отвечающие основным критериям проектирования, т. е. атрибуты должны не быть многозначными, отвечать соглашению об именованиях и т. д.)
17. Используя результаты задачи 16, постройте ER-диаграмму.

Примечание

Задачи 18—20 могут послужить основой при разработке небольшого проекта.

18. Руководитель консалтинговой фирмы попросил вас создать БД, которая содержит табличную структуру из табл. 4.8.

Таблица 4.8. Структура таблицы к задаче 18

Имя атрибута	Пример
CLIENT_NUM	289
CLIENT_NAME	James D. Smith
DATE	12-Mar-02
CONTRACT	5842
CLASS_1	Database
CLASS_2	Networking
CLASS_3	
CLASS_4	
REGION	SE
CONS_NUM_1	25
CONS_NAME_1	Angela M. Jameson
CONS_NUM_2	34
CONS_NAME_2	Gerald K. Ricardo
CONS_NUM_3	
CONS_NAME_3	
CONS_NUM_4	
CONS_NAME_4	

Эта таблица создана для того, чтобы руководитель компании мог должным образом обеспечить консультирование клиентов. Цель состоит в том, чтобы свести клиента в данном регионе с консультантом, работающим в данном регионе, и удостовериться, что запросы клиента соответствуют специализации консультанта. Например, если клиент работает в банковском бизнесе и проживает на юго-востоке, то необходимо будет свести его с консультантом, работающим в юго-восточном регионе и специализирующемся в банковском бизнесе. База данных должна соответствовать следующим бизнес-правилам:

- каждый клиент расположен в одном регионе, но в регионе может быть несколько клиентов;
- каждый консультант может работать по нескольким контрактам, и в каждом контракте могут требоваться несколько консультантов;
- клиент может заключить более одного контракта, но каждый контракт подписывает только один клиент;
- в каждом контракте может требоваться несколько специализаций (например, клиенту могут требоваться консультации по базам данных и сетевым средствам);

- каждый консультант работает только в одном регионе, но в регионе могут работать несколько консультантов;
- каждый консультант имеет одну или более специализаций (классов), и по каждой специализации может иметься несколько консультантов. Например, консультант может быть специалистом по базам данных и сетевым средствам, а консалтинговая компания может принимать на работу несколько консультантов, специализирующихся на сетевых средствах;
- клиенту может потребоваться консультация более чем по одному направлению. Например, клиенту могут потребоваться консультации в области баз данных, по банковскому делу и сетевым средствам.

Используя это краткое описание требований и бизнес-правил, постройте диаграмму зависимостей для представленной в табл. 4.8 (весьма неполной!) структуры. Обозначьте все транзитивные и/или частичные зависимости.

- Разбейте диаграмму зависимостей, построенную в задаче 18, так, чтобы новые диаграммы были приведены к ЗНФ. (Подсказка: возможно, потребуется создать несколько новых атрибутов. Убедитесь, что новые диаграммы зависимостей содержат атрибуты, отвечающие основным критериям проектирования, т. е. атрибуты должны не быть многозначными, отвечать соглашению об именовании и т. д.)
- Используя результаты задачи 19, постройте ER-диаграмму.
- С помощью следующих трех записей в таблице CHARTER постройте диаграмму зависимостей для этой табличной структуры. Обозначьте все зависимости. (Содержимое записей расположено вертикально в целях экономии места. Запись EMP_NUM относится к пилоту, который выполняет чартерный рейс. Запись CHAR_MILES основана на вычислении протяженности маршрута по замкнутому кругу, включая пункты промежуточной посадки.) Для экономии места обозначайте атрибуты как A1, A2, A3, ..., A11. Например, CHAR_TRIP обозначьте как A1, CHAR_DATE как A2 и т. д.

Таблица 4.9. Структура таблицы CHARTER к задаче 21

	Запись 1	Запись 2	Запись 3	Запись 4	Запись 5
CHAR_TRIP	10232	10233	10234	10235	10236
CHAR_DATE	27-Jan-2002	27-Jan-2002	28-Jan-2002	28-Jan-2002	28-Jan-2002
CHAR_DESTINATION	STL	ATL	ATL	GNV	MEM
CHAR_MILES	580	470	510	1,204	280
CHAR_NUMBER	784	784	546	567	546
CUST_LNAME	Brown	Brown	Alero	Green	Alero
EMP_NUM	32	32	18	41	18
EMP_LNAME	Yantil	Yantil	Smith	Chen	Smith
AC_NUM	2098W	2098W	6711Y	6711Y	3124R

Таблица 4.9 (окончание)

	Запись 1	Запись 2	Запись 3	Запись 4	Запись 5
MOD_CODE	PA31-350	PA31-350	C-90	C-90	C-421
MOD_CHG_MILE	\$2.12	\$2.12	\$3.85	\$3.85	\$2.37

22. Разбейте диаграмму зависимостей из задачи 21, чтобы создать табличную структуру, удовлетворяющую требованиям 3НФ. Обозначьте все зависимости.
23. Постройте ER-диаграмму, отображающую преобразованные диаграммы, созданные в задаче 22. Докажите, что ER-диаграмма соответствует базе данных, с помощью которой можно отслеживать все данные, представленные в задаче 21. Покажите все сущности, связи, связности, необязательности и мощности.
24. Руководитель консалтинговой фирмы попросил вас разработать базу данных, с помощью которой можно подбирать консультанта для клиента. Цель — свести в данном регионе клиентов с консультантами таким образом, чтобы сфера интересов клиента и специализация консультанта совпадали. Например, если клиент работает в сфере банковского бизнеса и находится в юго-восточном регионе, то необходимо свести его с консультантом, работающим в этом регионе, который является специалистом по банковскому делу. Точно так же если сферой интересов клиента является автотранспортный бизнес в юго-западном регионе, то для него необходимо подобрать консультанта, работающего в этом регионе и являющегося экспертом в сфере автотранспорта. Необходимо обеспечить выполнение следующих бизнес-правил:
 - клиент может инициировать несколько контрактов, но каждый контракт заключается только одним клиентом;
 - каждый клиент расположен в одном регионе, но в регионе может быть несколько клиентов;
 - каждый консультант может обслуживать несколько клиентов, а каждого клиента может обслуживать несколько консультантов;
 - каждый консультант работает в одном регионе, но в регионе может быть несколько консультантов;
 - каждый консультант имеет одну или несколько специализаций (классов) и по каждой специализации может быть несколько консультантов. Например, консультант может быть специалистом в области баз данных и по сетевому обеспечению, а консалтинговая компания может принимать на работу несколько консультантов по сетевому обеспечению;
 - клиенту может понадобиться консультация более чем по одной проблеме, например, по базам данных, банковскому делу и сетевому обеспечению.

На основании этой информации можно построить табличную структуру (табл. 4.10) (очень неполную!). Для экономии места примеры атрибутов расположены вертикально. Примеры значений атрибутов проясняют предназначение некоторых атрибутов. Обратите внимание на пустые значения и на то, что табличная структура позволяет связать с каждым клиентом до трех консультантов и до трех специализаций.

Таблица 4.10. Табличная структура к задаче 24

Имя атрибута		Примеры значений	
CLIENT_NUM	289	289	304
CLIENT_NAME	James D. Smith	James D. Smith	Anne R. Morrow
DATE	11-Feb-2002	14-Feb-2002	22-Feb-2002
CONTRACT	5842	5843	5844
CLASS_1	Базы данных	Бухгалтерия	Базы данных
CLASS_2	Сетевые средства		Статистика
CLASS_3			Бухгалтерия
REGION	SE	SE	MW
CONSULTANT_NUM_1	25	25	38
CONSULTANT_NAME_1	Angela M. James	Angela M. James	Henry D. Smith
CONSULTANT_NUM_2	34		45
CONSULTANT_NAME_2	Gerald K. Ricardo		Mary T. Orlando
CONSULTANT_NUM_3			51
CONSULTANT_NAME_3			James D. Cason

Используя эту информацию, постройте диаграмму зависимостей для этой табличной структуры. Обозначьте все транзитивные и/или частичные зависимости.

25. Разбейте диаграмму зависимостей, построенную в задаче 24, так, чтобы новые диаграммы были приведены к ЗНФ. (Подсказка: возможно, потребуется создать несколько новых атрибутов. Убедитесь, что новые диаграммы зависимостей содержат атрибуты, отвечающие основным критериям проектирования, т. е. атрибуты должны не быть многозначными, отвечать соглашению об именовании и т. д.)
26. Используйте результаты процесса нормализации, полученные в задаче 25, для создания ER-диаграммы с полным набором обозначений для этого фрагмента базы данных. Добавьте сущность CHARGE, связанную с сущностью CLASS, что позволит консалтинговой компании выставлять клиентам счета на все выполненные работы. (Сущность CLASS содержит почасовые тарифы по всем консультационным услугам — базы данных, сетевое обеспечение, бухгалтерия и т. д.)

Примечание

Для решения задач 27—29 используйте диаграмму зависимостей, представленную на рис. 4.20.

27. Разбейте диаграмму зависимостей для создания двух новых диаграмм — одной в ЗНФ, а другой в 2НФ.

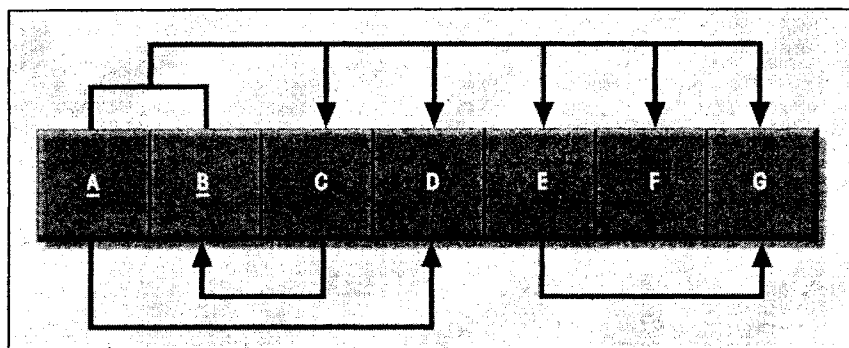


Рис. 4.20. Начальная диаграмма зависимостей для задач 27–29

28. Модифицируйте диаграммы зависимостей, построенные в задаче 27, с тем, чтобы получить набор диаграмм, удовлетворяющих требованиям 3НФ. Чтобы учесть весь набор атрибутов полностью, скопируйте диаграмму зависимостей (3НФ) из задачи 21, а затем представьте новую диаграмму, которая также будет соответствовать 3НФ. (Подсказка: одна из ваших диаграмм будет приведена к 3НФ, но не к БКНФ.)
29. Модифицируйте диаграмму зависимостей из задачи 28 так, чтобы получить набор диаграмм зависимостей, соответствующих 3НФ и БКНФ. Чтобы учесть все атрибуты, скопируйте диаграмму зависимостей (3НФ), представленную в задаче 21, а затем представьте новые диаграммы 3НФ и БКНФ.
30. Предположим, что вам представлена база данных, которая получена импортированием таблицы Excel. Из набора данных следует, что у профессора (professor) может консультироваться несколько человек (advisees), преподаватель может состоять в нескольких комитетах, а также может редактировать более одного журнала.

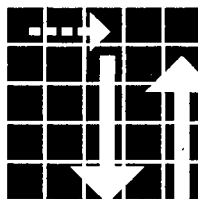
Таблица 4.11. Табличная структура для задачи 30

PROF_NUMBER	ADVISEE_NUMBER	COMMITTEE_CODE	JOURNAL_CODE
903	1235	PROMO	JMIS
903	1342	TRAFFIC	JQED
903	1432	APPEALS	MGTP
904	2143	TRAFFIC	REWY
904	1277	DEVELOPMENT	JTU

На основании этих данных

- определите многозначные зависимости;
- устраните многозначные зависимости, преобразовав таблицу в 4НФ.

Глава 5



Язык структурированных запросов (SQL)

В этой главе мы обсудим следующее:

- ☐ основные команды и функции SQL;
- ☐ использование SQL для манипулирования данными (добавление, модификация, удаление и поиск данных);
- ☐ использование SQL при запросах к базе данных с целью извлечения нужной информации;
- ☐ использование SQL при администрировании данных (создание таблиц, индексов и представлений);
- ☐ дополнительные возможности SQL, например, обновляемые представления, хранимые процедуры и триггеры.

Обзор

В этой главе мы будем изучать основы стандартного языка реляционной модели, — *языка структурированных запросов* (Structured Query Language, SQL). Этот язык содержит набор команд, предоставляющих пользователям возможность создавать базы данных и табличные структуры, манипулировать данными, администрировать данные, а также создавать запросы к базе данных для извлечения необходимой информации. Практически все программное обеспечение РСУБД поддерживает SQL, а многие поставщики программного обеспечения разрабатывают расширения стандартного набора команд SQL.

Поскольку словарь SQL достаточно прост, этот язык относительно легко изучить. Его простота обеспечивается еще и тем, что большая часть его функций остается как бы "за кадром". Например, всего лишь одна команда SQL создает сложный набор структур базы данных. Другая одиночная команда создает сложные табличные структуры, необходимые для хранения данных и манипулирования ими.

SQL — это непроцедурный язык, т. е. пользователь определяет лишь *что* необходимо сделать, но не *как* это делать. Чтобы выполнить SQL-команду, конечным пользователям или программистам не нужно знать ни формат физического хранения данных, ни сложные действия, имеющие место при выполнении SQL-команды.

В этой главе представлены основные команды и функции SQL, а также дополнительные возможности, которые имеются в большинстве РСУБД, а именно хранимые процедуры и триггеры. Хранимые процедуры позволяют использовать в SQL некоторые особенности процедурных языков программирования, например, условия и циклы. Триггеры позволяют автоматизировать выполнение важных операций. SQL может быть также расширен за счет функций, определенных пользователем.

Несмотря на все свое удобство и мощь, язык SQL является далеко не единственным инструментом подобного типа. Он, к сожалению, не обеспечивает удобный ввод данных, а также их исправление и добавление. SQL как таковой не предназначен для создания меню, специальных отчетных форм, бланков, всплывающих меню или других утилит и экранных форм, которые обычно необходимы конечному пользователю. Все эти возможности обеспечиваются дополнительными программными модулями. Тем не менее, несмотря на имеющиеся ограничения, можно сказать, что SQL является мощным инструментом извлечения информации и управления данными. Целью SQL является манипулирование данными (добавление, модификация, удаление и поиск информации) и их администрирование (создание таблиц, индексов и представлений).

5.1. Введение в SQL

В идеальном случае язык базы данных позволяет создавать базы данных и табличные структуры для выполнения основных задач управления данными (добавление, удаление и модификация), а также для выполнения сложных запросов по преобразованию неупорядоченной информации в структурированную форму. Более того, подобной язык должен выполнять все эти действия при минимальном участии пользователя, а структура его команд и синтаксис должны быть просты в изучении. Наконец, язык должен быть переносимым, т. е. соответствовать некоторым базовым стандартам, чтобы вам не пришлось изучать его заново при переходе с одной РСУБД на другую.

SQL достаточно хорошо отвечает требованиям, предъявляемым к идеальному языку баз данных. Во-первых, SQL соответствует двум базовым понятиям.

- ❑ Он является *языком определения данных*. В SQL включены команды создания табличных структур базы данных, а также определения прав доступа к базе данных.
- ❑ Он является *языком манипулирования данными*. В него включены команды вставки, обновления, удаления и извлечения данных внутри таблиц БД.

Во-вторых, SQL относительно прост в изучении. Словарь его основного набора команд содержит менее 100 слов. Более того, SQL является непроцедурным языком — нужно просто определить, *что* вы хотите сделать, не заботясь о том, *как* это будет сделано.

Наконец, Национальный институт стандартизации США (American National Standards Institute, ANSI) выпустил стандарт SQL — самую последнюю версию, которая называется SQL-99 или SQL3¹. Стандарты ANSI на язык SQL поддерживаются также

¹ "Information Technology — Database Languages — SQL — Part 2: Foundation (SQL/Foundation)", Документ за номером ANSI/ISO/IEC 9075-2-1999 от Национального института стандартизации США, www.ansi.org.

Международной организацией по стандартизации (International Organization for Standardization, ISO)². Несмотря на то что спецификации коммерческих и государственных баз данных обычно требуют придерживаться стандартов ANSI/ISO SQL, многие разработчики и поставщики РСУБД добавляют и свои специальные расширения этого языка. Поэтому редко когда возможно перенести SQL-приложение с одной РСУБД на другую без каких-либо исправлений.

Не стоит расстраиваться из-за того, что существует много диалектов SQL, поскольку различия между ними минимальны. Используйте вы Oracle, SQL Server, DB2, Microsoft Access, Informix или какую-либо другую РСУБД, опыт показывает, что достаточно потратить совсем немного времени на изучение руководства по программному обеспечению, чтобы разобраться в конкретной версии SQL, особенно если вы внимательно изучите материал, представленный в этой главе. Говоря кратко, все, что будет рассматриваться в этой главе, справедливо для всех версий SQL.

Есть ряд веских причин для изучения SQL:

- ❑ усилия ANSI привели к тому, что этот язык стал фактическим стандартом практически для всех реляционных баз данных. На самом деле, многие специалисты по реляционным БД склонны считать, что "если нет SQL, то нет и реляционности";
- ❑ SQL стал основой настоящей и будущей интеграции систем управления базами данных (СУБД), позволяя объединять иерархические, сетевые, реляционные и объектно-ориентированные базы данных;
- ❑ SQL стал катализатором разработки и распространения баз данных и клиент-серверной архитектуры БД. Мы рассмотрим подобную архитектуру в *гл. 10* и *12*.

5.2. Команды описания данных

В этом разделе мы рассмотрим, как при создании таблиц баз данных используются стандартные SQL-команды описания данных. Однако перед тем как приступить к изучению синтаксиса SQL-команд, прежде всего, представим простую модель БД и ее таблицы, которые нам будут нужны, чтобы продемонстрировать использование SQL в этой главе. При этом у нас будет ясная картина конфигурации данных, что облегчит понимание того, что мы собираемся делать, и как нам в этом может помочь SQL.

5.2.1. Модель базы данных

В этой главе для демонстрации выполнения SQL-команд мы будем использовать простую базу данных, состоящую всего из двух таблиц: PRODUCT (товар) и VENDOR (поставщик). Модель этой БД представлена на рис. 5.1.

² Обратите внимание, что ISO это греческое слово, означающее "идентичность". И в то же время ISO является акронимом International Standards Organization (Международная организация по стандартизации). В эту организацию входят национальные организации по стандартизации более чем 75 стран.

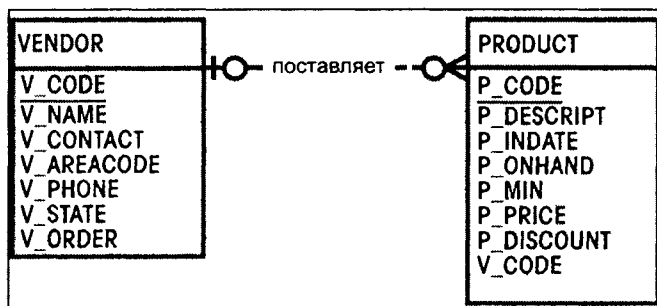


Рис. 5.1. Модель базы данных

В базе данных, представленной на рис. 5.1, отражены следующие бизнес-правила:

- ☐ поставщик может поставлять множество товаров;
- ☐ некоторые поставщики не поставляют (еще?) товары (т. е. в список могут быть включены *потенциальные* поставщики);
- ☐ если товар поставляется, то только *одним* поставщиком;
- ☐ некоторые товары не поставляются (т. е. некоторые товары могут производиться непосредственно на самом предприятии или закупаться на открытом рынке).

Поэтому между сущностями VENDOR и PRODUCT имеет место связь 1:M, причем необязательная как с той, так и с другой стороны.

5.2.2. Таблицы и их компоненты

Для демонстрации некоторых SQL-команд мы реализовали модель БД (см. рис. 5.1), создав базу данных, представленную на рис. 5.2.

В этих таблицах можно выделить следующие особенности (соответствующие бизнес-правилам, которые отражены на ER-диаграмме, представленной на рис. 5.1):

- ☐ таблица VENDOR содержит поставщиков, на которых нет ссылок в таблице PRODUCT. Скорее всего, некоторые поставщики в таблице VENDOR указаны в качестве резерва, т. е. эти поставщики никогда не поставляли продукцию, но они присутствуют в таблице VENDOR как *потенциальные* поставщики. Проектировщики баз данных в таких случаях говорят, что сущность PRODUCT *необязательна* для сущности VENDOR, поскольку поставщик может существовать и без ссылок на поставку его товара. Мы подробно изучали такие необязательные связи в гл. 3;
- ☐ имеющиеся значения атрибута V_CODE в таблице PRODUCT должны (это так и есть) соответствовать значениям этого атрибута в таблице VENDOR, чтобы обеспечить целостность на уровне ссылок;
- ☐ некоторые товары поставляются предприятиями-поставщиками, некоторые производятся самостоятельно, а часть закупается на специальных распродажах. Другими словами, товар необязательно приобретается у поставщика. Поэтому сущность VENDOR необязательна для сущности PRODUCT.

База данных: CH5_TEXT							
Таблица: PRODUCT							
P_CODE	P_DESCRIPTION	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
1GERB1	Power painter, 15 psi, 3-nozzle	03-Dec-01	8	5	\$109.99	0.00	25595
13-Q2P2	7.25-in. pwr. saw blade	13-Jan-02	32	15	\$14.99	0.05	21344
14-Q1L3	9.00-in. pwr. saw blade	13-Jan-02	18	12	\$17.49	0.00	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	15	8	\$39.95	0.00	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	23	5	\$43.99	0.00	23119
2232-QTY	B&D jigsaw, 12-in. blade	30-Dec-01	8	5	\$109.92	0.05	24288
2232-QWE	B&D jigsaw, 8-in. blade	24-Jan-02	6	5	\$99.87	0.05	24288
2238-QPD	B&D cordless drill, 1/2-in.	20-Nov-01	12	5	\$38.95	0.05	25595
23109-HB	Claw hammer	20-Jan-02	23	10	\$5.95	0.10	21225
23114-AA	Sledge hammer, 12 lb.	02-Feb-02	8	5	\$14.40	0.05	
54778-2T	Rat-tail file, 1/8-in. fine	15-Jan-02	43	20	\$4.99	0.00	21344
89-WRE-Q	Hicut chain saw, 18 in.	07-Dec-01	11	5	\$256.99	0.05	24288
PVC23DR1	PVC pipe, 3.5-in., 8-ft	20-Jan-02	188	75	\$5.87	0.00	
SM-18277	1.25-in. metal screw, 25	29-Dec-01	172	75	\$6.99	0.00	21225
SW-23118	2.5-in. wd. screw, 50	24-Dec-01	237	100	\$8.45	0.00	21231
WR37T3	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-02	18	5	\$118.95	0.10	25585

Таблица: VENDOR							
V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER	
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y	
21226	SuperLo, Inc.	Flushing	904	215-8995	FL	N	
21231	D&E Supply	Singh	615	228-3245	TN	Y	
21344	Gomez Bros.	Ortega	615	889-2546	KY	N	
22567	Dome Supply	Smith	901	678-1419	GA	N	
23119	Randssets Ltd.	Anderson	901	678-3998	GA	Y	
24004	Brackman Bros.	Browning	615	228-1410	TN	N	
24288	ORDVA, Inc.	Hakford	615	898-1234	TN	Y	
25443	B&K, Inc.	Smith	904	227-0093	FL	N	
25501	Damal Supplies	Smythe	615	890-3529	TN	N	
25595	Rubicon Sic.	Orton	904	456-0092	FL	Y	

Рис. 5.2. Содержимое таблиц PRODUCT и VENDOR базы данных CH5_TEXT

Некоторые из представленных только что условий были введены исключительно для демонстрации специфических возможностей SQL. Например, в таблице PRODUCT есть пустые значения атрибута V_CODE, чтобы впоследствии можно было показать, как отслеживать их с помощью SQL. (О значениях null мы подробно говорили в гл. 3. Здесь они отражают приведенные только что бизнес-правила.) Мы специально создали такую упрощенную среду данных, чтобы сосредоточиться на изучении сложностей SQL, а не самой БД.

5.2.3. Создание базы данных и табличных структур

В гл. 1 было показано, что СУБД представляет собой набор программ, управляющих структурой базы данных. База данных — это интегрированная компьютерная структура совместного доступа, в которой содержится информация конечного пользователя и метаданные. Все эти данные хранятся в виде таблиц. Перед началом использования новой СУБД необходимо решить две задачи.

- Во-первых, создать структуру базы данных. В ней будут храниться все таблицы БД. С точки зрения СУБД, структура БД это набор физических файлов, храня-

щихся на диске. Когда создается новая база данных, СУБД автоматически создает таблицы, в которых хранятся метаданные. Чтобы создать новую структуру БД, необходимо обладать полномочиями администратора БД.

- Во-вторых, создать таблицы, в которых будут храниться данные конечного пользователя.

Процесс создания структуры БД, как правило, отличается в разных РСУБД. Радует лишь то, что он достаточно прост в любой СУБД. (В гл. 16 будет показано, как создать структуру и как управлять информационной средой внутри такой структуры.)

В этой главе мы сосредоточимся на командах ANSI SQL, которые необходимы при создании таблиц и манипулировании ими. Для выполнения этой задачи необходимо ввести понятие схемы. *Схема* представляет собой группу связанных между собой объектов базы данных (например, таблиц и индексов). Обычно схема является собственностью пользователя или приложения. В одной базе данных может храниться несколько схем, принадлежащих разным пользователям и приложениям. Можно представлять себе схему как логически сгруппированные таблицы или логическую базу данных. Схемы полезны тем, что группируют таблицы по владельцам и обеспечивают первый уровень защиты, разрешая пользователю просматривать таблицы, принадлежащие только ему. Если используемая вами РСУБД соответствует стандартам ANSI, то схема БД создается командой:

```
CREATE SCHEMA AUTHORIZATION <creator>;
```

Поэтому если создателем является Jones, то команда будет выглядеть так:

```
CREATE SCHEMA AUTHORIZATION JONES;
```

Следует отметить, что *за исключением процесса создания базы данных* большинство поставщиков РСУБД используют язык SQL, который мало чем отличается от стандарта ANSI. Тем не менее, иногда можно столкнуться с небольшими синтаксическими отличиями. Например, в большинстве РСУБД SQL-команды заканчиваются символом ";" (точка с запятой). Однако в некоторых реализациях SQL точка с запятой не требуется. Мы в примечаниях будем предупреждать вас там, где это необходимо, о наиболее существенных различиях версий SQL.

5.2.4. Создание структуры таблиц

Создав структуру базы данных, можно приступить к определению структуры таблиц PRODUCT и VENDOR. SQL-команды, с помощью которых создаются структуры таблиц, основаны на словаре данных, представленном в табл. 5.1.

Обратите внимание на типы используемых данных в табл. 5.1. Помните, что выбор типа данных обычно диктуется природой самих данных и их предназначением. Например, для P_PRICE (стоимость), очевидно, требуется какой-то числовой тип; определять же его как символьный тип не имеет смысла. Точно так же очевидно, что название фирмы-поставщика должно иметь символьный тип. Например, VARCHAR(35) будет подходящим типом, поскольку названия фирм-поставщиков представляют собой строку переменной длины, и в этом случае допускается строка длиной до 35 символов. Аббревиатура штата США всегда состоит из двух символов, поэтому для данного атрибута логично выбрать тип CHAR(2).

Таблица 5.1. Словарь данных для БД СН5

Таблица	Атрибут	Содержимое	Тип	Формат	Диапазон	Обязателен	РК или FK	Таблица, на которую ссылается FK
PRODUCT	P_CODE	Код товара	CHAR(10)	XXXX...	Не определен	Да	РК	
	P_DESCRIPT	Описание овра	VCHAR(35)	Xxxx....	Не определен	Да		
	P_INDATE	Дата поступления на склад	DATE	DD_MM_YYYY	Не определен	Да		
	P_ONHAND	Количество	SMALLINT	####	0—999	Да		
	P_MIN	Минимальное количество	SMALLINT	####	0—999	Да		
VENDOR	P_PRICE	Цена	NUMBER(8,2)	#####	0.00—9999.00	Да		
	P_DISCOUNT	Скидка	NUMBER(4,2)	0.##	0.00—0.20	Да		
	V_CODE	Код поставщика	INTEGER	###	100—999	Да	FK	VENDOR
	V_CODE	Код поставщика	INTEGER	###	1000—9999	Да	РК	
	V_NAME	Название поставщика	CHAR(35)	Xxxxx...	Не определен	Да		
	V_CONTACT	Контактное лицо	CHAR(25)	Xxxxx...	Не определен	Да		

Таблица 5.1 (окончание)

Таблица	Атрибут	Содержимое	Тип	Формат	Диапазон	Обязателен	PK или FK	Таблица, на которую ссылается FK
	V_AREACODE	Код города	CHAR(3)	999	Не определен	Да		
	V_PHONE	Номер телефона	CHAR(8)	999-9999	Не определен	Да		
	V_STAE	Штат	CHAR(2)	XX	Не определен	Да		
	V_ORDER	Предыдущий заказ	CHAR(1)	X	Y или N	Да		

Примечание:

PK — первичный ключ

FK — внешний ключ

CHAR — символьные данные фиксированной длины от 1 до 255 символов

VARCHAR (VCHAR) — символьные данные переменной длины (от 1 до 2000 символов)/ В Oracle обозначаются как VARCHAR2

NUMBER — числовые данные; NUMBER(8,2) используется для отображения чисел с двумя знаками после запятой общим количеством до 8 знаков, включая десятичную точку. Некоторые СУБД допускают использование типа данных MONEY или CURRENCY

INT — целые значения от -2 147 483 648 до +2 147 483 647 (4 байта)

SMALLINT — целые значения от -32 768 до +32 767 (2 байта)

Для атрибута P_INDATE (дата поступления) предпочтительнее выбрать тип DATE (юлианский тип даты), а не символьный тип, поскольку юлианский тип позволяет выполнять операции сравнения и выполнения арифметических действий с датами. Например, в том случае, если мы захотим узнать, сколько дней прошло между 1 марта 1997 и 15 апреля 2002, то можно просто записать $4/15/2002 - 03/01/1997^3$. Мы также можем легко определить, какое число наступит через 60 дней после 15 декабря 2001, просто записав $12/15/2001 + 60$. Или же можно использовать системную дату PCYBD (SYSDATE), чтобы определить, какое число наступит через 60 дней, начиная с текущей даты. Арифметические действия над датами могут пригодиться при выписке счетов. Возможно, нам потребуется начислять проценты клиенту через 60 дней после выписки счета. Такие элементарные арифметические операции над датами невозможны при использовании символьного типа.

С другой стороны, выбор типа данных требует и соответствующей профессиональной интуиции. Например, для атрибута V_CODE (код поставщика) тип данных выбирается из следующих соображений:

- если мы хотим, чтобы компьютер автоматически создавал новый код поставщика, добавляя единицу к последнему коду, то необходимо использовать для V_CODE числовой тип. (Вы не можете выполнять арифметические операции над символьными данными!) Тип INTEGER гарантирует, что будут использоваться только числовые (целые) значения. Большинство реализаций SQL допускают использование типа SMALLINT для целых значений, состоящих не более чем из 6 цифр;
- если арифметические операции над V_CODE производить нежелательно, то его необходимо как символьный тип, даже если он состоит только из цифр. Символьные данные — самые "быстрые" при обработке запросов. Поэтому если не требуется выполнять над атрибутом арифметические операции, его следует хранить как символьный тип.

В этой главе мы будем использовать для атрибута V_CODE тип INTEGER.

Словарь данных, представленный в табл. 5.1, содержит лишь некоторые типы данных, поддерживаемые в SQL. Мы ограничились количеством типов, чтобы при выполнении примеров можно было использовать любую PCYBD. Если ваша PCYBD поддерживает ANSI SQL, то она поддерживает гораздо больше типов данных, чем представлено в табл. 5.2. Кроме того, как правило, PCYBD поддерживает и дополнительные типы данных, не определенные в ANSI SQL.

Таблица 5.2. Некоторые распространенные типы данных SQL

Тип данных	Формат	Описание
Числовой (numeric)	NUMBER(L,D)	Используется для хранения чисел (с фиксированной и с плавающей точкой) в диапазоне от 10^{-130} до 10^{126} с числом значащих цифр до 38. Позволяет задавать формат числа общим количеством знакомест и количеством цифр после десятичной точки. Объявление NUMBER(7,2) задает число, которое хранится с двумя знаками после десятичной точки и имеет длину до 6 знакомест (включая знак числа и десятичную точку).

³ Здесь использован "американский" формат записи даты ЧЧ/ММ/ГГГГ. — Прим. пер.

Таблица 5.2 (продолжение)

Тип данных	Формат	Описание
		Примеры: 12.32, 134.99. Если задан только один параметр, например, NUMBER(7), то для числа определено только общее количество знакомест без задания количества цифр после десятичной точки. Поэтому для объявления NUMBER(7) допустимы значения и 0.023 и —243562. Объявление NUMBER без параметров также допустимо, при этом для хранения используется системное определение формата хранения "по умолчанию"
	INTEGER	Часто допускается аббревиатура INT. Представляет собой целые числа (без десятичной точки), поэтому в таком формате нельзя хранить дробные значения. Длина до 11 цифр. Примеры: 12, 36576, 9989765
	SMALLINT	Как INTEGER, но количество цифр ограничено шестью. Если значения атрибута относительно небольшие, то в целях повышения эффективности используйте SMALLINT вместо INT
	DECIMAL(L,D)	Как NUMBER, но с ограниченным <i>снизу</i> числом знакомест (не менее шести), т. е. большая длина допустима, а менее шести знакомест — нет. DECIMAL(9,2). DECIMAL(9) и DECIMAL — допустимые примеры
Символьный (character)	CHAR(L)	Символьные данные фиксированного размера до 255 символов. Если вы храните строки меньшей длины, чем указанная в параметре L, то оставшиеся места заполняются пробелами. Поэтому если задано определение CHAR(25), то строки "Smith" и "Katzenjammer" будут храниться как Smithbbbbbbbbbbbbbbbbbbbbbb и Katzenjammerbbbbbbbbbbbbbbbbbb (символом b здесь обозначен пробел). Короче говоря, если необходимо хранить строки переменной длины, то не стоит использовать спецификацию CHAR, потому что при этом память будет использоваться неэффективно. Однако, например, телефонный код города в США всегда состоит из трех цифр, поэтому использование CHAR(3) в данном случае будет оправдано
	VARCHAR(L)	Символьные данные переменной длины. Использование объявления VARCHAR(25) позволяет хранить строки до 25 символов длиной. В отличие от объявления CHAR, VARCHAR не хранит заполняющие пробелы. Поэтому если строки "Smith" и "Katzenjammer" объявлены как VARCHAR(25), то они хранятся как Smith и Katzenjammer. Поэтому объявление VARCHAR более эффективно при хранении строк переменной длины, чем CHAR. В РСУБД Oracle используются объявления и VARCHAR и VARCHAR2

Таблица 5.2 (окончание)

Тип данных	Формат	Описание
Date (Date)	DATE	Это объявление позволяет хранить данные в юлианском формате. Значение года хранится как четырехзначное число от 0001 до 9999, значение месяца — двузначное число от 01 до 12, значение дня — двузначное число от 01 до 31. Другими словами, спецификация DATE автоматически резервирует 10 знакомест

В дополнение к типам данных, перечисленным в табл. 5.2, в SQL поддерживаются некоторые другие типы: TIME, TIMESTAMP, REAL, DOUBLE, FLOAT и интервалы, например, INTERVAL DAY TO HOUR. Многие PCyБД также пополняют этот список типами OLE (OLE — Object Linking and Embedding, технология связывания и внедрения объектов), LOGICAL, CURRENCY, autonumber (автонумератор) и sequence (последовательность). Мы упомянули эти типы данных только для того, чтобы показать многообразие имеющихся типов описания данных, что делает SQL очень мощным инструментом инфраструктуры БД. Однако поскольку эта глава представляет собой введение в основы SQL, мы ограничимся обсуждением типов данных, приведенных в табл. 5.2.

Используя свойства таблиц PRODUCT и VENDOR, мы можем реализовать структуры этих таблиц с помощью SQL-команды CREATE TABLE, которая имеет следующий формат:

```
CREATE TABLE <имя таблицы> (
<имя и свойства атрибута1,
имя и свойства атрибута2,
имя и свойства атрибута3,
обозначение первичного ключа,
обозначение внешнего ключа и его характеристики >);
```

Для наглядности большинство SQL-программистов размещают описание каждого атрибута в отдельной строке. Кроме того, для вертикального выравнивания свойств и характеристик атрибутов используются пробелы. Наконец, в именах как таблиц, так и атрибутов используются только заглавные буквы. Эти соглашения мы будем использовать не только в приведенных ниже примерах, но и во всех последующих главах книги.

```
CREATE TABLE VENDOR ( ,
```

```
V_CODE      INTEGER      NOT NULL UNIQUE,
```

```
V_NAME      VARCHAR(35)  NOT NULL,
```

```
V_CONTACT   VARCHAR(15)  NOT NULL,
```

```
V_AREACODE  CHAR(3)      NOT NULL,
```

```
V_PHONE     CHAR(8)      NOT NULL,
```

Комментарий:

пример: 21266

пример: SuperLoo, Inc

пример: Flushing

пример: 904

пример: 215-8995

V__STATE	CHAR(2)	NOT NULL	пример: FL
V__ORDER	CHAR(1)	NOT NULL,	пример: N
PRIMARY KEY (V__CODE));			

Примечание

- Комментарии не являются частью SQL-кода.
- Поскольку таблица PRODUCT содержит внешний ключ, который ссылается на таблицу VENDOR, желательно создавать таблицу VENDOR в первую очередь. (На самом деле, сторона "M" связи всегда ссылается на сторону "1", поэтому в связи 1:M всегда создавайте вначале сторону "1".)
- Если PCYБД не поддерживает формат VARCHAR или FCHAR, используйте формат CHAR.
- Если PCYБД не поддерживает формат SINT или SMALLINT, используйте формат INTEGER или INT. Если не поддерживается и формат INTEGER, используйте формат NUMBER.
- Если PCYБД не поддерживает спецификацию UNIQUE для первичных и внешних ключей, удалите ее из приведенного выше SQL-кода.

CREATE TABLE PRODUCT (<i>Комментарий</i>
P_CODE VARCHAR(10) NOT NULL UNIQUE,	пример: 2238/QPD
P_DESCRIPT VARCHAR(35) NOT NULL,	пример: Rat-tail file ?-in
P_INDATE DATE NOT NULL,	пример: 11/1/01
P_ONHAND SMALLINT NOT NULL,	пример: 43
P_MIN SMALLINT NOT NULL,	пример: 20
P_PRICE NUMBER(8,2) NOT NULL,	пример: 4.99
P_DISCOUNT NUMBER(5,2) NOT NULL,	пример: 0.05
V_CODE INTEGER,	пример: 21344
PRIMARY KEY (P_CODE),	
FOREIGN KEY (V_CODE) REFERENCES VENDOR	
ON DELETE RESTRICT	
ON UPDATE CASCADE);	

Помните, что вы не сможете удалить поставщика, а также товар, на который ссылается этот поставщик. Однако если вы делаете какие-то изменения в атрибуте V_CODE таблицы VENDOR, то они будут сделаны во всей системе каскадным образом в целях обеспечения целостности на уровне ссылок.

В только что приведенной SQL-команде создания таблицы можно выделить следующие особенности.

- ❑ Обозначение первичного ключа содержит спецификации **NOT NULL** и **UNIQUE**, обеспечивающие выполнение требования целостности на уровне сущностей.

Примечание

Если первичный ключ составной, то все его компоненты должны находиться внутри скобок и разделяться запятыми. Например, в гл. 2 первичный ключ таблицы **LINE** (строка) на рис. 2.28 состоит из двух атрибутов **INV_NUMBER** и **LINE_NUMBER**. В этом случае первичный ключ определяется так:

```
PRIMARY KEY (INV_NUMBER, LINE_NUMBER)
```

*Порядок следования компонентов первичного ключа имеет существенное значение, поскольку индексирование начинается с атрибута, который следует первым, затем индексируется следующий и т. д. В этом примере номера строк (**LINE_NUMBER**) будут упорядочены в пределах каждого номера счета (**INV_NUMBER**):*

```
INV_NUMBELINE_NUMBER
```

1001	1
1001	2
1002	1
1003	1
1003	2
1003	3
...	...

- ❑ Спецификация **NOT NULL**, применяемая для описания многих атрибутов, обеспечивает обязательность ввода информации. Если ввод данных в конкретном случае необходим, то спецификация **NOT NULL** не позволит конечному пользователю пропустить этот атрибут и продолжить вводить данные в следующую запись. Поскольку эта спецификация определяется на уровне таблицы и хранится в словаре данных, прикладные программы могут использовать эту информацию для автоматического контроля словаря данных.
- ❑ Список атрибутов заключается в скобки.
- ❑ После имени атрибута и определения его свойств ставится запятая.
- ❑ Командная последовательность заканчивается точкой с запятой (;). Следует иметь в виду, что в некоторых РСУБД ставить точку с запятой не нужно.

Об именовании столбцов

Не используйте в названиях столбцов математические символы (+, -, / и т. п.). Например, название **PER-NUM** повлечет за собой сообщение об ошибке, а название **PER_NUM** — вполне допустимо. Также не используйте зарезервированные слова, которые используются в SQL для выполнения специфических функций. Например, в некоторых СУРБД имя столбца **INITIAL** вызовет сообщение об ошибке "invalid column name" — неправильное имя столбца.

Вниманию пользователей Oracle

Нажатие клавиши <Enter> после каждой строки автоматически создает "line number" — номер строки, если до этого не была введена точка с запятой (символ нормального завершения любой SQL-команды). Например, команда CREATE TABLE в Oracle будет выглядеть так:

```
CREATE TABLE PRODUCT (  
2      P_CODE          VARCHAR2(10)  
                                CONSTRAINT PRODUCT_P_CODE_PK PRIMARY KEY,  
3      P_DESCRIPTOR    VARCHAR2(35)          NOT NULL,  
4      P_INDATE        DATE                  NOT NULL,  
5      P_ONHAND        NUMBER                NOT NULL,  
6      P_MIN           NUMBER                NOT NULL,  
7      P_PRICE         NUMBER(8,2)          NOT NULL,  
8      P_DISCOUNT     NUMBER(5,2)          NOT NULL  
9      V_CODE          NUMBER,  
                                CONSTRAINT PRODUCT_V_CODE_FK  
                                REFERENCES VENDOR(V_CODE);
```

5.2.5. Домены

В гл. 2 было сказано, что домен (domain) представляет собой множество допустимых значений столбца. В ANSI SQL мы можем определить домен, задав его имя, тип данных, значение по умолчанию и ограничения (условие) в домене. Синтаксис команды CREATE DOMAIN в ANSI SQL выглядит следующим образом:

```
CREATE DOMAIN <имя домена> AS DATA_TYPE  
[DEFAULT <значение по умолчанию>]  
[CHECK (<условие>)]
```

Тип данных в определении домена должен соответствовать одному из типов данных SQL (т. е. CHAR, DECIMAL, NUMBER и т. д.). Необязательное выражение DEFAULT позволяет задать значение по умолчанию для строк атрибута, если пользователь не определил для них других значений. Выражение CHECK также необязательно и позволяет определить условие для значений домена атрибута. Например, можно определить домен MATRIAL_STATUS (семейное положение) следующим образом:

```
CREATE DOMAIN MARITAL_STATUS AS VARCHAR(8)  
CHECK (VALUE IN ('Single', 'Married', 'Divorced', 'Widowed'))4;
```

Имя домена можно использовать при создании таблицы вместо типа данных атрибута в команде CREATE TABLE.

⁴ В русском варианте список ограничений был бы длинее: "Холост", "Не замужем", "Женат", "Замужем", "Разведен", "Разведена", "Вдовец", "Вдова". — Прим. пер.

Например, основную структуру таблицы EMPLOYEE можно определить следующим образом:

```
CREATE TABLE EMPLOYEE (  
EMP_NUM      INTEGER          NOT NULL  
              CONSTRAINT EMPPK PRIMARY KEY (EMP_NUM) ,  
EMP_LNAME    VARCHAR(15)      NOT NULL,  
EMP_FNAME    VARCHAR(15)      NOT NULL  
EMP_STATUS   MARITAL_STATUS   NOT NULL);
```

Еще одно преимущество использования доменов состоит в том, что вы можете получать значения доменов из других таблиц с помощью оператора SELECT в объявлении CHECK. Предположим, у вас имеется таблица DISCOUNTTBL для хранения диапазона возможных скидок, предоставляемых компаниями. В этом случае таблица DISCOUNTTBL может содержать всего один атрибут DISCOUNT. В свою очередь, в этой таблице могут иметься три строки, содержащие значения 10.00, 15.00 и 20.00. Исходя из этого, оператор CREATE DOMAIN может быть записан следующим образом:

```
CREATE DOMAIN DISCOUNT_RATE AS NUMBER (5,2)  
CHECK (VALUE IN (SELECT DISCOUNT FROM DISCOUNTTBL));
```

В этом случае домен DISCOUNT_RATE будет ограничен значениями 10.00, 15.00 и 20.00. Каждое новое значение, добавленное руководством отдела продаж в таблицу DISCOUNTTBL, автоматически станет частью домена DISCOUNT_RATE. Очень важным является то, что *использование определения домена не заменяет собой задание ограничений на целостность внешнего ключа*.

Необходимо тщательно взвесить преимущество использования доменов таблицы и создание новой связанной таблицы. В основном домены используются в следующих случаях:

- ☐ когда набор возможных значений атрибута сравнительно невелик, не меняется слишком часто, а свойства атрибута не настолько важны, что для них стоит создавать новый объект;
- ☐ когда основной целью является унификация свойств данных атрибута, входящего в несколько таблиц, и, опять-таки, если его свойства не имеют столь существенного значения, чтобы для них стоило создавать новый объект.

Для удаления определения домена используется оператор DROP DOMAIN. Его синтаксис выглядит следующим образом:

```
DROP DOMAIN <имя домена> [RESTRICT | CASCADE]5
```

Например, для удаления домена MARITAL_STATUS нужно выполнить команду:

```
DROP DOMAIN MARITAL_STATUS CASCADE;
```

⁵ Знак "|" обозначает логическое "ИЛИ", т. е. или RESTRICT или CASCADE. — Прим пер.

Во всех таблицах, где используются атрибуты, значения которых были определены доменом `MARITAL_STATUS`, модифицирующее выражение `CASCADE` автоматически изменит тип данных этих атрибутов на тот тип, который использовался в определении домена. Если, например, домен был определен как

```
CREATE DOMAIN MARITAL_STATUS AS VARCHAR(8)
```

то для всех связанных с этим определением атрибутов будет установлен тип данных `VARCHAR(8)`. В то же время модифицирующее выражение `RESTRICT` используется для того, чтобы предотвратить удаление домена до тех пор, пока в БД будут иметься атрибуты, связанные с этим доменом.

Примечание

Не все РСУБД поддерживают домены именно в таком виде, а некоторые РСУБД не поддерживают домены вообще. Например, в Oracle8i не поддерживается оператор `CREATE DOMAIN`. Для более полной информации загляните в руководство по РСУБД.

5.2.6. Ограничения целостности SQL

В гл. 2 мы узнали, что следование правилам целостности на уровнях сущностей и ссылок имеет существенное значение в среде реляционных баз данных. К счастью, большинство версий SQL поддерживают оба правила целостности. Например, целостность на уровне сущностей обеспечивается автоматически, если первичный ключ определен в операторе `CREATE TABLE`. Так, мы можем создать структуру таблицы `VENDOR` и обеспечить выполнение целостности на уровне сущностей с помощью спецификаций `NOT NULL` и `UNIQUE` в атрибуте первичного ключа, как было показано ранее в примерах команды `CREATE TABLE`:

```
V_CODE INTEGER NOT NULL UNIQUE
```

или с помощью определения первичного ключа `PRIMARY KEY (V_CODE)` в конце командной последовательности.

Если еще раз вернуться к командной последовательности создания таблицы `PRODUCT`, то стоит обратить внимание, что целостность на уровне ссылок обеспечивается следующим определением таблицы:

```
FOREIGN KEY (V_CODE) REFERENCES VENDOR
```

Кроме того, определения

```
ON DELETE RESTRICT
```

```
ON UPDATE CASCADE
```

гарантируют выполнение следующих двух условий:

- поскольку в список поставщиков включены и те, кто находится в резерве, мы не хотим удалять поставщика из этого списка, даже если на него нет ни одной ссылки в таблице `PRODUCT` (`ON DELETE RESTRICT`);
- с другой стороны, если в атрибуте `V_CODE` таблицы `VENDOR` сделаны изменения, то они должны автоматически найти отражение в таблице `PRODUCT`, где

есть ссылки на V_CODE (ON UPDATE CASCADE). Это ограничение делает невозможным существование в таблице PRODUCT значений V_CODE, которые указывали бы на несуществующие значения V_CODE в таблице VENDOR. Другими словами, спецификация ON UPDATE CASCADE гарантирует целостность на уровне ссылок. В гл. 3 было показано, почему так важно при реализации ER-моделей определить первичный и внешний ключи. Также необходимо отметить, что некоторые СУБД автоматически обеспечивают ограничения целостности на уровне ссылок. Например, если вы работаете в среде Oracle8i, то нет необходимости использовать ограничения ON DELETE RESTRICT и ON UPDATE RESTRICT в определении таблиц, поскольку данная РСУБД автоматически реализует эти ограничения, если определен внешний ключ.

5.3. Команды манипулирования данными

В это разделе мы познакомимся с SQL-командами (операторами), представленными в табл. 5.3.

Таблица 5.3. Операторы языка SQL

Оператор	Описание
INSERT	Позволяет построчно вводить данные в таблицу. Используется для начального задания элементов данных в новой табличной структуре или для добавления данных в таблицу, где уже имеется информация
SELECT	Позволяет сделать выборку из содержимого таблицы. Фактически оператор SELECT это команда запроса, а не управления данными. Несмотря на это, оператор SELECT вводится в этом разделе, поскольку с его помощью можно проверить результаты действий команд управления данными
COMMIT	Сохраняет данные на диске
UPDATE	Позволяет вносить изменения в данные
DELETE	Удаляет одну или более строк
ROLLBACK	Возвращает содержимое таблицы БД в ее первоначальное состояние (к моменту выполнения последней команды COMMIT)

5.3.1. Ввод данных

Если при создании таблицы PRODUCT был обозначен внешний ключ (см. разд. 5.2.6), то структура таблицы VENDOR должна быть создана до создания таблицы PRODUCT, а значения атрибутов таблицы VENDOR нужно вводить до ввода значений атрибутов таблицы PRODUCT. Поскольку атрибут V_CODE таблицы PRODUCT используется для ссылки на атрибут V_CODE таблицы VENDOR, то, очевидно, будет иметь место нарушение целостности, если значение V_CODE таблицы VENDOR не будет существовать во время заполнения таблицы PRODUCT.

Для ввода данных в таблицу в SQL используется оператор INSERT. Базовая структура оператора INSERT выглядит следующим образом:

```
INSERT INTO <имя таблицы> VALUES (значение атрибута1, значение атрибута2,  
... и т. д.)
```

Используя структуру таблицы **VENDOR**, определенную нами ранее, и пример данных для нее (см. рис. 5.2), можно ввести первую запись в таблицу:

```
INSERT INTO VENDOR  
VALUES (21225,'Bryson, Inc.','Smithson','615','223-3234','TN','Y');
```

Вводя данные, проверьте, что информация набрана вами правильно, и только после этого нажмите клавишу <Enter>. Вторую запись можно ввести так:

```
INSERT INTO VENDOR  
VALUES (21226,'Superloo, Inc.','Flushing','904','215-8995','FL','N');
```

и т. д. до тех пор, пока не будет введена вся необходимая информация. Записи таблицы **PRODUCT** вводятся таким же способом на основе данных, представленных на рис. 5.2. Например, первые две строки вводятся следующим образом (после ввода каждой строки необходимо нажимать клавишу <Enter>):

```
INSERT INTO PRODUCT  
VALUES ('11QER/31','Power painter, 15 psi., 3-nozzle', '12/03/01', 8,  
5, 109.99, 0.00, 25595);  
INSERT INTO PRODUCT  
VALUES ('13-Q2/P2','7.25-in. pwr. saw blade','01/13/02',32,15,14.99,  
0.05, 21344);
```

Примечание

Ввод даты может варьироваться в зависимости от используемого формата. 25 марта 2002 года можно ввести как "03/25/2002", или как "25-Mar-02", или как-нибудь иначе, в зависимости от установленного в РСУБД формата. В некоторых РСУБД не разрешается вводить даты в апострофах, поэтому в таких случаях нужно вводить 03/25/2002, а не "03/25/2002". Пользователи Access знают, что разделителем дат в этой РСУБД является решетка (#), т. е. дату 25 марта 2002 года нужно вводить как #3/25/2002#.

В приведенных выше операторах ввода данных отметим следующие особенности:

- ☐ содержимое строки заключено в скобки. (Обратите внимание, что первым символом после объявления **VALUES** является открывающая скобка, а последний символ в командной последовательности — закрывающая скобка.);
- ☐ символы (строки) и значения дат должны быть заключены в апострофы ('');
- ☐ численные величины в апострофы не заключаются;
- ☐ значения атрибутов разделяются запятыми.

Очевидно, что ввод данных в SQL организован не очень удобно. Понятно, что пользовательские приложения гораздо лучше приспособлены для этой цели. Например,

для создания представления данных (data view) и формы заполнения (entry form), представленных на рис. 5.3, использовались программные средства MS Access.

PRODUCT Table Data View and Data Entry

Product code:

Description:

Stock date:

Units on hand:

Minimum units:

Price:

Discount rate:

Vendor code:

Duck Data Entry System

Close the product form

Record: 1 of 16

Рис. 5.3. Представление данных и форма заполнения

До сих пор мы показывали, как вводить строки, в которых все атрибуты определены. Но что делать в том случае, если товар не имеет поставщика или вы не знаете кода поставщика? Говоря кратко, предположим, что мы хотим оставить значение кода поставщика пустым. Для ввода пустых значений используется такой синтаксис:

```
INSERT INTO PRODUCT
VALUES ('BRT-345','Titanium drill bit', '10/18/02', 75, 10, 4.50, 0.06,
      NULL);
```

Кстати, обратите внимание, что значение NULL допустимо лишь потому, что атрибут V_CODE необязателен (мы не использовали объявление NOT NULL в операторе CREATE TABLE для этого атрибута).

Бывает и так, что необязательными являются несколько атрибутов. Вместо того чтобы для таких атрибутов в команде INSERT несколько раз вводить значение NULL, мы можем указать только те атрибуты, которые имеют значимые величины. Если требуется ввести значения только в поля P_CODE и P_DESCRIPT, то можно написать такой оператор:

```
INSERT INTO PRODUCT(P_CODE, P_DESCRIPT)
VALUES ('BRT-345','Titanium drill bit');
```

5.3.2. Сохранение содержимого таблицы

Все изменения, внесенные в таблицу, физически не сохраняются на диске до тех пор, пока вы либо не выполните команду COMMIT <имя таблицы>, либо не закроете базу данных, либо не выйдете из SQL. Таким образом, если отказ питания или иной сбой во время работы произойдет до того, как вы зафиксируете (commit) сделанные вами изменения в таблице, ее содержимое останется без изменений. Поэтому будет нелишним периодически сохранять записи, размещенные в таблице PRODUCT, с помощью такого оператора:

```
COMMIT PRODUCT;
```

Оператор COMMIT часто используется для сохранения дополнений, изменений и удалений в содержимом таблицы. Как только данные с помощью этого оператора сохранены, можно выходить из программы-интерпретатора SQL.

Примечание

Если вы ввели только первые две строки в таблицу PRODUCT, как это было показано в предыдущем разделе, команда COMMIT сохранит эти две строки. Остальные записи вы можете добавить позже с помощью оператора INSERT, подобного тому, который использовался для ввода первых записей.

Не хотелось бы создать впечатление, что единственное предназначение команды COMMIT состоит в сохранении сделанных изменений. На самом деле, основное предназначение команд COMMIT и ROLLBACK (см. разд. 5.3.6) состоит в обеспечении целостности обновления БД при управлении транзакциями, что мы будем детально изучать в гл. 9.

5.3.3. Распечатка содержимого таблицы

Если вы хотите посмотреть содержимое таблицы, используйте команду SELECT, за которой должен следовать список необходимых вам атрибутов. Если же вы хотите просмотреть все атрибуты, то можно использовать символ "звездочка" (*), который означает "все атрибуты". (Символ, на место которого можно подставить другие символы или команды, называется группирующим символом (wildcard character) или метасимволом). Например, для вывода всех атрибутов и всех строк таблицы PRODUCT можно использовать следующий оператор:

```
SELECT * FROM PRODUCT;
```

На рис. 5.4 представлен результат выполнения этой команды.

Вниманию пользователей Oracle

В некоторых реализациях SQL (например, в Oracle SQL) имена атрибутов *обрезаются* для подгонки их к ширине столбца. Однако SQL*Plus PCУБД Oracle позволяет установить отображаемую ширину столбца с тем, чтобы вывести полное его имя. Формат вывода данных может быть также изменен независимо от способа хранения информации в таблице. Например, если вы захотите отобразить символ доллара (\$) при выводе значений атрибута P_PRICE, то можно декларировать это следующим образом:

```
COLUMN P_PRICE FORMAT $99,999.99
```

При этом формат вывода изменится с 12345.67 на \$12,345.67

	P_CODE	P_DESCRIPTION	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE
▶	TIGER31	Power painter, 15 psi, 3-nozzle	03-Dec-01	8	5	\$109.99	0.00	25595
	13-Q2/P2	7.25-in. pwr. saw blade	13-Jan-02	32	15	\$14.99	0.05	21344
	14-Q1/L3	9.00-in. pwr. saw blade	13-Jan-02	18	12	\$17.49	0.00	21344
	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	15	8	\$39.95	0.00	23119
	1558-QWI	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	23	5	\$43.99	0.00	23119
	2232AQTY	B&D jigsaw, 12-in. blade	30-Dec-01	8	5	\$109.92	0.05	24288
	2232AQWE	B&D jigsaw, 8-in. blade	24-Jan-02	6	5	\$99.87	0.05	24288
	2238AQPD	B&D cordless drill, 1/2-in.	20-Nov-01	12	5	\$38.95	0.05	25595
	23109-HB	Claw hammer	20-Jan-02	23	10	\$5.95	0.10	21225
	23114-AA	Sledge hammer, 12 lb.	02-Feb-02	8	5	\$14.40	0.05	
	54778-2T	Rat-tail file, 1/8-in. fine	15-Jan-02	43	20	\$4.99	0.00	21344
	89-WRE-Q	Hicut chain saw, 16 in.	07-Dec-01	11	5	\$256.99	0.05	24288
	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Jan-02	188	75	\$5.87	0.00	
	SM-18277	1.25-in. metal screw, 25	29-Dec-01	172	75	\$6.99	0.00	21225
	SW-23116	2.5-in. wd. screw, 50	24-Dec-01	237	100	\$8.45	0.00	21231
	WR3/TT3	Steel matting, 4'x8'x1/8", 5" mesh	17-Jan-02	18	5	\$119.95	0.10	25595

Рис. 5.4. Содержимое таблицы PRODUCT

Команду SQL можно задавать и в одной строке, однако командная последовательность SQL выглядит более структурированно, если ее компоненты размещаются в отдельных строках с соответствующими отступами. Поэтому предыдущую команду лучше записать так:

```
SELECT *
FROM PRODUCT;
```

Вместо использования метасимвола можно перечислить все атрибуты таблицы:

```
SELECT P_CODE, P_DESCRIPTION, P_INDATE, P_ONHAND, P_MIN, P_PRICE,
P_DISCOUNT, V_CODE
FROM PRODUCT;
```

Примечание

Порядок распечатки может отличаться от представленного на рис. 5.4, который является результатом управляемых системой индексных операций на основе первичного ключа. Далее будет показано, как можно управлять выводом, чтобы он всегда был представлен в нужном порядке. На данном этапе нужно только проверить распечатку и убедиться, что в ней перечислены все записи, не задумываясь о порядке их появления в списке.

5.3.4. Изменение данных

Для модификации данных в таблице используется команда UPDATE. Например, если вы захотите изменить значение атрибута P_INDATE во второй строке на January 18, 2002, то для правильного определения местоположения строки используйте значение первичного ключа (13-Q2/P2).

```
UPDATE PRODUCT
SET P_INDATE = '01/18/2002'
WHERE P_CODE = '13-Q2/P2';
```

(Помните, что в Oracle команду SET необходимо использовать в виде SET P_INDATE = '18-JAN-2002').

Если требуется изменить более одного атрибута в строке, нужно в команде разделять их друг от друга запятой:

```
UPDATE PRODUCT
SET P_INDATE = '01/18/2002', P_PRICE = 15.99, P_MIN = 10
WHERE P_CODE = '13-Q2/P2';
```

Убедитесь в правильности сделанных изменений, распечатав содержимое таблицы PRODUCT с помощью такой команды:

```
SELECT *
FROM PRODUCT;
```

5.3.5. Восстановление содержимого таблиц

До тех пор, пока вы не выполнили команду COMMIT для сохранения изменений в таблице, вы можете вернуть БД в ее предыдущее состояние с помощью команды ROLLBACK. В этой команде не нужно определять имя таблицы. SQL предполагает, что восстанавливать нужно базу данных, находящуюся в настоящий момент в памяти. Поэтому нужно просто выполнить команду

```
ROLLBACK;
```

и нажать клавишу <Enter>. После этого можно снова использовать оператор SELECT, чтобы убедиться, что команда ROLLBACK действительно восстановила исходные значения данных. Более подробно мы изучим команды COMMIT и ROLLBACK в гл. 9.

5.3.6. Удаление строк таблицы

Строки таблицы очень просто удалить с помощью оператора DELETE. Например, если из таблицы PRODUCT нужно удалить восьмую строку (значение первичного ключа P_CODE = '2238/QPD'), используется такой оператор:

```
DELETE FROM PRODUCT
WHERE P_CODE = '2238/QPD';
```

Обратите внимание, что значение первичного ключа позволяет SQL точно определить строку, которую необходимо удалить. Удаление не ограничивается только заданием первичного ключа: в объявлении WHERE можно использовать любой атрибут. Например, следующая команда удалит восемь строк (в которых значение атрибута P_MIN = 5):

```
DELETE FROM PRODUCT
WHERE P_MIN = 5;
```

(Сверьтесь с содержимым таблицы PRODUCT, представленной на рис. 5.2.)

Примечание

Для возврата к прежнему состоянию таблицы можно использовать оператор ROLLBACK. В остальных разделах этой главы мы будем использовать исходное состояние этой таблицы.

5.4. Запросы

Все запросы основаны на команде SELECT. Фактически для того, чтобы проверить все действия, выполненные в предыдущем разделе, мы пользовались простейшим запросом

```
SELECT *  
FROM PRODUCT;
```

чтобы вывести все строки по всем столбцам таблицы. В этом разделе мы рассмотрим команду SELECT более подробно, изучив ограничения, которые можно задавать в качестве критериев поиска. Команда SELECT вместе с соответствующими условиями поиска является очень мощным средством структурирования данных. Например, в следующем разделе мы покажем, как создавать запросы, дающие ответ на вопросы вроде следующих: "Какие товары поставляются данным поставщиком?", "Какие товары дешевле \$10.00?", "Какое количество товара от данного поставщика было продано между 5 января 2002 и 20 марта 2002?"

5.4.1. Распечатка части содержимого таблицы

Можно выбрать часть содержимого таблицы, поименовав необходимые поля и задав ограничения на строки, которые необходимо включить в распечатку. Синтаксис команды SELECT выглядит в этом случае таким образом:

```
SELECT <столбец (столбцы)>  
FROM <имя таблицы>  
WHERE <условия>;
```

Оператор SELECT будет извлекать все строки, соответствующие условиям, заданным в объявлении WHERE. Помните, что объявление WHERE необязательно. Если ни одна строка не отвечает заданным условиям, то вам будет представлен пустой экран или сообщение о том, что ни одна строка не удовлетворяет заданным условиям. Например, запрос:

```
SELECT P_DESCRIPTOR, P_INDATE, P_PRICE, V_CODE  
FROM PRODUCT  
WHERE V_CODE = 21344;
```

выведет описание, дату и стоимость товаров для поставщика с номером 21344, как показано на рис. 5.5.

	P_DESCRPT	P_INDATE	P_PRICE	V_CODE
▶	7.25-in. pwr. saw blade	13-Jan-02	\$14.99	21344
	9.00-in. pwr. saw blade	13-Jan-02	\$17.49	21344
	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99	21344

Рис. 5.5. Выборка атрибутов из таблицы PRODUCT для поставщика с номером 21344

Примечание

Пользователи MS Access могут воспользоваться генератором запросов QBE (Query By Example — запрос по образцу). Хотя генератор запросов QBE создает собственную версию SQL-команд, можно вводить в окне Access SQL и стандартные команды, как это показано на третьем фрагменте рис. 5.6. На рис. 5.6 представлены экран QBE, созданный генератором QBE код SQL и модифицированный SQL-запрос.

Примечание

Обратите внимание, что Access автоматически обозначает источник данных с помощью имени таблицы в качестве префикса. Позже будет показано, что префикс в виде имени таблицы используется для предотвращения двусмысленности при написании оператора SELECT, в котором используется несколько таблиц. Например, и таблица VENDOR и таблица PRODUCT содержат атрибут V_CODE. Поэтому если используются обе эти таблицы (при их объединении), необходимо указывать источник для атрибута V_CODE.

На содержимое таблицы может быть наложено любое число логических ограничений. Структура SQL-команды, таким образом, предоставляет практически неограниченные возможности. Например, для задания ограничений и условий можно использовать математические символы, представленные в табл. 5.4.

Таблица 5.4. Математические операторы

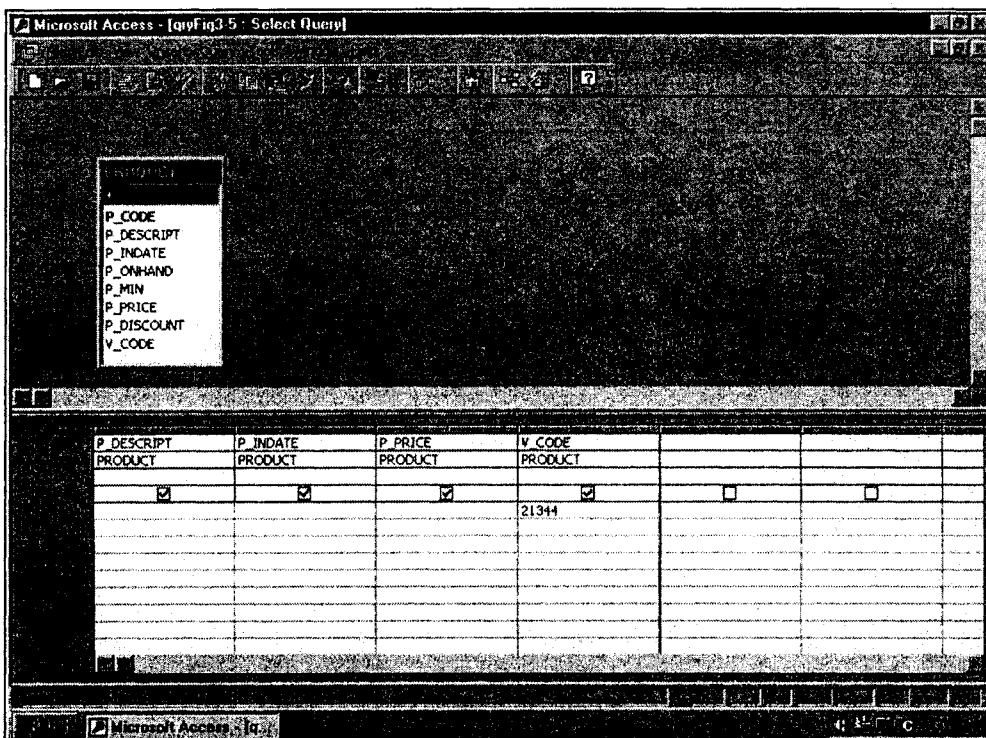
Символ	Значение	Символ	Значение
=	Равно	>	Больше
<	Меньше	>=	Больше или равно
<=	Меньше или равно	<>	Не равно*

* В некоторых версиях SQL вместо <> используется !=.

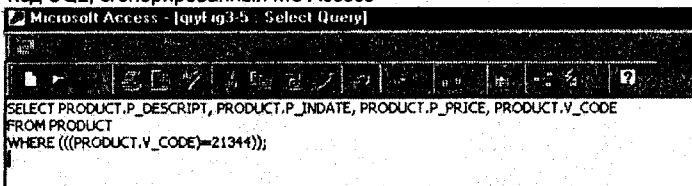
В следующем примере используется оператор "не равно":

```
SELECT P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE <> 21344;
```

Результат выполнения этой команды представлен на рис. 5.7, где выведены все строки, в которых код поставщика не равен 21344.



Код SQL, сгенерированный MS Access



Стандартный SQL

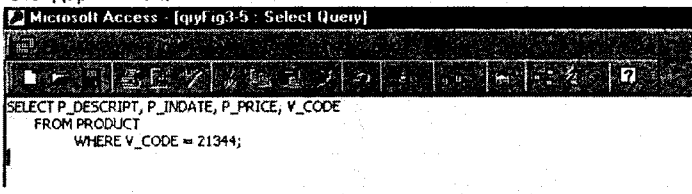


Рис. 5.6. QBE Microsoft Access и соответствующие ему команды SQL

На рис. 5.7 можно заметить, что пустые значения, обнаруженные в таблице PRODUCT (см. рис. 5.2), не включены в результат выполнения команды SELECT.

	P_DESCRIPTION	P_INDATE	P_PRICE	V_CODE
▶	Power painter, 15 psi, 3-nozzle	03-Dec-01	\$109.99	25595
▶	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	\$39.95	23119
▶	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	\$43.99	23119
▶	B&D jigsaw, 12-in. blade	30-Dec-01	\$109.92	24288
▶	B&D jigsaw, 8-in. blade	24-Jan-02	\$99.87	24288
▶	B&D cordless drill, 1/2-in.	20-Nov-01	\$38.95	25595
▶	Claw hammer	20-Jan-02	\$5.95	21225
▶	Hicut chain saw, 16 in.	07-Dec-01	\$256.99	24288
▶	1.25-in. metal screw, 25	29-Dec-01	\$6.99	21225
▶	2.5-in. wd. screw, 50	24-Dec-01	\$8.45	21231
▶	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-02	\$119.95	25595

Рис. 5.7. Выборка атрибутов таблицы PRODUCT, в которых код поставщика не равен 21344

Командная последовательность

```
SELECT P_DESCRIPTION, P_ONHAND, P_MIN, P_PRICE
FROM PRODUCT
WHERE P_PRICE <= 10;
```

даст результат, представленный на рис. 5.8.

	P_DESCRIPTION	P_ONHAND	P_MIN	P_PRICE
▶	Claw hammer	23	10	\$5.95
▶	Rat-tail file, 1/8-in. fine	43	20	\$4.99
▶	PVC pipe, 3.5-in., 8-ft	188	75	\$5.87
▶	1.25-in. metal screw, 25	172	75	\$6.99
▶	2.5-in. wd. screw, 50	237	100	\$8.45

Рис. 5.8. Выборка атрибутов из таблицы PRODUCT с ограничением на атрибут P_PRICE

Математические операции с символьными атрибутами

Поскольку компьютер идентифицирует все символы по их числовому ASCII-коду (American Standard Code for Information Interchange, американский стандартный код обмена информацией), математические операторы можно использовать и для ограничений на значения символьных атрибутов. Поэтому команда

```
SELECT P_CODE, P_DESCRIPTION, P_ONHAND, P_MIN, P_PRICE
FROM PRODUCT
WHERE P_CODE < '1558-QW1';
```

правильная, и с ее помощью будет получен список всех строк, для которых P_CODE по алфавиту меньше, чем 1558-QW1. (Поскольку значение ASCII-кода

символа "B" больше, чем кода символа "A", отсюда следует, что "A меньше B"). Поэтому результат действия приведенной выше команды будет выглядеть так, как это представлено на рис. 5.9.

	P_CODE	P_DESCRIPT	P_ONHAND	P_MIN	P_PRICE
▶	10ER/81	Power painter, 15 psi., 3-nozzle	8	5	\$109.99
	13-Q2/P2	7.25-in. pwr. saw blade	32	15	\$14.99
	14-Q1/L3	9.00-in. pwr. saw blade	18	12	\$17.49
	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15	8	\$39.95

Рис. 5.9. Выборка атрибутов из таблицы PRODUCT: результат сравнения ASCII-кодов

Замечание о коде ASCII

Все символы (такие, как буква "T", цифра "8" или знак "+") в компьютере имеют свой код. Ниже представлены ASCII-коды некоторых символов:

Символ	ASCII-код*	Символ	ASCII-код
A	65	*	42
a	97	&	38
B	66	1	49
b	98	2	50
[91	3	51

* Если вам потребуется полная таблица ASCII-кодов, обратитесь к документации на ваш компьютер.

Сравнение строк (символов) осуществляется слева направо. Такой порядок особенно важен при сравнении таких атрибутов, как имена или названия. Например, строка "Ardmore" будет *больше* чем строка "Aargenson", но *меньше*, чем строка "Brown"; такие результаты можно использовать при создании, например, алфавитного списка поставщиков.

Если в виде строк хранятся символы цифр 0—9, то такое сравнение слева направо может стать причиной аномалий. Например, как и ожидается, ASCII-код символа "5" больше кода символа "4". Но в то же время "5" будет больше чем "44", поскольку первый символ в строке "44" меньше символа "5". По этой причине можно получить совершенно неожиданные результаты при сравнении данных, хранящихся в символьном формате. Например, сравнение слева направо ASCII-символов может привести к тому, что дата "01/01/2002" окажется более ранней, чем дата "12/31/2001", поскольку строка "01/01/2002" меньше строки "12/31/2001", т. к. самый левый символ "0" строки "01/01/2002" меньше символа "1" в строке "12/31/2001". Естественно, если строки дат хранятся в формате уууу/мм/дд, то сравнение даст правильный результат, однако не многим понравится столь экзотичное представление даты. Вот почему все современные PCУБД поддерживают тип данных DATE

(дата), который необходимо использовать для хранения именно дат. Кроме того, используя тип данных DATE, вы сможете выполнять над датами арифметические действия.

Математические операции с датами

Обработка дат зачастую сильнее зависит от используемого программного обеспечения, чем какие-либо другие процедуры. Например, если вы используете XDB, OS/2 EE Database Manager или DB2 компании IBM, то запрос на вывод всех строк наименований товаров, полученных после 20 января 2002 года, выглядит так:

```
SELECT P_DESCRPT, P_ONHAND, P_MIN, P_PRICE, P_INDATE
FROM PRODUCT
WHERE P_INDATE >= '01/20/2002';
```

(Помните, что формат даты устанавливается через РСУБД. Пользователи Oracle используют формат "20-JAN-2002", пользователи Access — формат #1/20/02#/.). Результат обработки запроса с ограничением по дате представлен на рис. 5.10.

	P_DESCRPT	P_ONHAND	P_MIN	P_PRICE	P_INDATE
▶	B&D jigsaw, 8-in blade	6	5	\$99.87	24-Jan-02
	Claw hammer	23	10	\$5.95	20-Jan-02
	Sledge hammer, 12 lb.	8	5	\$14.40	02-Feb-02
	PVC pipe, 3.5-in., 8-ft	188	75	\$5.87	20-Jan-02

Рис. 5.10. Выборка атрибутов из таблицы PRODUCT:
ограничение по дате

Вычисляемые столбцы и псевдонимы столбцов в операторе SELECT

Предположим, что вам необходимо определить общую стоимость товаров, находящихся в данный момент на складе. Конечно, такая задача потребует перемножения количества каждого имеющегося в распоряжении товара и его текущей стоимости. Эту задачу можно выполнить с помощью такой команды:

```
SELECT P_DESCRPT, P_ONHAND, P_PRICE, P_ONHAND*P_PRICE
FROM PRODUCT;
```

Если такая команда выполняется в MS Access, то мы получим результат, представленный на рис. 5.11.

В SQL допускаются любые расчеты или формулы для вычисления столбцов. Такие формулы могут содержать любые математические операторы и функции, применяемые к атрибутам в любых таблицах, заданных в объявлении FROM оператора SELECT. Обратите внимание (см. рис. 5.11), что вычисленный столбец помечен как "Expr1". На самом деле, MS Access автоматически добавляет метку Expr# ко всем вычисляемым столбцам. (Первый вычисляемый столбец будет обозначен как Expr1, второй Expr2 и т. д.)

	P_DESCRPT	P_ONHAND	P_PRICE	Expr1
▶	Power painter, 15 psi., 3-nozzle	8	\$109.99	\$879.92
	7.25-in. pwr. saw blade	32	\$14.99	\$479.68
	9.00-in. pwr. saw blade	18	\$17.49	\$314.82
	Hrd. cloth, 1/4-in., 2x50	15	\$39.95	\$599.25
	Hrd. cloth, 1/2-in., 3x50	23	\$43.99	\$1,011.77
	B&D jigsaw, 12-in. blade	8	\$109.92	\$879.36
	B&D jigsaw, 8-in. blade	6	\$99.87	\$599.22
	B&D cordless drill, 1/2-in.	12	\$38.95	\$467.40
	Claw hammer	23	\$5.95	\$136.85
	Sledge hammer, 12 lb.	8	\$14.40	\$115.20
	Rat-tail file, 1/8-in. fine	43	\$4.99	\$214.57
	Hicut chain saw, 16 in.	11	\$256.99	\$2,826.89
	PVC pipe, 3.5-in., 8-ft	188	\$5.87	\$1,103.56
	1.25-in. metal screw, 25	172	\$6.99	\$1,202.28
	2.5-in. wd. screw, 50	237	\$8.45	\$2,002.65
	Steel matting, 4'x8'x1/8", .5" mesh	18	\$119.95	\$2,159.10

Рис. 5.11. Оператор SELECT с вычисляемым столбцом

Необходимо также помнить, что стандарты SQL допускают создание и использование в операторе SELECT псевдонимов для столбцов. *Псевдоним* это альтернативное название, присвоенное столбцу или таблице в каком-либо SQL-операторе с помощью модификатора AS. Например, предыдущий оператор можно записать так:

```
SELECT P_DESCRPT, P_ONHAND, P_PRICE, P_ONHAND*P_PRICE AS TOTVALUE
FROM PRODUCT;
```

Результат выполнения этого оператора представлен на рис. 5.12.

	P_DESCRPT	P_ONHAND	P_PRICE	TOTVALUE
▶	Power painter, 15 psi., 3-nozzle	8	\$109.99	\$879.92
	7.25-in. pwr. saw blade	32	\$14.99	\$479.68
	9.00-in. pwr. saw blade	18	\$17.49	\$314.82
	Hrd. cloth, 1/4-in., 2x50	15	\$39.95	\$599.25
	Hrd. cloth, 1/2-in., 3x50	23	\$43.99	\$1,011.77
	B&D jigsaw, 12-in. blade	8	\$109.92	\$879.36
	B&D jigsaw, 8-in. blade	6	\$99.87	\$599.22
	B&D cordless drill, 1/2-in.	12	\$38.95	\$467.40
	Claw hammer	23	\$5.95	\$136.85
	Sledge hammer, 12 lb.	8	\$14.40	\$115.20
	Rat-tail file, 1/8-in. fine	43	\$4.99	\$214.57
	Hicut chain saw, 16 in.	11	\$256.99	\$2,826.89
	PVC pipe, 3.5-in., 8-ft	188	\$5.87	\$1,103.56
	1.25-in. metal screw, 25	172	\$6.99	\$1,202.28
	2.5-in. wd. screw, 50	237	\$8.45	\$2,002.65
	Steel matting, 4'x8'x1/8", .5" mesh	18	\$119.95	\$2,159.10

Рис. 5.12. Оператор SELECT с вычисляемым столбцом и псевдонимом

5.4.2. Логические операторы: AND, OR и NOT

В SQL допускается использование логических ограничений на запросы. Например, если необходимо вывести список из таблицы, где

V_CODE = 21344 OR V_CODE = 24288

(OR — логическое ИЛИ), можно использовать такую командную последовательность:

```
SELECT P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE V_CODE = 21344
      OR V_CODE = 24288;
```

Будет создан список из шести строк, удовлетворяющий этому логическому ограничению (рис. 5.13).

	P_DESCRPT	P_INDATE	P_PRICE	V_CODE
▶	25-in. pwr. saw blade	13-Jan-02	\$14.99	21344
	9.00-in. pwr. saw blade	13-Jan-02	\$17.49	21344
	B&D jigsaw, 12-in. blade	30-Dec-01	\$109.92	24288
	B&D jigsaw, 8-in. blade	24-Jan-02	\$99.87	24288
	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99	21344
	Hicut chain saw, 16 in.	07-Dec-01	\$256.99	24288

Рис. 5.13. Выборка атрибутов таблицы PRODUCT: логическое ИЛИ

Логический оператор AND (логическое И) имеет похожий синтаксис. Следующая команда создает список всех строк, для которых значение P_PRICE (стоимость) меньше \$50.00 AND (И) для которых значение P_INDATE (дата выписки счета) превышает 5 января 2002 года.

```
SELECT P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE P_PRICE < 50
      AND P_INDATE > '01/05/2002';
```

Результат выполнения этой команды представлен на рис. 5.14.

	P_DESCRPT	P_INDATE	P_PRICE	V_CODE
▶	25-in. pwr. saw blade	13-Jan-02	\$14.99	21344
	9.00-in. pwr. saw blade	13-Jan-02	\$17.49	21344
	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	\$43.99	23119
	Claw hammer	20-Jan-02	\$5.95	21225
	Sledge hammer, 12 lb.	02-Feb-02	\$14.40	
	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99	21344
	PVC pipe, 3.5-in., 8-ft	20-Jan-02	\$5.87	

Рис. 5.14. Выборка атрибутов таблицы PRODUCT: логическое И

Можно комбинировать логическое ИЛИ и логическое И для дополнительного ограничения выборки. Предположим, необходимо вывести таблицу в соответствии со следующими условиями:

- ☐ значение P_INDATE после 5 января 2002 и значение P_PRICE меньше \$50.00;
- ☐ или значение V_CODE равно 21288.

Необходимую распечатку можно получить с помощью такой команды:

```
SELECT P_DESCRPT, P_INDATE, P_PRICE, V_CODE
FROM PRODUCT
WHERE (P_PRICE < 50 AND P_INDATE > '01/05/2002')
OR V_CODE = 24288;
```

Обратите внимание на использование скобок для группировки логических операций. От того, как вы поставите скобки, зависит, как будут выполняться логические операции. Условия, перечисленные внутри скобок, всегда выполняются в первую очередь. Вы, конечно, помните со школы правило старшинства арифметических действий: $2 + (3 \times 5) = 17$, поскольку $3 \times 5 = 15$, а $2 + 15 = 17$.

Если в запросе имеется несколько ограничений, то в применении операций OR и AND достаточно просто запутаться. На самом деле, использованием логических операций занимается специальный раздел математики, называемый *булевой алгеброй* (алгеброй логики).

Результат выполнения предыдущего запроса представлен на рис. 5.15.

	P_DESCRPT	P_INDATE	P_PRICE	V_CODE
▶	25-in. pwr. saw blade	13-Jan-02	\$14.99	21344
▶	9.00-in. pwr. saw blade	13-Jan-02	\$17.49	21344
	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	\$43.99	23119
	B&D jigsaw, 12-in. blade	30-Dec-01	\$109.92	24288
	B&D jigsaw, 8-in. blade	24-Jan-02	\$99.87	24288
	Claw hammer	20-Jan-02	\$5.95	21225
	Sledge hammer, 12 lb.	02-Feb-02	\$14.40	
	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99	21344
	Hicut chain saw, 16 in.	07-Dec-01	\$256.99	24288
	PVC pipe, 3.5-in., 8-ft	20-Jan-02	\$5.87	

Рис. 5.15. Выборка атрибутов таблицы PRODUCT: логические И и ИЛИ

Обратите внимание, что в распечатку включены строки, где V_CODE = 24288, несмотря на значения атрибутов P_INDATE и P_PRICE.

Если ваша версия SQL поддерживает логический оператор NOT, (логическое НЕ), то его синтаксис точно такой же, как и у операторов OR и AND. Например, если вы хотите получить список всех строк, для которых код поставщика не равен 21344, то можно использовать такую командную последовательность:

```
SELECT *
FROM PRODUCT
WHERE V_CODE NOT 21344;
```


Логическое НЕ (оператор NOT) можно применять совместно с операторами AND и OR.

Примечание

Если ваша версия SQL не поддерживает логическое НЕ, то такого же результата можно добиться с помощью оператора условия:

```
WHERE V_CODE <> 21344
```

Если ваша версия SQL не поддерживает <>, то можно использовать такое выражение:

```
WHERE V_CODE != 21344
```

5.4.3. Специальные операторы

ANSI SQL допускает использование специальных операторов (модификаторов) в объявлении WHERE. К таким специальным операторам относятся:

- BETWEEN — задание диапазона значений;
- IS NULL — проверка на пустое значение атрибута;
- LIKE — проверка подобия символьных строк;
- IN — проверка присутствия значения атрибута в перечисленном наборе;
- EXIST — проверка того, имеет ли атрибут какое-то значение. В сущности, этот оператор противоположен оператору IS NULL.

Рассмотрим эти модифицирующие операторы подробнее. Если ваше программное обеспечение реализует стандартный SQL, то для определения границ значений атрибута можно использовать оператор BETWEEN. Например, если необходимо посмотреть список всех товаров, цена которых находится в диапазоне от \$50.00 до \$100.00, то можно использовать такую команду:

```
SELECT *  
FROM PRODUCT  
WHERE P_PRICE BETWEEN 50.00 AND 100.00;
```

Если PCYБД не поддерживает оператор BETWEEN, то можно поступить следующим образом:

```
SELECT *  
FROM PRODUCT  
WHERE P_PRICE > 50.00  
AND P_PRICE < 100.00;
```

Стандартный SQL допускает использовать оператор IS NULL для проверки, не имеет ли атрибут пустых значений. Например, пусть для одного или более товаров в таблице PRODUCT значение P_MIN не определено. Такие пустые значения могут быть найдены с помощью следующей команды:

```
SELECT P_CODE, P_DESCRIPT  
FROM PRODUCT  
WHERE V_CODE IS NULL;
```

Точно так же, если захотите проверить наличие пустых значений дат, в стандартном SQL можно выполнить такую команду:

```
SELECT P_CODE, P_DESCRIPT
FROM PRODUCT
WHERE P_INDATE IS NULL;
```

SQL допускает использование метасимволов для поиска совпадений, когда строка целиком неизвестна:

□ % (процент) — любое количество последующих символов. Например:

- 'J%' включает в себя Johnson, Jones, Jemigan, July и J-231Q;
- 'Jo%' включает в себя Johnson и Jones;

□ _ (подчеркивание) — один любой символ. Например:

- '_23-456-6789' включает в себя 123-456-6789, 223-456-6789 и 323-456-6789;
- '_23-_56-678_' включает в себя 123-156-6781, 123-256-6782 и 823-956-6788;
- '_o_es' включает в себя Jones, Cones, Cokes, totes и roles.

Примечание

В некоторых РСУБД, например, в MS Access, вместо метасимволов % и _ используются метасимволы * и ?. В большинстве реализаций SQL также обеспечивается поиск с учетом регистра. РСУБД Oracle не выведет строку, содержащую 'Jones', если при поиске фамилии использовать шаблон 'jo%', поскольку 'Jones' начинается с прописной буквы, а шаблон поиска — со строчной.

Метасимволы можно использовать в сочетании друг с другом. Например, поиск по шаблону '_l%' может дать результаты Al, Alton, Elgin, Blakeston, blank, bloated и eligible.

Метасимволы можно использовать в операторе LIKE. Например, в результате следующего запроса будут выведены все строки таблицы VENDOR для контактных лиц, фамилии которых начинаются со Smith.

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE V_CONTACT LIKE 'Smith%';
```

Если вы еще раз заглянете в исходную таблицу VENDOR на рис. 5.2, то сможете заметить, что приведенный только что SQL-запрос выдаст три записи: две для Smith и одну для Smithson.

Предположим, что мы выполняем такую SQL-команду:

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE V_CONTACT LIKE 'SMITH%';
```

В результате не будет выведено ни одной строки. Что произошло?

А произошло то, что запросы, связанные с поиском строк и символов, могут быть чувствительны к регистру, поскольку ASCII-код символа в верхнем регистре отлича-

ется от кода того же символа в нижнем регистре, т. е. варианты 'SMITH', 'Smith' и 'smith' считаются различными (не идентичными). Поскольку в таблице нет поставщиков, фамилии которых начинаются с заглавных букв 'SMITH' (все буквы заглавные), использованных в шаблоне поиска 'SMITH%', не было найдено ни одной строки. Соответствия будут обнаружены, если запрос записан корректно.

Некоторые РСУБД, например, XDB и MS Access, автоматически выполняют преобразование для устранения зависимости от регистра. Другие, например, Oracle, предоставляют специальную функцию UPPER для конвертирования символов таблиц и запросов в верхний регистр. (Конвертирование выполняется только в памяти компьютера и не воздействует на фактический формат хранения данных в таблице.) Поэтому если вы хотите избежать проблем с зависимостью от регистра, и если ваша СУРБД позволяет использовать функцию UPPER, то можно получить правильные результаты с помощью такого запроса:

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE UPPER(V_CONTACT) LIKE 'SMITH%';
```

В результате выполнения только что приведенного запроса будет получен список, в который включены все строки, содержащие фамилии, начинающиеся на 'Smith', независимо от регистра ('Smith', 'smith', 'SMITH' и т. п.).

Естественно, в сочетании с метасимволами можно использовать и логические операторы. Например, в результате выполнения запроса

```
SELECT V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM VENDOR
WHERE V_CONTACT NOT LIKE 'Smith%';
```

будет получен список всех поставщиков, имя которых не начинается со Smith.

Предположим, что вы не знаете, как пишется фамилия — Johnson или Johnsen. Метасимвол () позволит найти совпадения для любого написания. Правильный поиск в этом случае можно выполнить с помощью такого запроса:

```
SELECT *
FROM VENDOR
WHERE V_CONTACT LIKE 'Johns_n'
```

Метасимволы, таким образом, позволяют отыскивать совпадения даже в том случае, если написание объекта известно только приблизительно.

Многие запросы, в которых используется оператор OR, проще построить с помощью специального оператора IN. Например, запрос

```
SELECT *
FROM PRODUCT
WHERE V_CODE = 21344
OR V_CODE = 24288;
```

может быть записан в следующей форме:

```
SELECT *  
FROM PRODUCT  
WHERE V_CODE IN (21344, 24288);
```

Если атрибут определяет множество символов, то в определении подмножества нужно использовать апострофы. Например, если при определении таблицы тип данных атрибута V_CODE определен, как CHAR(5), то предыдущий запрос следует записать иначе:

```
SELECT *  
FROM PRODUCT  
WHERE V_CODE IN ('21344', '24288');
```

Оператор EXISTS можно использовать всюду, где требуется выполнить команду для атрибута, имеющего не пустое значение. Например, определенные строки таблицы PRODUCT можно удалить с помощью такой команды:

```
DELETE FROM PRODUCT  
WHERE P_CODE EXISTS;
```

Если необходимо вывести список всех строк, в которых код поставщика не пустой, то можно выполнить такой запрос:

```
SELECT *  
FROM PRODUCT  
WHERE V_CODE EXISTS;
```

Оператор EXISTS поддерживается не во всех СУБД. Если РСУБД не поддерживает оператор EXISTS, то предыдущий запрос можно записать в таком виде:

```
SELECT *  
FROM PRODUCT  
WHERE V_CODE IS NOT NULL;
```

5.5. Дополнительные команды управления данными

В разд. 5.2 мы обсудили создание табличных структур, а также ввод, модификацию, удаление и распечатку информации. В этом разделе мы исследуем способы изменения структуры таблиц при помощи изменения свойств атрибутов и добавления новых столбцов, а в заключение рассмотрим способы копирования таблицы (или части таблицы) и их удаления.

Все изменения в структуре таблицы выполняются с помощью оператора ALTER, который размещается за ключевым словом, указывающим на характер производимых изменений. Оператор допускает два режима: MODIFY и ADD. Режим MODIFY позволяет изменять свойства столбца, а в режиме ADD можно добавлять столбцы. В большинстве РСУБД не допускается удаление столбцов, поскольку

при этом может быть удалена критически важная информация, которая используется в других таблицах.

Синтаксис команды ALTER может быть представлен в двух форматах:

```
ALTER TABLE <имя таблицы>
```

```
MODIFY (<имя столбца> <новые свойства столбца>);
```

или

```
ALTER TABLE <имя таблицы>
```

```
ADD (<имя нового столбца> <свойства нового столбца>);
```

5.5.1. Изменение типа данных столбца

С помощью только что представленной команды ALTER тип данных атрибута V_CODE (INTEGER) таблицы PRODUCT можно изменить на символьный (CHAR):

```
ALTER TABLE PRODUCT
```

```
MODIFY (V_CODE CHAR(5));
```

Сложные РСУБД (например, Oracle) не позволяют изменять тип данных, пока в столбце содержатся какие-нибудь данные. Например, если вы захотите изменить тип поля V_CODE с целого на символьный, то при выполнении команды

```
ALTER TABLE PRODUCT
```

```
MODIFY (V_CODE CHAR(5));
```

будет выдано сообщение об ошибке, поскольку в столбце V_CODE содержатся данные. Появление такого сообщения легко объяснимо. Если вы меняете тип данных V_CODE, то вы должны помнить, что атрибут V_CODE таблицы PRODUCT ссылается на атрибут V_CODE таблицы VENDOR. Если типы данных не соответствуют друг другу, то будет иметь место нарушение целостности на уровне ссылок, что и приведет к сообщению об ошибке. Если в столбце V_CODE отсутствуют данные, то указанная команда произведет ожидаемые изменения в структуре в том случае, если в процессе создания таблицы PRODUCT этот атрибут не был определен как внешний ключ.

5.5.2. Изменение свойств атрибута

Если столбец, который необходимо изменить, содержит данные, то вы можете изменить любые свойства столбца, не затрагивающие тип данных. Например, если вы захотите увеличить ширину столбца P_PRICE до 9 цифр, то выполните такую команду:

```
ALTER TABLE PRODUCT
```

```
MODIFY (P_PRICE DECIMAL(9,2));
```

Если теперь вновь вывести содержимое таблицы, то можно заметить, что ширина столбца P_PRICE увеличилась на одну цифру. (Помните, что в вашей версии SQL скобки могут и не использоваться.)

Примечание

Некоторые РСУБД могут устанавливать ограничения на изменение свойств атрибута. Например, при выполнении предыдущего оператора SQL в Oracle8i может возникнуть ошибка в том случае, если в атрибуте P_PRICE таблицы PRODUCT содержатся какие-либо значения. Причина такого ограничения в том, что изменение свойств атрибута может повлиять на целостность данных в БД. Фактически некоторые изменения атрибутов можно выполнить только при отсутствии каких-либо данных в строках, на которые влияет данный атрибут.

5.5.3. Добавление столбцов

Существующую таблицу можно изменить путем добавления одного или более столбцов. В следующем примере в таблицу PRODUCT будет добавлен столбец с именем P_SALECODE. (Мы впоследствии будем использовать данные этого столбца для определения длительности хранения товара на складе с тем, чтобы назначить для него специальную цену.)

Предположим, что предполагаемые значения P_SALECODE — 1, 2 или 3. Поскольку мы не собираемся выполнять арифметические действия над этим атрибутом, для него можно использовать символьный тип. Вся эта информация нашла отражение в следующей команде:

```
ALTER TABLE PRODUCT  
ADD (P_SALECODE CHAR(1));
```

При добавлении какого-либо столбца не стоит включать объявление NOT NULL. Если это сделать, то возникнет сообщение об ошибке. Причина в том, что при добавлении нового столбца в таблице уже имеются значащие строки, и это приведет к тому, что в новый столбец будет иметь пустые значения в этих строках. Поэтому при добавлении нового столбца объявление NOT NULL использовать нельзя.

5.5.4. Удаление столбцов

Возможен случай, когда вам необходимо удалить имеющиеся столбцы. Предположим, что из таблицы VENDOR необходимо удалить атрибут V_ORDER. Для этого можно использовать такую команду:

```
ALTER TABLE VENDOR  
DROP COLUMN V_ORDER;
```

Опять-таки, некоторые РСУБД накладывают ограничения на удаление столбцов. Например, вы не сможете удалить атрибуты, которые входят в состав внешнего ключа, или удалить атрибут из таблицы, которая состоит из одного атрибута.

5.5.5. Занесение данных в новый столбец

Для занесения информации в строки столбца в SQL имеется оператор UPDATE. Команда INSERT, представленная ранее, не может определить местоположение отдельного элемента столбца; нужно или вводить всю строку целиком или не вводить

ничего. Например, чтобы ввести значение "2" для P_SALECODE в четвертой строке, можно использовать команду UPDATE со значением первичного ключа P_CODE '1546-QQ2'. (Убедитесь, что в исходной таблице P_CODE = '1546-QQ2' соответствует четвертой строке!) Команда обновления выглядит так:

```
UPDATE PRODUCT
  SET P_SALECODE = '2'
  WHERE P_CODE = '1546-QQ2';
```

Результат действия этой команды представлен на рис. 5.16.

	P_CODE	P_DESCRIPT	P_INDATE	P_PRICE	P_SALECODE
▶	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	\$39.95	2

Рис. 5.16. Выборка атрибутов таблицы PRODUCT:
занесение данных в новый атрибут P_SALECODE

Последующие данные можно занести таким же способом, определяя местоположение с помощью первичного ключа (P_CODE) и столбца (P_SALECODE). Например, если мы хотим ввести для P_SALECODE значение "1" при значениях первичного ключа P_CODE '2232/QWE' и '2232/QTY', то можно использовать такую команду:

```
UPDATE PRODUCT
  SET P_SALECODE = '1'
  WHERE P_CODE IN ('2232/QWE', '2232/QTY');
```

Если ваша РСУБД не поддерживает объявление IN, то можно выполнить две команды:

```
UPDATE PRODUCT
  SET P_SALECODE = '1'
  WHERE P_CODE = '2232/QWE';
```

```
UPDATE PRODUCT
  SET P_SALECODE = '1'
  WHERE P_CODE = '2232/QTY';
```

Результат выполнения только что приведенных команд можно проверить с помощью такого запроса:

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE, P_SALECODE
  FROM PRODUCT;
```

Выборка, полученная с помощью этого запроса, представлена на рис. 5.17.

Процесс занесения данных в определенные ячейки таблицы с помощью оператора UPDATE на самом деле очень неудобен. К счастью, если можно установить связь между элементами ввода и существующими столбцами, то эту операцию можно использовать для присвоения значений всем соответствующим сегментам. Предполю-

жим, что необходимо разместить в таблице коды продаж, основанные на P_INDATE, с помощью такого расписания:

P_INDATE	P_SALECODE
до 25 декабря 2001	2
между 16 января и 21 января 2002	1

	P_CODE	P_DESCRIPTION	P_INDATE	P_PRICE	P_SALECODE
▶	1QER31	Power painter, 15 psi., 3-nozzle	03-Dec-01	\$109.99	
	13-Q2/P2	7.25-in. pwr. saw blade	13-Jan-02	\$14.99	
	14-Q1/L3	9.00-in. pwr. saw blade	13-Jan-02	\$17.49	
	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	\$39.95	2
	1558-QVW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	\$43.99	
	2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-01	\$109.92	1
	2232/QVW	B&D jigsaw, 8-in. blade	24-Jan-02	\$99.87	1
	2238/QPD	B&D cordless drill, 1/2-in.	20-Nov-01	\$38.95	
	23109-HB	Claw hammer	20-Jan-02	\$5.95	
	23114-AA	Sledge hammer, 12 lb.	02-Feb-02	\$14.40	
	54778-2T	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99	
	89-VVRE-Q	Hicut chain saw, 16 in.	07-Dec-01	\$256.99	
	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Jan-02	\$5.87	
	SM-18277	1.25-in. metal screw, 25	29-Dec-01	\$6.99	
	SW-23116	2.5-in. wd. screw, 50	24-Dec-01	\$8.45	
	VR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	17-Jan-02	\$119.95	

Рис. 5.17. Выборка атрибутов таблицы PRODUCT: занесение данных

Тогда это можно выполнить с помощью таких команд:

```
UPDATE PRODUCT
```

```
SET P_SALECODE = '2'
WHERE P_INDATE < '12/25/2001';
```

```
UPDATE PRODUCT
```

```
SET P_SALECODE = '1'
WHERE P_INDATE >= '01/16/2002'
AND P_INDATE < '01/21/2002';
```

Для проверки результатов действия этих команд можно выполнить такой запрос:

```
SELECT P_CODE, P_DESCRIPTION, P_INDATE, P_PRICE, P_SALECODE
FROM PRODUCT;
```

Результаты запроса представлены на рис. 5.18.

Все изменения необходимо зафиксировать с помощью команды COMMIT.

	P_CODE	P_DESCRIPTION	P_INDATE	P_PRICE	P_SALECODE
▶	11QER/31	Power painter, 15 psi., 3-nozzle	03-Dec-01	\$109.99	2
	13-Q2/P2	7.25-in. pwr. saw blade	13-Jan-02	\$14.99	
	14-Q1/L3	9.00-in. pwr. saw blade	13-Jan-02	\$17.49	
	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	\$39.95	2
	1558-QVW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	\$43.99	
	2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-01	\$109.92	1
	2232/QWE	B&D jigsaw, 8-in. blade	24-Jan-02	\$99.87	1
	2238/QPD	B&D cordless drill, 1/2-in.	20-Nov-01	\$38.95	2
	23109-HB	Claw hammer	20-Jan-02	\$5.95	1
	23114-AA	Sledge hammer, 12 lb.	02-Feb-02	\$14.40	
	54778-2T	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99	
	89-WRE-Q	Hicut chain saw, 16 in.	07-Dec-01	\$256.99	2
	PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Jan-02	\$5.87	1
	SM-18277	1.25-in. metal screw, 25	29-Dec-01	\$6.99	
	SW-23116	2.5-in. wd. screw, 50	24-Dec-01	\$8.45	2
	VR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-02	\$119.95	1

Рис. 5.18. Выборка атрибутов таблицы PRODUCT: многократное обновление

Примечание

Помните, что эти изменения в P_SALECODE выполнены в дополнение к изменениям, сделанным ранее для кодов товара 2232/QTY, 2232/QWE и 1546-QQ2. Вот почему в представленной только что распечатке для кодов товара 2232/QTY и 2232/QWE код продаж равен 1, а для товара 1546-QQ2 — 2, несмотря на то, что эти товары не попадают в диапазон дат P_INDATE, определенный в двух последних командах.

5.5.6. Арифметические операторы и правило старшинства

Команда SQL UPDATE часто используется в сочетании с арифметическими операторами, представленными в табл. 5.5.

Таблица 5.5. Арифметические операторы

Арифметический оператор	Описание
+	Сложение
-	Вычитание
*	Умножение
/	Деление
^	Возведение в степень

Не путайте символ умножения (*) с метасимволом (*), применяемым в некоторых реализациях SQL (например, в MS Access); последний используется только в строках сравнения, в то время как первый — в контексте математических вычислений.

Необходимо учитывать основные правила старшинства арифметических операций. Вы должны помнить, что если операции не заключены в скобки, то операция возведения в степень предшествует операциям умножения и деления, которые в свою очередь предшествуют операциям сложения и вычитания. Например, $8 + 2 * 5 = 8 + 10 = 18$, но $(8 + 2) * 5 = 10 * 5 = 50$. Точно так же $4 + 5 ^ 2 * 3 = 4 + 25 * 3 = 79$, но $(4 + 5) ^ 2 * 3 = 81 * 3 = 243$, в то время как при выполнении операций $(4 + 5 ^ 2) * 3$ получается ответ $(4 + 25) * 3 = 29 * 3 = 87$.

Арифметические операторы применяются при обновлении данных. Например, если количество товара на складе (таблица PRODUCT) становится ниже определенного минимального уровня, то необходимо заказать у поставщика некоторое количество этого товара. Предположим, например, что вы заказали 20 единиц товара 2232/QWE. Когда эти 20 единиц поступят на склад, их необходимо будет добавить в список:

```
UPDATE PRODUCT
```

```
SET P_ONHAND = P_ONHAND + 20  
WHERE P_CODE = '2232/QWE';
```

Если вы пожелаете добавить 10% к стоимости всех товаров, стоимость которых ниже \$50, то можно использовать такую команду:

```
UPDATE PRODUCT
```

```
SET P_PRICE = P_PRICE*1.10  
WHERE P_PRICE < 50.00;
```

5.5.7. Копирование части таблицы

В последующих главах при проектировании БД будет показано, что может возникнуть необходимость разбиения таблицы на несколько частей. К счастью, SQL позволяет копировать содержимое отдельных столбцов таблицы, чтобы данные во вновь созданную таблицу (или таблицы) не пришлось вновь вводить вручную. Например, если вы хотите скопировать атрибуты P_DESCRIPT и P_PRICE из таблицы PRODUCT в новую таблицу с названием PART, прежде всего создайте структуру таблицы PART с помощью такого оператора:

```
CREATE TABLE PART (
```

```
PART_CODE CHAR(8)          NOT NULL          UNIQUE,  
PART_DESCRIPT             CHAR(35),  
PART_PRICE                 DECIMAL(8,2),  
PRIMARY KEY (PART_CODE));
```

Обратите внимание, что имена столбцов таблицы PART не идентичны именам столбцов в исходной таблице, и что число столбцов в новой таблице не обязательно совпадает с числом столбцов в исходной таблице. В данном случае первый столбец таблицы PART называется PART_CODE, в отличие от P_CODE в исходной таблице PRODUCT. В таблице PART имеются всего лишь три столбца, а не семь, как в ис-

ходной таблице. Однако свойства столбцов совпадают: вы не можете копировать символьные атрибуты в числовые структуры и наоборот.

Теперь можно скопировать соответствующие столбцы таблицы **PRODUCT** во вновь созданную таблицу **PART** с помощью такой командной последовательности:

```
INSERT INTO <таблица-получатель> <имена столбцов таблицы-получателя>
SELECT <имена столбцов, предназначенных для копирования>
FROM <имя таблицы-источника>;
```

В нашем случае эта команда будет выглядеть так:

```
INSERT INTO PART (PART_CODE, PART_DESCRIPT, PART_PRICE)
SELECT P_CODE, P_DESCRIPT, P_PRICE FROM PRODUCT;
```

Проверить содержимое таблицы **PART** можно с помощью запроса

```
SELECT *
FROM PART;
```

Результат выполнения этого запроса представлен на рис. 5.19.

	PART_CODE	PART_DESCRIPT	PART_PRICE
▶	11QER31	Power painter, 15 psi., 3-nozzle	\$109.99
	13-Q2/P2	7.25-in. pwr. saw blade	\$14.99
	14-Q1/L3	9.00-in. pwr. saw blade	\$17.49
	1546-QQ2	Hrd. cloth, 1/4-in., 2x50	\$39.95
	1558-QWM	Hrd. cloth, 1/2-in., 3x50	\$43.99
	2232/QTY	B&D jigsaw, 12-in. blade	\$109.92
	2232/QWE	B&D jigsaw, 8-in. blade	\$99.87
	2238/QPD	B&D cordless drill, 1/2-in.	\$38.95
	23109-HB	Claw hammer	\$5.95
	23114-AA	Sledge hammer, 12 lb.	\$14.40
	54778-2T	Rat-tail file, 1/8-in. fine	\$4.99
	89-WRE-Q	Hicut chain saw, 16 in.	\$256.99
	PVC23DRT	PVC pipe, 3.5-in., 8-ft	\$5.87
	SM-18277	1.25-in. metal screw, 25	\$6.99
	SW-23116	2.5-in. wd. screw, 50	\$8.45
	WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	\$119.95

Рис. 5.19. Атрибуты таблицы **PART**, скопированные из таблицы **PRODUCT**

В гл. 4 мы отмечали, что возможность копирования части таблицы в процессе нормализации имеет очень большое значение.

Примечание

Мы советуем сделать копию таблицы **PRODUCT** и присвоить ей имя **PRODUCT_2**. Прежде всего, создайте табличную структуру **PRODUCT_2** (с помощью команды **CREATE**), затем скопируйте в нее все столбцы таблицы **PRODUCT** с помощью команды **INSERT**. По-

сле этого вы можете практиковаться в использовании SQL на любой из этих таблиц, и если в одной из них вы сохраните нежелательные результаты, то сможете восстановить содержимое с помощью другой таблицы.

5.5.8. Удаление таблицы из базы данных

Таблицу можно удалить из БД с помощью оператора DROP. Например, мы можем удалить только что созданную таблицу PART с помощью такой команды:

```
DROP TABLE PART;
```

5.5.9. Обозначение первичных и внешних ключей

При создании таблиц PRODUCT и VENDOR в *разд. 5.2.4* мы декларировали первичные и внешние ключи, чтобы обеспечить соблюдение правил целостности. Но предположим, что на этапе создания таблиц мы забыли определить первичные и внешние ключи. Или могло случиться так, что мы импортировали таблицы из другой БД, и при этом соблюдение правил целостности не обеспечивалось. В любом случае вы можете переустановить правила целостности с помощью команды ALTER. Например, если первичный ключ таблицы PRODUCT еще не определен, то можно выполнить команду:

```
ALTER TABLE PRODUCT  
  ADD PRIMARY KEY (P_CODE);
```

Если в таблице PRODUCT еще не определен внешний ключ, то это можно сделать с помощью такой команды:

```
ALTER TABLE PRODUCT  
  ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Если в таблице PRODUCT не определены ни первичный ключ, ни внешний ключ, то это можно сделать с помощью одной команды:

```
ALTER TABLE PRODUCT  
  ADD PRIMARY KEY (P_CODE)  
  ADD FOREIGN KEY (V_CODE) REFERENCES VENDOR;
```

Одной командой можно назначить даже составные первичные ключи, а также несколько внешних ключей. Например, если необходимо обеспечить выполнение правил целостности для таблицы LINE из *гл. 2*, то можно выполнить такую команду:

```
ALTER TABLE LINE  
  ADD PRIMARY KEY (INV_NUMBER, LINE_NUMBER)  
  ADD FOREIGN KEY (INV_NUMBER) REFERENCES INVOICE  
  ADD FOREIGN KEY (PROD_CODE) REFERENCES PRODUCT;
```

5.6. Более сложные запросы и функции SQL

Одно из самых важных преимуществ SQL состоит в том, что этот язык допускает создавать сложные пользовательские запросы в свободном формате. Логические операторы, которые мы использовали для модификации содержимого таблиц, можно использовать и в запросах. Язык SQL предоставляет также множество полезных функций, с помощью которых можно производить вычисления, находить максимум и минимум, рассчитывать среднее значение и т. д. Более того, в SQL допускается ограничивать запросы только теми элементами, которые не дублируются, или элементами, дубликаты которых могут быть сгруппированы.

5.6.1. Упорядочивание списка

Выражение **ORDER BY** особенно полезно в том случае, если порядок следования строк в списке для вас имеет существенное значение. Команда вида

```
ORDER BY <атрибуты>
```

выводит список, упорядоченный по возрастанию. Для получения списка, упорядоченного по убыванию, используется такой формат команды:

```
ORDER BY <атрибуты> DESC
```

Например, если нужно распечатать содержимое таблицы **PRODUCT** в порядке возрастания значения атрибута **P_PRICE**, то можно выполнить такую команду:

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE  
FROM PRODUCT  
ORDER BY P_PRICE;
```

Результат действия этой команды представлен на рис. 5.20. Обратите внимание, что команда **ORDER BY** упорядочила список по возрастанию цены.

Сравните эту распечатку с фактической таблицей, представленной на рис. 5.2, и обратите внимание, чем отличается рис. 5.20: самые дешевые товары находятся в начале таблицы, а затем они расположены по возрастанию стоимости. Не забывайте о том, что процедура **ORDER BY** является логическим действием, реальное содержимое таблицы остается неизменным.

Для получения списка в порядке убывания необходимо выполнить такую команду:

```
SELECT P_CODE, P_DESCRIPT, P_INDATE, P_PRICE  
FROM PRODUCT  
ORDER BY P_PRICE DESC;
```

Предположим, что необходимо создать телефонный справочник. Хорошо, если этот справочник будет упорядочен на трех уровнях — по фамилии, имени и отчеству (инициалу).

1. Упорядочивание (**ORDER BY**) по фамилии.
2. Последующее упорядочивание (**ORDER BY**) по имени.
3. Последующее упорядочивание (**ORDER BY**) по отчеству (инициалу).

P_CODE	P_DESCRIPT	P_INDATE	P_PRICE
54778-21	Rat-tail file, 1/8-in. fine	15-Jan-02	\$4.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Jan-02	\$5.87
23109-NB	Claw hammer	20-Jan-02	\$5.95
SM-18277	1.25-in. metal screw, 25	29-Dec-01	\$6.99
SW-23116	2.5-in. wvd. screw, 50	24-Dec-01	\$8.45
23114-AA	Sledge hammer, 12 lb.	02-Feb-02	\$14.40
13-Q2/P2	7.25-in. pwr. saw blade	13-Jan-02	\$14.99
14-Q1/L3	9.00-in. pwr. saw blade	13-Jan-02	\$17.49
2238/QPD	B&D cordless drill, 1/2-in.	20-Nov-01	\$38.95
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	\$39.95
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	\$43.99
2232/QWE	B&D jigsaw, 8-in. blade	24-Jan-02	\$99.87
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-01	\$109.92
11QER/31	Power painter, 15 psi., 3-nozzle	03-Dec-01	\$109.99
WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	17-Jan-02	\$119.95
89-WRE-Q	Hicut chain saw, 16 in.	07-Dec-01	\$256.99

Рис. 5.20. Выборка атрибутов таблицы PRODUCT: упорядочивание по возрастанию атрибута P_PRICE

Такой многоуровневый процесс называется *каскадным упорядочиванием* и выполняется простым перечислением атрибутов, расположенных после оператора ORDER BY и разделенных запятыми:

ORDER BY <атрибут 1, атрибут 2, ...>

(Термин "каскадный" используется для того, чтобы подчеркнуть ступенчатый характер процесса логической сортировки.) Например, пусть требуется создать телефонный справочник сотрудников, данные по которым представлены в таблице EMPLOYEE базы данных CHARTER (гл. 2, задача 30). Для удобства фрагмент таблицы EMPLOYEE представлен на рис. 5.21.

С учетом содержимого таблицы EMPLOYEE команда SQL

```
SELECT EMP_LNAME, EMP_FNAME, EMP_INITIAL, EMP_AREACODE, EMP_PHONE
FROM EMPLOYEE
ORDER BY EMP_LNAME, EMP_FNAME, EMP_INITIAL;
```

приведет к результату, который представлен на рис. 5.22.

Оператор ORDER BY используется во многих приложениях, особенно из-за наличия модификатора DESC. Например, можно составить список счетов по датам в убывающем порядке, проконтролировать бюджет подразделения или исследовать любое число финансовых сделок.

Выражение ORDER BY можно использовать в сочетании с другими SQL-командами. Например, взгляните, как реализованы ограничения на дату и стоимость в следующей командной последовательности:

```
SELECT P_DESCRIPT, V_CODE, P_INDATE, P_PRICE
FROM PRODUCT
```

```

WHERE P_INDATE < '01/21/2002'
AND P_PRICE <= 50.00
ORDER BY V_CODE, P_PRICE DESC;

```

Результат действия этой команды представлен на рис. 5.23.

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
100	Mr.	Kolmycz	George	D	615	324-5456
101	Ms.	Lewis	Rhonda	G	615	324-4472
102	Mr.	Vandam	Rhett		901	675-8993
103	Ms.	Jones	Anne	M	615	898-3456
104	Mr.	Lange	John	P	901	504-4430
105	Mr.	Williams	Robert	D	615	890-3220
106	Mrs.	Smith	Jeanine	K	615	324-7883
107	Mr.	Diante	Jorge	D	615	890-4567
108	Mr.	Wiesenbach	Paul	R	615	897-4358
109	Mr.	Smith	George	K	901	504-3339
110	Mrs.	Genkazi	Leighla	W	901	569-0093
111	Mr.	Washington	Rupert	E	615	890-4925
112	Mr.	Johnson	Edward	E	615	898-4387
113	Ms.	Smythe	Melanie	P	615	324-9006
114	Ms.	Brandon	Marie	G	901	882-0845
115	Mrs.	Saranda	Hermine	R	615	324-5505
116	Mr.	Smith	George	A	615	890-2984

Рис. 5.21. Фрагмент таблицы EMPLOYEE

EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_AREACODE	EMP_PHONE
Brandon	Marie	G	901	882-0845
Diante	Jorge	D	615	890-4567
Genkazi	Leighla	W	901	569-0093
Johnson	Edward	E	615	898-4387
Jones	Anne	M	615	898-3456
Kolmycz	George	D	615	324-5456
Lange	John	P	901	504-4430
Lewis	Rhonda	G	615	324-4472
Saranda	Hermine	R	615	324-5505
Smith	George	A	615	890-2984
Smith	George	K	901	504-3339
Smith	Jeanine	K	615	324-7883
Smythe	Melanie	P	615	324-9006
Vandam	Rhett		901	675-8993
Washington	Rupert	E	615	890-4925
Wiesenbach	Paul	R	615	897-4358
Williams	Robert	D	615	890-3220

Рис. 5.22. Результаты выполнения запроса по созданию списка телефонов

	P_DESCRPT	V_CODE	P_INDATE	P_PRICE
►	3&D cordless drill, 1/2-in.	25595	20-Nov-01	\$38.95
	Hrd. cloth, 1/2-in., 3x50	23119	15-Jan-02	\$43.99
	Hrd. cloth, 1/4-in., 2x50	23119	15-Dec-01	\$39.95
	9.00-in. pwr. saw blade	21344	13-Jan-02	\$17.49
	7.25-in. pwr. saw blade	21344	13-Jan-02	\$14.99
	Rat-tail file, 1/8-in. fine	21344	15-Jan-02	\$4.99
	2.5-in. wd. screw, 50	21231	24-Dec-01	\$8.45
	1.25-in. metal screw, 25	21225	29-Dec-01	\$6.99
	Claw hammer	21225	20-Jan-02	\$5.95
	PVC pipe, 3.5-in., 8-ft		20-Jan-02	\$5.87

Рис. 5.23. Запрос, основанный на множественных ограничениях

Обратите внимание, что значения P_PRICE для V_CODE = 23119 представлены в порядке убывания от \$43.99 до \$39.95, в то время как значения P_PRICE для V_CODE = 21344 показаны в порядке убывания от \$17.49 до \$4.99. Цены перечислены в убывающем порядке в рамках одного значения V_CODE, в рамках другого значения V_CODE они тоже расположены в порядке убывания.

Примечание

- Если используется выражение ORDER BY, то пустые значения перечисляются в первую очередь независимо от порядка следования (убывающего или возрастающего).
- Выражение ORDER BY должно всегда быть последним в командной последовательности.

5.6.2. Вывод уникальных значений

Сколько различных поставщиков представлено в настоящий момент в таблице PRODUCT? Простой запрос (SELECT) в данном случае не поможет, особенно если в таблице содержится несколько тысяч строк, и придется отсеивать одинаковые коды поставщиков вручную. К счастью, оператор SQL DISTINCT предназначен для получения списка только тех значений, которые отличаются один от другого. Например, с помощью команды

```
SELECT DISTINCT V_CODE
FROM PRODUCT;
```

можно вывести только отличающиеся (distinct) коды поставщиков (V_CODE) в таблице PRODUCT, как это представлено на рис. 5.24. (Обратите внимание, что в первой строке распечатки представлено пустое значение.)

Примечание

Oracle размещает пустые значения V_CODE в верхней части списка, в то время как Access размещает их в конце списка. Местоположение пустых значений не влияет на содержимое списка.

V_CODE
21225
21231
21344
23119
24288
25595

Рис. 5.24. Распечатка отличающихся (различных) значений V_CODE таблицы PRODUCT

5.6.3. Агрегатные функции SQL

В языке SQL можно выполнять различные обобщающие (агрегатные) функции, например: подсчет числа строк, соответствующих определенным условиям, нахождение минимального и максимального значений определенных атрибутов, суммирование значений в отдельных столбцах, а также нахождение среднего значения. Агрегатные функции представлены в табл. 5.6.

Таблица 5.6. Некоторые основные агрегатные функции SQL

Функция	Результат
COUNT	Число строк, содержащих не пустые значения для данного столбца
MIN	Минимальное значение атрибута в данном столбце
MAX	Максимальное значение атрибута в данном столбце
SUM	Сумма всех значений выбранных атрибутов для данного столбца
AVG	Среднее арифметическое значений данного столбца

Вниманию пользователей Access

В предыдущих разделах мы использовали MS Access для записи стандартных SQL-команд и для представления результатов. Однако агрегатные функции, представленные в этом разделе, не могут быть реализованы в SQL-коде. Вместо этого Access разбивает выполнение большинства агрегатных функций на два этапа: во-первых, создается запрос с помощью функций QBE, затем вы используете результаты этого запроса для создания нужной агрегатной функции. Например, команда SQL

```
SELECT COUNT (DISTINCT V_CODE)
FROM PRODUCT;
```

реализуется, во-первых, созданием запроса, представленного на рис. 5.25 (панель А). Этот результат затем используется в качестве входных данных для запроса, который и генерирует окончательный результат, представленный на рис. 5.25 (панель В).

Поскольку мы хотим проиллюстрировать формат стандартных SQL-команд, большую часть остальных командных последовательностей мы будем выполнять в среде Oracle.

Панель А			Панель В	
	V_CODE	Country_CODE		Country_CODE
▶		0	▶	0
	21225	2		
	21231	1		
	21344	3		
	23119	2		
	24288	3		
	25595	3		

Рис. 5.25. Запрос в запросе: вложенные процессы

COUNT

Поскольку оператор COUNT предназначен для подсчета числа определенных значений атрибута, он используется в сочетании с оператором DISTINCT. Предположим, что вы хотите подсчитать количество различных поставщиков в таблице PRODUCT. Из ответа, полученного с помощью оператора SQL (рис. 5.26), следует, что число различных кодов поставщиков в таблице PRODUCT равно шести (пустые значения в расчет не принимаются).

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT COUNT(DISTINCT V_CODE)
2 FROM PRODUCT;

COUNT(DISTINCT V_CODE)
-----
6

SQL> SELECT COUNT(DISTINCT V_CODE)
2 FROM PRODUCT
3 WHERE P_PRICE < 10.00;

COUNT(DISTINCT V_CODE)
-----
3

SQL> SELECT COUNT(*)
2 FROM PRODUCT
3 WHERE P_PRICE <= 10.00;

COUNT(*)
-----
5

SQL> |

```

Рис. 5.26. Примеры использования функции COUNT

Агрегатные функции могут использоваться в сочетании с различными SQL-командами. Например, второй набор SQL-команд на рис. 5.26 отвечает на вопрос "Сколько поставщиков, имеющих в таблице PRODUCT, поставляют товары, цена которых ниже или равна \$10.00?". Ответ (3) указывает на то, что в таблице имеются три поставщика, удовлетворяющих данным условиям.

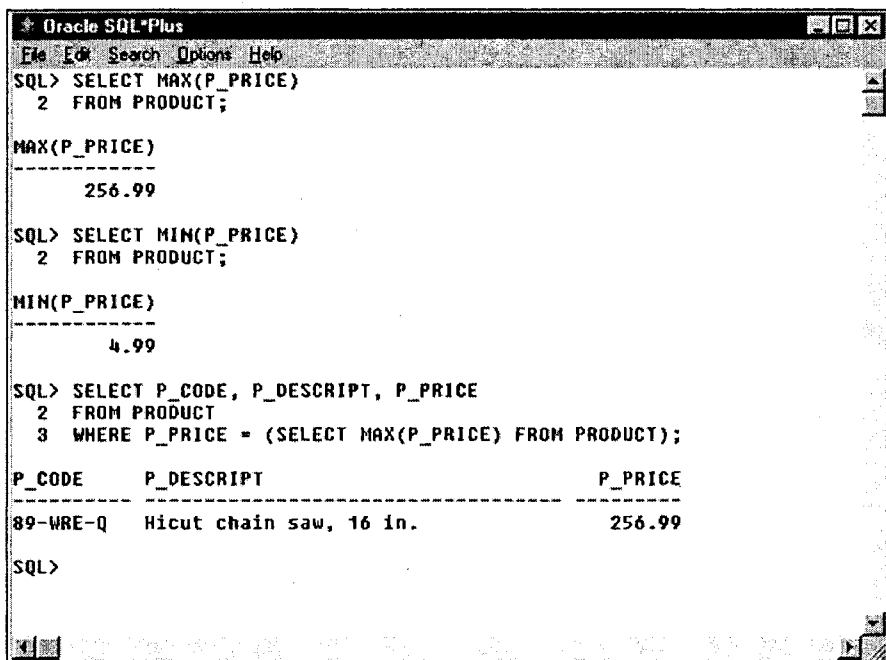
В сочетании с функцией COUNT можно использовать метасимволы. Например, в третьем наборе SQL-команд на рис. 5.26 используется метасимвол * при ответе на вопрос "Сколько строк в таблице PRODUCT имеют значение P_PRICE не больше чем \$10.00?". Ответ (5) говорит о том, что 5 наименований товара имеют цену, которая удовлетворяет заданному ограничению.

MAX и MIN

Функции MAX и MIN помогут решить такие задачи:

- ☐ нахождение максимальной цены товара в таблице PRODUCT;
- ☐ нахождение минимальной цены товара в таблице PRODUCT.

Правильный ответ на первую задачу \$256.99 обеспечивается первой SQL-командой, представленной на рис. 5.27. Вторая SQL-команда выдает минимальную стоимость товара — \$4.99.



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT MAX(P_PRICE)
2 FROM PRODUCT;

MAX(P_PRICE)
-----
256.99

SQL> SELECT MIN(P_PRICE)
2 FROM PRODUCT;

MIN(P_PRICE)
-----
4.99

SQL> SELECT P_CODE, P_DESCRIPT, P_PRICE
2 FROM PRODUCT
3 WHERE P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);

P_CODE      P_DESCRIPT                      P_PRICE
-----
89-WRE-Q    Hicut chain saw, 16 in.         256.99

SQL>
```

Рис. 5.27. Примеры выполнения функций MAX и MIN

В третьем наборе SQL-команд, представленном на рис. 5.27, показано, что числовые функции могут также использоваться совместно с более сложными логическими ограничениями. *Числовые функции выдают только одно значение* из всех значений, найденных в таблице: единственное максимальное значение, единственное минимальное значение или единственное среднее значение. Об этом нужно всегда помнить. Например, исследуем такой вопрос:

"Какой товар имеет наибольшую стоимость?"

Хотя он кажется достаточно простым, однако SQL-команда

```
SELECT P_CODE, P_DESCRIPT, P_PRICE  
FROM PRODUCT  
WHERE P_PRICE = MAX(P_PRICE);
```

не даст желаемого результата, поскольку:

- ☐ команда **SELECT** выведет список множества значений стоимости;
- ☐ сравнение со значениями **MAX(P_PRICE)** делается в командной последовательности слишком поздно: значения **P_PRICE** уже извлечены из таблицы перед тем, как функция **MAX(P_PRICE)** выполняет операцию сравнения!

Чтобы ответить на такой запрос, мы должны разрешить проблему, связанную с ограничением на единственность значения функции **MAX**, при помощи процедуры *вложенных запросов* (nested query). Как следует из названия этой процедуры, вложенный запрос — это запрос в запросе. В данном случае вложенный запрос состоит из двух частей:

- ☐ *внутренний цикл*, который выполняется в первую очередь;
- ☐ *внешний цикл*, который выполняется последним. Внешний цикл это всегда первая SQL-команда (в данном случае **SELECT**) в командной последовательности.

Используя следующую командную последовательность в качестве образца, отметим, что во внутреннем цикле, прежде всего, ищется максимальная стоимость товара, которая сохраняется в памяти. Поскольку во внешнем цикле теперь имеется значение, с которым можно сравнить **P_PRICE**, запрос будет выполняться правильно.

```
SELECT P_CODE, P_DESCRIPT, P_PRICE  
FROM PRODUCT  
WHERE P_PRICE = (SELECT MAX(P_PRICE) FROM PRODUCT);
```

Выполнение этого вложенного запроса дает правильный ответ, представленный в нижней части рис. 5.27.

SUM

Функция **SUM** вычисляет общую сумму для любого заданного атрибута на основе какого-либо условия. Например, если нужно найти общую стоимость всех товаров, хранящихся на складе, можно выполнить такую команду:

```
SELECT SUM(P_ONHAND*P_PRICE)  
FROM PRODUCT;
```

поскольку общая стоимость получается перемножением количества единиц товара на его стоимость.

AVG

Формат функции AVG похож на формат функций MAX и MIN и поэтому подпадает под те же ограничения. Первая последовательность SQL-команд, представленная на рис. 5.28, показывает, как можно получить среднее значение атрибута P_PRICE, которое в данном случае равно 56.17125. В результате выполнения второй последовательности команд на рис. 5.28 получается пять строк, описывающих товары, стоимость которых превышает среднее значение. Обратите внимание, что во втором запросе использовались вложенные запросы и выражение ORDER BY, которое мы обсуждали ранее.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT AVG(P_PRICE)
2 FROM PRODUCT;

AVG(P_PRICE)
-----
56.17125

SQL> SELECT P_DESCRIPT, P_ONHAND, P_PRICE, U_CODE
2 FROM PRODUCT
3 WHERE P_PRICE > (SELECT AVG(P_PRICE) FROM PRODUCT)
4 ORDER BY P_PRICE DESC;

P_DESCRIPT                                P_ONHAND  P_PRICE  U_CODE
-----
Hicut chain saw, 16 in.                   11      256.99   24288
Steel matting, 4'x8'x1/6", .5" mesh       18      119.95   25595
Power painter, 15 psi., 3-nozzle          8      109.99   25595
B&D jigsaw, 12-in. blade                  8      109.92   24288
B&D jigsaw, 8-in. blade                   6       99.87   24288

SQL>

```

Рис. 5.28. Примеры использования функции AVG

5.6.4. Группирование данных

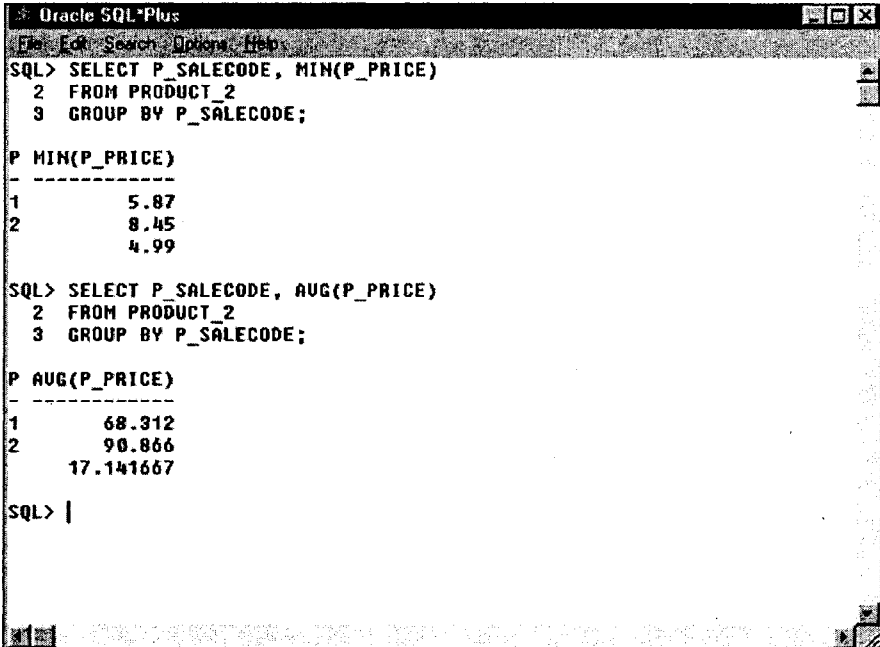
С помощью выражения GROUP BY можно быстро и без труда построить различные гистограммы. Оператор GROUP BY может включать в себя различные арифметические функции. Например, чтобы определить минимальную стоимость по каждому коду продаж, можно использовать первую командную последовательность, представленную на рис. 5.29:

```

SELECT P_SALECODE, MIN(P_PRICE)
FROM PRODUCT_2
GROUP BY P_SALECODE;

```

Вторая командная последовательность создает среднюю стоимость по каждому коду продаж. Обратите внимание, что пустые значения P_SALECODE включены в группирование.



```
Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT P_SALECODE, MIN(P_PRICE)
2 FROM PRODUCT_2
3 GROUP BY P_SALECODE;

P MIN(P_PRICE)
-----
1          5.87
2          8.45
          4.99

SQL> SELECT P_SALECODE, AVG(P_PRICE)
2 FROM PRODUCT_2
3 GROUP BY P_SALECODE;

P AVG(P_PRICE)
-----
1          68.312
2          90.866
          17.141667

SQL> |
```

Рис. 5.29. Пример использования оператора GROUP BY

Примечание

Если вы не создали резервную таблицу PRODUCT_2, как мы рекомендовали ранее в этой главе, то в данных примерах нужно заменить PRODUCT_2 на PRODUCT.

Выражение GROUP BY допустимо только тогда, когда оно используется в сочетании с одной из арифметических функций SQL, таких как COUNT, MIN, MAX, AVG и SUM. Например, если вы попытаетесь осуществить группирование таким способом:

```
SELECT V_CODE, P_CODE, P_DESCRIPT, P_PRICE
FROM PRODUCT_2
GROUP BY V_CODE;
```

то получите сообщение "not a GROUP BY expression" (некорректное выражение GROUP BY). Однако если вы запишете предыдущую командную последовательность SQL в сочетании с какой-нибудь арифметической функцией, оператор GROUP BY будет выполняться правильно. На рис. 5.30 показано сообщение об ошибке для пер-

вой группы SQL-команд. Вторая командная последовательность правильно отвечает на вопрос "Сколько товаров поставляет каждый поставщик?", поскольку здесь используется функция COUNT.

```

* Oracle SQL*Plus
File Edit Search Options Help
2 FROM PRODUCT_2
3 GROUP BY U_CODE;
SELECT U_CODE, P_CODE, P_DESCRIPT, P_PRICE
*
ERROR at line 1:
ORA-00979: not a GROUP BY expression

SQL> SELECT U_CODE, COUNT(DISTINCT(P_CODE))
2 FROM PRODUCT_2
3 GROUP BY U_CODE;

U_CODE COUNT(DISTINCT(P_CODE))
-----
21225          2
21231          1
21344          3
23119          2
24288          3
25595          3
          2

7 rows selected.

SQL>

```

Рис. 5.30. Неправильное использование оператора GROUP BY

Исследуя результаты выполнения запроса на рис. 5.30, обратите внимание, что в последней строке вывода имеется пустое значение V_CODE, говорящее о том, что два вида товара не связаны с какими-либо поставщиками. Может быть, эти товары предприятие производит самостоятельно или они закуплены по альтернативным каналам. Или же оператор, который вводил информацию, просто забыл ввести код поставщика (пустые значения могут означать что угодно, вы помните об этом?).

Модификатор HAVING оператора GROUP BY

Очень полезным расширением оператора GROUP BY является модифицирующий оператор HAVING. В основном HAVING действует так же, как выражение WHERE в операторе SELECT. Тем не менее, выражение WHERE применяется к столбцам и выражениям для отдельной строки, в то время как оператор HAVING применяется к результату действия команды GROUP BY. Например, пусть необходимо узнать количество на складе товара, поставляемого каждым поставщиком, но в данный момент времени вы желаете ограничить список товарами стоимостью менее \$10.00. Первая часть такого требования выполняется при помощи оператора GROUP BY,

как это представлено в первой командной последовательности на рис. 5.31. Во второй последовательности команд для достижения желаемого результата оператор HAVING применяется в сочетании с оператором GROUP BY.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT V_CODE, COUNT(DISTINCT(P_CODE)), AVG(P_PRICE)
2 FROM PRODUCT_2
3 GROUP BY V_CODE;

  V_CODE COUNT(DISTINCT(P_CODE))  AVG(P_PRICE)
-----
21225      2                6.47
21231      1                8.45
21344      3               12.49
23119      2               41.97
24288      3          155.59333
25595      3                89.63
              2             10.135

7 rows selected.

SQL> SELECT V_CODE, COUNT(DISTINCT(P_CODE)), AVG(P_PRICE)
2 FROM PRODUCT_2
3 GROUP BY V_CODE
4 HAVING AVG(P_PRICE) < 10;

  V_CODE COUNT(DISTINCT(P_CODE))  AVG(P_PRICE)
-----
21225      2                6.47
21231      1                8.45

SQL>

```

Рис. 5.31. Применение модификатора HAVING

Использование оператора WHERE (вместо оператора HAVING) во второй командной последовательности на рис. 5.31 приводит к сообщению об ошибке.

Чтобы показать гибкость SQL, мы можем комбинировать операторы и агрегатные функции. Например, приведенный ниже оператор SQL выполнит следующие действия:

- ☐ суммирует значения P_ONHAND, группируя их по V_CODE;
- ☐ выберет только строки, в которых общий итог превышает 5;
- ☐ расположит результаты в убывающем порядке по общему количеству на складе.

```

SELECT V_CODE, SUM(P_ONHAND) AS SumQTY
FROM PRODUCT
GROUP BY V_CODE
HAVING (SUM(P_ONHAND)>5)
ORDER BY SUM(P_ONHAND) DESC;

```


5.6.5. Виртуальные таблицы: создание представления

С помощью оператора `CREATE VIEW` можно создать виртуальную (логическую) таблицу. Такая логическая таблица существует только в памяти, хотя с ней можно работать как с реальной таблицей. И в этом случае можно создавать представления (views), не беспокоясь о том, что пользователь может по неосторожности удалить данные или добавить некорректную информацию в реальную таблицу.

Обратите внимание на синтаксис первой команды SQL (рис. 5.32), создающей логическую таблицу `PRODUCT_3`. Эта таблица (или представление) содержит только три атрибута (`P_DESCRIPT`, `P_ONHAND` и `P_PRICE`) и только для цен свыше \$50.00. Вторая команда использует таблицу, созданную в предыдущей команде.

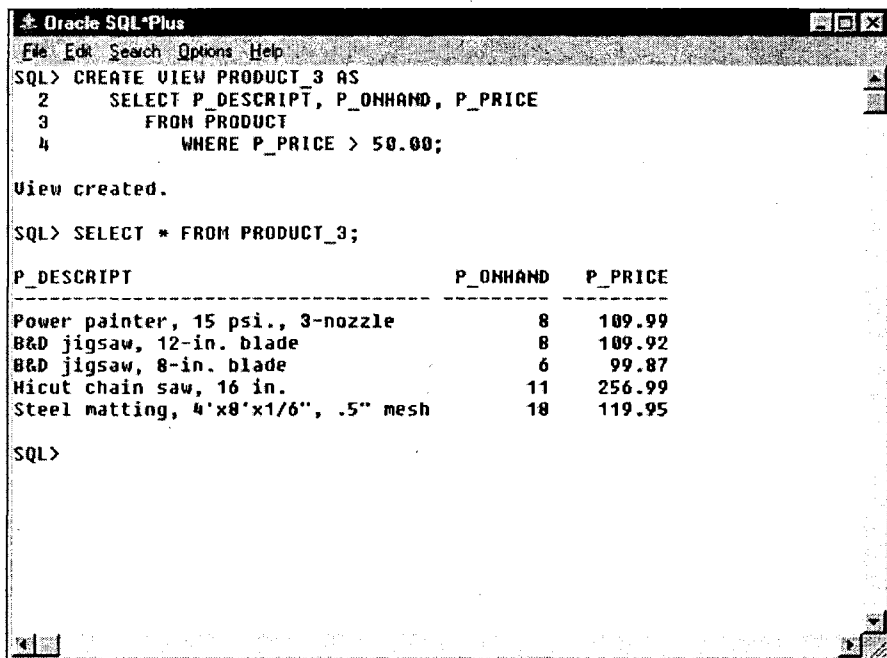


Рис. 5.32. Создание виртуальной таблицы с помощью команды `CREATE VIEW`

С помощью представлений можно обеспечить пользователям доступ ко всем данным, не беспокоясь о том, что они могут быть повреждены в реальных таблицах. А возможность предоставить пользователям доступ только к определенной части информации также обеспечивает конфиденциальность и повышает защищенность данных. Представления обновляются динамически. Поэтому если добавляются новые товары, которые отвечают условию `P_PRICE > 50.00`, они автоматически появятся в представлении `PRODUCT_3` при следующем его выводе.

Представления можно использовать в качестве основы для отчета. Например, если конечному пользователю необходимо выпустить отчет о суммарной стоимости товара, имеющегося на складе, по поставщикам, то можно создать такое представление:

```
CREATE VIEW SUMPRDXVEN AS
  SELECT V_CODE, SUM(P_ONHAND) AS TOTQTY, MAX(P_ONHAND) AS MAXQTY,
  MIN(P_ONHAND) AS MINQTY
  FROM PRODUCT
  GROUP BY V_CODE;
```

5.6.6. Индексы в SQL

Из гл. 2 вы узнали, что такое индексирование и как можно его использовать для повышения эффективности поиска данных. Индексы обычно создаются для того, чтобы обеспечить некоторые специфические условия поиска в таблицах. Однако во многих РСУБД имеются утилиты, позволяющие декларировать индексы на выбранных атрибутах уже при создании таблиц. На самом деле, при декларировании первичного ключа СУБД автоматически создает уникальный индекс. Но даже при наличии этой возможности может потребоваться дополнительное индексирование. Поэтому возможность быстрого и эффективного создания индекса в любой момент времени имеет существенное значение. SQL-индексирование по какому-нибудь атрибуту можно создать с помощью команды `CREATE INDEX`. Например, следующая командная последовательность создает индекс с названием `P_CODEX` по атрибуту `P_CODE` таблицы `PRODUCT`:

```
CREATE INDEX P_CODEX
  ON PRODUCT (P_CODE);
```

Если вы перезаписываете имеющийся индекс, то SQL обязательно выведет предупреждение, чтобы предохранить существующую структуру индексов в словаре данных от повреждения. С помощью модификатора `UNIQUE` можно даже создать индекс, препятствующий использованию повторяющихся значений. Такая возможность очень полезна в том случае, когда индексное поле (атрибут) является первичным ключом, значения которого не должны дублироваться:

```
CREATE UNIQUE INDEX P_CODEX
  ON PRODUCT (P_CODE);
```

Если вы теперь попытаетесь ввести дублирующее значение в `P_CODE`, то будет выведено сообщение об ошибке "duplicate value in index" (дублирование значения в индексе).

В общем случае нужно создавать индекс по любому полю, которое используется в качестве ключа поиска. Например, если вы хотите выполнить отчет по всем товарам, которые приобретаются у поставщиков, будет правильным создать в таблице `PRODUCT` индекс по атрибуту `V_CODE`. Помните, что данный поставщик может поставлять несколько товаров. Поэтому в данном случае нельзя создавать индекс с модификатором `UNIQUE` (уникальный). Более того, чтобы сделать поиск еще эф-

фективнее, рекомендуется создать индекс по нескольким атрибутам. Например, можно выполнить такую команду:

```
CREATE INDEX VENPRODХ
  ON PRODUCT (V_CODE, P_CODE);
```

5.6.7. Объединение таблиц базы данных

Возможность объединять (соединять) таблицы по общим атрибутам это, наверное, наиболее существенное отличие реляционных баз данных от БД неизвестного происхождения. Реляционная операция JOIN (соединение) выполняется в том случае, когда данные извлекаются более чем из одной таблицы за один раз. (Вспомните определение естественного соединения — natural JOIN — и эквисоединения — equi JOIN, приведенные в гл. 2. Помните, что язык SQL основан на реляционной алгебре.)

При объединении таблиц нужно просто перечислить таблицы в модификаторе FROM оператора SELECT. СУБД создает декартово произведение каждой таблицы в модифицирующем операторе FROM. Однако чтобы получить корректные результаты, необходимо выбрать только те строки, в которых совпадают общие атрибуты (более подробно о реляционном операторе JOIN см. гл. 2). Это можно сделать с помощью оператора WHERE. Используйте оператор WHERE для обозначения общего атрибута, используемого для связи таблиц.

Предположим, что нужно объединить две таблицы VENDOR и PRODUCT базы данных CH5_TEXT. Поскольку V_CODE является внешним ключом в таблице PRODUCT и первичным ключом в таблице VENDOR, можно установить связь (link) именно по этому ключу (табл. 5.7).

Таблица 5.7. Создание связей с помощью внешних ключей

Таблица	Необходимые атрибуты	Связующий атрибут
PRODUCT	P_DESCRIPT, P_PRICE	V_CODE
VENDOR	V_COMPANY, V_PHONE	V_CODE

Если имена некоторых атрибутов появляются в объединенной таблице более одного раза, для таких атрибутов в команде SELECT необходимо указывать таблицу-источник. Команда соединения (объединения) таблиц PRODUCT и VENDOR выглядит следующим образом:

```
SELECT PRODUCT.P_DESCRIPT, PRODUCT.P_PRICE, VENDOR.V_NAME,
  VENDOR.V_CONTACT, VENDOR.V_AREACODE, VENDOR.V_PHONE
  FROM PRODUCT, VENDOR
  WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```

Результат выполнения приведенной командной последовательности представлен на рис. 5.33.

Этот результат может выглядеть несколько иначе, поскольку SQL-команда создает распечатку, в которой порядок следования столбцов не существен. На самом деле,

скорее всего, ваша распечатка будет выглядеть иначе уже при следующем выполнении этой команды. Однако можно создать более предсказуемый список с помощью оператора упорядочивания ORDER BY:

```
SELECT P_DESCRPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
ORDER BY P_PRICE;
```

	P_DESCRPT	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
▶	Claw hammer	\$5.95	Bryson, Inc.	Smithson	615	223-3234
	1.25-in. metal screw, 25	\$6.99	Bryson, Inc.	Smithson	615	223-3234
	2.5-in. wd. screw, 50	\$8.45	O&E Supply	Singh	615	228-3245
	7.25-in. pwr. saw blade	\$14.99	Gomez Bros.	Ortega	615	889-2546
	9.00-in. pwr. saw blade	\$17.49	Gomez Bros.	Ortega	615	889-2546
	Rat-tail file, 1/8-in. fine	\$4.99	Gomez Bros.	Ortega	615	889-2546
	Hrd. cloth, 1/4-in., 2x50	\$39.95	Randssets Ltd.	Anderson	901	678-3998
	Hrd. cloth, 1/2-in., 3x50	\$43.99	Randssets Ltd.	Anderson	901	678-3998
	B&D jigsaw, 12-in. blade	\$109.92	ORDVA, Inc.	Hakford	615	898-1234
	B&D jigsaw, 8-in. blade	\$99.87	ORDVA, Inc.	Hakford	615	898-1234
	Hicut chain saw, 16 in.	\$256.99	ORDVA, Inc.	Hakford	615	898-1234
	Power painter, 15 psi, 3-nozzle	\$109.99	Rubicon Sis.	Orton	904	456-0092
	B&D cordless drill, 1/2-in.	\$38.95	Rubicon Sis.	Orton	904	456-0092
	Steel matting, 4"x8"x1/8", 5" mesh	\$119.95	Rubicon Sis.	Orton	904	456-0092

Рис. 5.33. Результат операции объединения таблиц

В этом случае распечатка всегда будет упорядочена по стоимости от наиболее низкой цены к более высокой.

Примечание

В предыдущей командной последовательности в качестве префикса использовались имена таблиц. Например, использование имени PRODUCT.P_PRICE предпочтительнее имени P_PRICE. В большинстве современных РСУБД (например, Oracle) не требуется использовать имена таблиц в качестве префиксов, за исключением тех случаев, когда одинаковые имена атрибутов встречаются в нескольких объединяемых таблицах. В данном случае V_CODE используется в качестве внешнего ключа в таблице PRODUCT и в качестве первичного ключа в таблице VENDOR, поэтому приходится использовать имена таблиц в качестве префиксов в операторе WHERE. Другими словами, предыдущий запрос мы можем записать в таком виде:

```
SELECT P_DESCRPT, P_PRICE, V_NAME,
V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE;
```

Естественно, если имя атрибута встречается в нескольких местах, в качестве префикса нужно использовать имя его начальной таблицы. Если вы не будете следовать этому правилу, SQL выведет сообщение об ошибке, где будет указано на двусмысленность в определении начальной таблицы атрибута.

Отметим, что предыдущая командная последовательность объединяет строки таблиц **PRODUCT** и **VENDOR**, в которых значения атрибута **V_CODE** одинаковы. Поскольку любой поставщик может доставить любое количество заказанных товаров, таблица **PRODUCT** может содержать несколько элементов **V_CODE** для каждого элемента **V_CODE** в таблице **VENDOR**. Другими словами, каждому значению **V_CODE** в таблице **VENDOR** может соответствовать несколько строк в таблице **PRODUCT**.

Если оператор **WHERE** не задан, то в результате товар будет ассоциирован по правилам естественного соединения (*natural JOIN*, см. гл. 2). Поскольку таблица **PRODUCT** содержит 16 строк, а таблица **VENDOR** — 11 строк, декартово произведение будет представлять собой список из $(16 \times 11) = 176$ строк, поскольку каждая строка в таблице **PRODUCT** будет соединяться с каждой строкой таблицы **VENDOR**.

К объединенным таблицам применимы все SQL-команды. Например, вполне допустима следующая командная последовательность:

```
SELECT P_DESCRPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE
AND P_INDATE > '01/15/2002';
```

Результат выполнения этой команды приведен на рис. 5.34.

	P_DESCRPT	P_PRICE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE
▶	Claw hammer	\$5.95	Bryson, Inc.	Smithson	615	223-3234
▶	B&D jig saw, 8-in. blade	\$99.87	ORDVA, Inc.	Hakford	615	898-1234
▶	Steel matting, 4'x8'x1/8", 5" mesh	\$119.95	Rubicon Sis.	Orton	904	456-0092

Рис. 5.34. Упорядочивание и ограничение после выполнения объединения

Для идентификации источника данных (таблицы) можно использовать псевдонимы. В частности, в следующей командной последовательности для обозначения таблиц **PRODUCT** и **VENDOR** используются псевдонимы **A** и **B**. В качестве псевдонима можно использовать любой корректный идентификатор таблицы. (Здесь также опущены префиксы имен таблиц, поскольку в операторе **SELECT** нет дублирующих имен атрибутов.)

```
SELECT P_DESCRPT, P_PRICE, V_NAME, V_CONTACT, V_AREACODE, V_PHONE
FROM PRODUCT A, VENDOR B
WHERE A.V_CODE = B.V_CODE
ORDER BY P_PRICE;
```

Рекурсивные запросы

Псевдонимы особенно полезны, когда таблицу нужно объединять саму с собой в рекурсивных запросах. Например, пусть мы имеем дело с таблицей **EMP**, представленной на рис. 5.35.

EMP_NUM	EMP_TITLE	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_DOW	EMP_HIRE_DATE	EMP_AREA_CODE	EMP_PHONE	EMP_MGR
100	Mr.	Kolmycz	George	D	15-Jun-42	15-Mar-85	615	324-5456	
101	Ms.	Lewis	Rhonda	G	19-Mar-85	25-Apr-86	615	324-4472	100
102	Mr.	Vandam	Rhett		14-Nov-68	20-Dec-90	901	875-8993	100
103	Ms.	Jones	Anne	M	16-Oct-74	28-Aug-94	615	898-3456	100
104	Mr.	Lange	John	P	08-Nov-71	20-Oct-94	901	504-4430	105
105	Mr.	Williams	Robert	D	14-Mar-75	06-Nov-96	615	890-3220	
106	Mrs.	Smith	Jeanine	K	12-Feb-68	05-Jan-89	615	324-7893	105
107	Mr.	Diante	Jorge	D	21-Aug-74	02-Jul-94	615	890-4567	105
108	Mr.	Wesenbach	Paul	R	14-Feb-68	18-Nov-92	615	887-4358	
109	Mr.	Smith	George	K	18-Jun-61	14-Apr-89	901	504-3339	108
110	Mrs.	Genkazi	Leighla	W	19-May-70	01-Dec-90	901	589-0093	108
111	Mr.	Washington	Rupert	E	03-Jan-86	21-Jun-83	615	890-4825	105
112	Mr.	Johnson	Edward	E	14-May-61	01-Dec-83	615	898-4387	100
113	Ms.	Smythe	Melanie	P	15-Sep-70	11-May-99	615	324-9006	105
114	Ms.	Brandon	Marie	G	02-Nov-66	15-Nov-79	901	882-0845	108
115	Mrs.	Saranda	Hermine	R	25-Jul-72	23-Apr-93	615	324-5505	105
116	Mr.	Smith	George	A	08-Nov-65	10-Dec-88	615	890-2984	108

Рис. 5.35. Содержимое таблицы EMP

Как на основе данных этой таблицы получить список всех сотрудников с фамилиями их руководителей? Эту задачу можно решить, объединив таблицу EMP саму с собой. В данном случае мы также будем использовать псевдонимы для различения таблиц. Командная последовательность SQL может выглядеть так:

```
SELECT A.EMP_NUM, A.EMP_LNAME, A.EMP_MGR, B.EMP_LNAME
FROM EMP A, EMP B
WHERE A.EMP_MGR=B.EMP_NUM
ORDER BY A.EMP_MGR;
```

Результат действия этой команды представлен на рис. 5.36.

EMP_NUM	A.EMP_LNAME	EMP_MGR	B.EMP_LNAME
102	Johnson	100	Kolmycz
103	Jones	100	Kolmycz
102	Vandam	100	Kolmycz
101	Lewis	100	Kolmycz
115	Saranda	105	Williams
113	Smythe	105	Williams
111	Washington	105	Williams
107	Diante	105	Williams
106	Smith	105	Williams
104	Lange	105	Williams
116	Smith	108	Wesenbach
114	Brandon	108	Wesenbach
110	Genkazi	108	Wesenbach
109	Smith	108	Wesenbach

Рис. 5.36. Использование псевдонимов при соединении таблицы самой с собой

Внешние соединения

На рис. 5.33 представлен результат объединения таблиц **PRODUCT** и **VENDOR**. На результирующем экране перечислены 14 строк товара. Если сравнить этот результат с таблицей **PRODUCT** на рис. 5.2, то можно заметить отсутствие двух товаров. Как это получилось? Причина в том, что в таблице имеются два товара с пустым значением атрибута **V_CODE**. Поскольку совпадения пустых "значений" атрибута **V_CODE** таблицы **VENDOR** не определены, они не представлены и в окончательном результате объединения. Если вы исследуете таблицу **VENDOR** на рис. 5.2, то увидите также, что имеется несколько поставщиков, у которых нет совпадающих значений **V_CODE** в таблице **PRODUCT**. Чтобы включить такие строки в окончательный результат, необходимо использовать внешнее соединение (**outer JOIN**).

	P_CODE	V_CODE	V_NAME
▶	23109-HE	21225	Bryson, Inc.
	SM-16277	21225	Bryson, Inc.
		21226	SuperLoo, Inc.
	SW-23116	21231	D&E Supply
	13-Q2/P2	21344	Gomez Bros.
	14-Q1/L3	21344	Gomez Bros.
	54778-2T	21344	Gomez Bros.
		22567	Dome Supply
	1546-QQ2	23119	Randsets Ltd.
	1558-QW1	23119	Randsets Ltd.
		24004	Brackman Bros.
	2232/QTY	24288	ORDVA, Inc.
	2232/QWE	24288	ORDVA, Inc.
	89-WRE-Q	24288	ORDVA, Inc.
		25443	B&K, Inc.
		25501	Damal Supplies
	11QER/31	25595	Rubicon Sis.
	2238/QPD	25595	Rubicon Sis.
	WR3/TT3	25595	Rubicon Sis.

Рис. 5.37. Левостороннее внешнее соединение

Есть два типа внешнего соединения (см. гл. 2). В следующем левостороннем внешнем соединении (**left outer JOIN**) будут представлены все строки таблицы **VENDOR** и все совпадающие строки таблицы **PRODUCT**.

```
SELECT P_CODE, VENDOR.[V_CODE], V_NAME
FROM VENDOR
LEFT JOIN PRODUCT ON VENDOR.[V-CODE] = PRODUCT.V_CODE;
```

Представленная команда будет правильно работать в MS Access, но не в Oracle. Левостороннее внешнее соединение в версии Oracle8i выглядит так:

```
SELECT P_CODE, PRODUCT.V_CODE, V_NAME
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE (+) = VENDOR.V_CODE
```

На рис. 5.37 представлен результат выполнения левостороннего внешнего соединения.

Правостороннее внешнее соединение (right outer JOIN) объединяет обе таблицы и представляет все строки товаров со всеми совпадающими строками поставщиков. SQL-команда для правостороннего соединения выглядит так:

```
SELECT PRODUCT.P_CODE, VENDOR.[V_CODE], V_NAME
FROM VENDOR
RIGHT JOIN PRODUCT ON VENDOR.[V-CODE] = PRODUCT.V_CODE;
```

И опять только что приведенная команда работает правильно в MS Access, но не в Oracle. Правостороннее соединение в Oracle8i выглядит так:

```
SELECT P_CODE, PRODUCT.V_CODE, V_NAME
FROM PRODUCT, VENDOR
WHERE PRODUCT.V_CODE = VENDOR.V_CODE (+);
```

На рис. 5.38 представлен результат действия команды правостороннего внешнего соединения.

	P_CODE	V_CODE	V_NAME
▶	23114-AA		
	PVC23DRT		
	23109-HB	21225	Bryson, Inc.
	SM-18277	21225	Bryson, Inc.
	SW-23116	21231	D&E Supply
	13-Q2/P2	21344	Gomez Bros.
	14-Q1/L3	21344	Gomez Bros.
	54778-2T	21344	Gomez Bros.
	1546-QQ2	23119	Randsets Ltd.
	1558-QW1	23119	Randsets Ltd.
	2232/QTY	24288	ORDVA, Inc.
	2232/QWE	24288	ORDVA, Inc.
	89-VVRE-Q	24288	ORDVA, Inc.
	11QER/31	25595	Rubicon Sis.
	2238/QPD	25595	Rubicon Sis.
	WR3/TT3	25595	Rubicon Sis.

Рис. 5.38. Правостороннее внешнее соединение

5.7. Обновляемые представления

Одна из самых распространенных операций в среде рабочих баз данных — пакетные процедуры обновления атрибутов основной таблицы на основе оперативных данных. Как следует из ее названия, *пакетная процедура обновления* (batch update routine) объединяет транзакции в один пакет для обновления основной таблицы за одну операцию. Например, пакетная процедура обновления обычно используется для корректировки количества имеющегося на складе товара на основе общего количества продаж. Такие процедуры обычно выполняются ночью во время минимальной загрузки системы, при этом количество товара на складе обновляется с учетом продаж в отдаленных подразделениях, мобильными торговыми агентами и т. д.

Для иллюстрации использования пакетных процедур обновления определим основную таблицу (или мастер-таблицу) товаров (PRODUCT) и таблицу объемов продаж за месяц (PMSALES) (рис. 5.39). Из рис. 5.39 следует, что между таблицами есть связь 1:1.

База данных: UV				
Таблица: PRODUCT (мастер-таблица)			Таблица: PMSALES (таблица ежемесячных продаж)	
PROD_ID	PROD_DESC	PROD_QOH	PROD_ID	PMS_QTY
A123	SCREWS	60	A123	7
BX34	NUTS	37	BX34	3
C583	BOLTS	50		

Рис. 5.39. Содержимое таблиц PRODUCT и PMSALES

С помощью таблиц, представленных на рис. 5.39, обновим таблицу PRODUCT, вычитая ежемесячный объем продаж товара (PMS_QTY) в таблице PMSALES из атрибута PROD_QOH (объем товара на складе) таблицы PRODUCT. Для этого можно воспользоваться запросом такого вида:

```
UPDATE PRODUCT, PMSALES
```

```
SET PRODUCT.PROD_QOH = PROD_QOH - PMS_QTY
```

```
WHERE PRODUCT.PROD_ID = PMSALES.PROD_ID;
```

Обратите внимание, что этот оператор обновления вызывает следующую цепочку событий:

1. Соединение таблиц PRODUCT и PMSALES;
2. Обновление атрибута PROD_QOH в каждой строке таблицы PRODUCT, где значение атрибута PROD_ID совпадает с его значением в таблице PMSALES.

В MS Access этот запрос будет работать правильно, но в Oracle появится сообщение об ошибке (рис. 5.40).

Примечание

Чтобы таблицу можно было использовать в пакетных процедурах, данные PMSALES предпочтительнее хранить в таблице, а не в представлении.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> COLUMN PROD_ID FORMAT A8
SQL> SELECT * FROM PRODUCT;

PROD_ID  PROD_DESC          PROD_QOH
-----
A123     SCREWS              67
BX34     NUTS                40
C583     BOLTS               50

SQL> SELECT * FROM PMSALES;

PROD_ID  PMS_QTY
-----
A123     7
BX34     3

SQL> UPDATE PRODUCT,PMSALES
2     SET PRODUCT.PROD_QOH = PRODUCT.PROD_QOH - PMSALES.PMS_QTY
3     WHERE PRODUCT.PROD_ID = PMSALES.PROD_ID;
UPDATE PRODUCT,PMSALES
*
ERROR at line 1:
ORA-00971: missing SET keyword

SQL>

```

Рис. 5.40. Сообщение об ошибке при выполнении операции обновления в Oracle

Почему в среде Oracle возникает ошибка? Причина в том, что Oracle ожидает только одно имя таблицы. На самом деле, Oracle запрещает соединение таблиц в операторе UPDATE. Для разрешения этой проблемы необходимо создать обновляемое представление. *Обновляемое представление* это представление, которое можно использовать для обновления атрибутов в основной таблице (или таблицах), используемой для представления. Важно отметить, что не все представления являются обновляемыми. Фактически есть несколько ограничений, влияющих на обновление представления, часть из них определяет поставщик программного обеспечения. (Мы в дальнейшем будем использовать Oracle. Для того чтобы узнать, какие ограничения накладываются на обновляемые представления в вашей РСУБД, рекомендуется внимательно изучить руководство пользователя.) К наиболее общим ограничениям можно отнести следующие:

- ☐ запрет на использование выражения GROUP BY в обновляемых представлениях;
- ☐ запрет на использование операций над множествами (UNION, INTERSECT и т. п.);
- ☐ основная часть ограничений касается использования оператора JOIN или группирующих операторов над представлениями.

Чтобы привести запрос в соответствие с требованиями Oracle, необходимо создать обновляемое представление с именем PMSVUPD, как это показано на рис. 5.41.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> RUN
1 CREATE VIEW PMSVUPD AS (
2 SELECT PRODUCT.PROD_ID, PRODUCT.PROD_QOH, PMSALES.PMS_QTY
3 FROM PRODUCT, PMSALES
4 WHERE PRODUCT.PROD_ID = PMSALES.PROD_ID)

View created.

SQL> SELECT * FROM PMSVUPD;

PROD_ID  PROD_QOH  PMS_QTY
-----
A123          67        7
BX34          40        3

SQL> |

```

Рис. 5.41. Создание обновляемого представления в Oracle

Примечание

Простой способ определения того, можно ли использовать представление для обновления основной таблицы, — проверить результаты представления. Если столбец первичного ключа основной таблицы, которую вы собираетесь обновлять, имеет в представлении уникальные значения, то основная таблица обновляема. Например, если в столбце PROD_ID представления значения 'A123' или 'BX34' встречаются больше одного раза, то таблицу PRODUCT нельзя обновить через представление.

После создания обновляемого представления (см. рис. 5.41) мы можем использовать команду UPDATE для его обновления, т. е. обновить таблицу PRODUCT. На рис. 5.42 приведен пример использования команды UPDATE и окончательное содержимое таблицы PRODUCT после выполнения операции UPDATE.

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> RUN
1 UPDATE PMSVUPD
2 SET PROD_QOH = PROD_QOH - PMS_QTY

2 rows updated.

SQL> SELECT * FROM PRODUCT;

PROD_ID  PROD_DESC  PROD_QOH
-----
A123     SCREWS      60
BX34     NUTS        37
C583     BOLTS       50

SQL> |

```

Рис. 5.42. Обновление таблицы PRODUCT с использованием обновляемого представления в Oracle

Несмотря на то что пакетная процедура обновления, которую мы только что продемонстрировали, достигла цели, и мы обновили основную таблицу данными из оперативной таблицы, в действительности предпочтительнее решать проблему обновления с помощью процедурной версии языка SQL. В следующем разделе мы рассмотрим процедурный язык SQL, который можно использовать при решении задач обновления, приведенных в конце этой главы (см. задачу 37).

5.8. Процедурный язык SQL

До сих пор мы считали, что SQL является декларирующим языком, который позволяет читать, записывать и удалять данные в БД. Например, было показано, как можно с помощью SQL обновлять данные в записях, добавлять записи и удалять их. Но представим, что нам необходимо оформить заказ поставщику на товар, если количество данного товара на складе станет ниже определенного минимума? Вы, конечно, можете выполнить оператор SQL, чтобы распечатать список всех товаров из таблицы PRODUCT, количество которых меньше определенного минимума. Затем можно использовать этот список для обновления таблицы ORDER. Наконец, используя таблицу ORDER, можно оформить заказ. Такой многоступенчатый процесс очень неэффективен, поскольку требует написать и выполнить серию SQL-операторов каждый раз, когда требуется оформить такой заказ. Более того, работа в среде SQL требует от оператора определенных знаний.

Очевидно, что оформление заказа — это важная часть сферы управления. Поэтому желательно автоматически выполнять эту процедуру, когда количество товара на складе достигло определенного минимума. К сожалению, SQL не поддерживает хранение множества процедур, основанных на некоторых логических условиях, т. е. не поддерживает выполнение процедур *по условию*, что свойственно процедурным языкам, в которых используется такой формат:

```
IF <условие>
  THEN <выполнение процедуры>
  ELSE <выполнение альтернативной процедуры>
ENDIF
```

Примечание

Словарь Вебстера определяет процедуру как действие, метод или определенный порядок выполнения. В среде баз данных процедуры обычно включают в себя одно или более действий, которые должны выполняться при определенных условиях.

В SQL также не поддерживаются операции цикла, обычные для процедурных языков. Такие операции позволяют выполнять повторяющиеся действия, которые часто встречаются в программировании. Обычный формат операций цикла выглядит так:

```
DO WHILE
  <выполнение процедуры>
END DO
```

Если необходимо выполнять операции типа IF-THEN-ELSE или DO-WHILE, т. е. операции процедурного программирования, то придется использовать один из таких языков программирования, как Cobol, Fortran, C, Pascal или Visual Basic.

Оказывается можно вставлять (встраивать) операторы SQL в программу, написанную на процедурном языке. На самом деле, использование *встроенного* SQL остается в настоящее время наиболее общим подходом обеспечения процедурных возможностей в бизнес-приложениях. Вот почему множество так называемых традиционных бизнес-приложений представляют собой гигантское количество строк программ, написанных на языке COBOL. Поскольку такой подход все еще широко распространен в традиционных приложениях, его следствием является повторение прикладного кода во многих программах. Поэтому когда требуются процедурные изменения, их приходится делать во многих местах программного кода. Такая избыточность информации зачастую создает массу проблем.

Другой подход состоит в выделении важных фрагментов кода, которые могут совместно использоваться различными прикладными программами. Преимущество такого подхода в том, что код приложения выделяется в отдельную программу, таким образом, облегчая сопровождение и контроль логики. В любом случае, появление распределенных баз данных (см. гл. 10) и объектно-ориентированных БД (см. гл. 11) требовало все большего количества кода, который было необходимо хранить и выполнять внутри базы данных. Чтобы соответствовать этим требованиям, поставщики РСУБД разработали большое число расширений языков программирования. Такие расширения включают в себя:

- структуры управления потоками заданий (IF-THEN-ELSE, DO-WHILE) для логических представлений;
- объявление переменных и обозначения внутри процедур;
- обработка ошибок.

Процедурный SQL (PL/SQL) позволяет использовать процедурный код и SQL-операторы, хранящиеся в БД. Процедурный код выполняется РСУБД при его инициировании (опосредованно или напрямую) конечным пользователем. Конечные пользователи могут использовать PL/SQL в следующих целях:

- для создания триггеров;
- для создания хранимых процедур;
- для создания функций PL/SQL.

Не надо путать функции PL/SQL со встроенными функциями SQL. В этой главе мы использовали некоторые встроенные функции SQL (например, MIN и MAX). SQL-функции могут использоваться только внутри SQL-операторов, в то время как функции PL/SQL могут вызываться только в программах PL/SQL, например, в триггерах и хранимых процедурах. Обсуждение языка PL/SQL мы начнем с изучения триггеров баз данных.

5.8.1. Триггеры

Триггером (trigger) называется процедурный SQL-код, который *автоматически* выполняется РСУБД при наступлении события манипулирования данными. Необходимо помнить, что

- ☐ триггер всегда инициируется до или после выбора, вставки или удаления строки данных;
- ☐ триггер всегда ассоциирован с таблицей БД;
- ☐ с каждой таблицей БД может быть связан один или более триггеров;
- ☐ триггер выполняется как часть транзакции, которая его инициировала.

Триггеры являются ключевыми элементами при работе с базой данных. Например:

- ☐ триггеры могут использоваться для установки ограничений, которые не были сделаны на этапе проектирования и реализации;
- ☐ триггеры расширяют функциональные возможности, позволяя автоматизировать ответственные операции и выдавая надлежащие предупреждения и советы для выполнения соответствующих корректирующих действий;
- ☐ триггеры можно использовать для обновления таблиц, вставки записей в таблицы и вызова различных хранимых процедур (см. *разд. 5.8.2*);
- ☐ триггеры расширяют возможности РСУБД и системы баз данных в целом.

То есть триггеры играют важнейшую роль при работе в среде баз данных.

Чтобы продемонстрировать создание и использование триггеров, мы рассмотрим простую задачу управления складом. Например, если количество товара на складе обновляется после факта продажи данного товара, то система должна (автоматически!) проверить, не достигло ли количество товара определенного допустимого минимума. Для демонстрации этого процесса сначала модифицируем исходную таблицу PRODUCT (см. рис. 5.2), добавив в нее минимальный объем заказа товара у поставщика (P_MIN_ORDER) и столбец признака повторного заказа (P_REORDER). Столбец P_REORDER имеет логический (булев) тип (yes/но, да/нет), и указывает на необходимость дополнительно заказать товар у поставщика. Начальное значение P_REORDER (по умолчанию) устанавливается в "но", и это будет учтено при разработке триггера. Модифицированная таблица PRODUCT представлена на рис. 5.43.

Примечание

Измененная таблица PRODUCT, представленная на рис. 5.43, хранилась в базе данных MS Access CH5_TEXT. Формат MS Access использовался по той причине, что он более доступен, в то же время другие РСУБД (например, Oracle) могут легко импортировать этот формат в свою структуру. Отметим, что MS Access использует для установки значения "yes/но" независимый переключатель (флажок), установленный флажок означает "yes", снятый — "но". При импортировании этот формат будет преобразован в формат, используемый вашей РСУБД.

Как только таблица PRODUCT будет импортирована из MS Access в вашу РСУБД, сразу же ее распечатайте, чтобы проконтролировать содержимое таблицы и форматы

полей. Поскольку для иллюстрации создания и использования триггеров и хранимых процедур мы используем Oracle, содержимое таблицы PRODUCT в Oracle должно соответствовать рис. 5.44. Обратите внимание, что формат поля P_REORDER изменился с "yes/no" на 1/0. При этом 1 означает "yes" (да), а 0 — "no" (нет).

P_CODE	P_DESCRIPTION	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE	P_MIN_ORDER	P_REORDER
11QER-1	Power painter, 15 psi, 3-nozzle	03-Dec-01	8	5	\$109.99	0.00	25595	25	<input type="checkbox"/>
13-Q2/P2	7.25-in. pwr. saw blade	13-Jan-02	32	15	\$14.99	0.05	21344	50	<input type="checkbox"/>
14-Q1/L3	9.00-in. pwr. saw blade	13-Jan-02	18	12	\$17.49	0.00	21344	50	<input type="checkbox"/>
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	15-Dec-01	15	8	\$39.95	0.00	23119	35	<input type="checkbox"/>
1558-QW1	Hrd. cloth, 1/2-in., 3x50	15-Jan-02	23	5	\$43.99	0.00	23119	25	<input type="checkbox"/>
2232/QTY	B&D jigsaw, 12-in. blade	30-Dec-01	8	5	\$109.92	0.05	24288	15	<input type="checkbox"/>
2232/QWE	B&D jigsaw, 8-in. blade	24-Jan-02	6	5	\$99.87	0.05	24288	15	<input type="checkbox"/>
2238/QPD	B&D cordless drill, 1/2-in.	20-Nov-01	12	5	\$38.95	0.05	25595	12	<input type="checkbox"/>
23109-H9	Claw hammer	20-Jan-02	23	10	\$5.95	0.10	21225	25	<input type="checkbox"/>
23114-AA	Sledge hammer, 12 lb.	02-Feb-02	8	5	\$14.40	0.05		12	<input type="checkbox"/>
S4778-2T	Rat-tail file, 1/8-in. fine	15-Jan-02	43	20	\$4.99	0.00	21344	25	<input type="checkbox"/>
89-WRE-Q	Hicut chain saw, 16 in.	07-Dec-01	11	5	\$256.99	0.05	24288	10	<input type="checkbox"/>
PVC23DRT	PVC pipe, 3.5-in., 8-ft	20-Jan-02	188	75	\$5.87	0.00		50	<input type="checkbox"/>
SM-18277	1.25-in. metal screw, 25	29-Dec-01	172	75	\$8.99	0.00	21225	50	<input type="checkbox"/>
SW-23116	2.5-in. wd. screw, 50	24-Dec-01	237	100	\$8.45	0.00	21231	100	<input type="checkbox"/>
WR3/TT3	Steel matting, 4'x8'x1/8", 5" mesh	17-Jan-02	18	5	\$119.95	0.10	25595	10	<input type="checkbox"/>

Рис. 5.43. Измененная таблица PRODUCT

* Oracle SQL*Plus									
File Edit Search Options Help									
SQL> GC:\CH5\PRODLIST.SQL									
P_CODE	P_DESCRIPTION	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	V_CODE	P_MIN_ORDER	P_REORDER
11QER/21	Power painter,	03-DEC-01	8	5	109.99	0.00	25595	25	0
13-Q2/P2	7.25-in. pwr.	13-JAN-02	32	15	14.99	0.05	21344	50	0
14-Q1/L3	9.00-in. pwr.	13-JAN-02	18	12	17.49	0.00	21344	50	0
1546-QQ2	Hrd. cloth, 1/	15-DEC-01	15	8	39.95	0.00	23119	35	0
1558-QW1	Hrd. cloth, 1/	15-JAN-02	23	5	43.99	0.00	23119	25	0
2232/QTY	B&D jigsaw, 12	30-DEC-01	8	5	109.92	0.05	24288	15	0
2232/QWE	B&D jigsaw, 8-	24-JAN-02	6	5	99.87	0.05	24288	15	0
2238/QPD	B&D cordless d	20-NOV-01	12	5	38.95	0.05	25595	12	0
23109-H9	Claw hammer	20-JAN-02	23	10	5.95	0.10	21225	25	0
23114-AA	Sledge hammer,	02-FEB-02	8	5	14.40	0.05		12	0
S4778-2T	Rat-tail file,	15-JAN-02	43	20	4.99	0.00	21344	25	0
89-WRE-Q	Hicut chain sa	07-DEC-01	11	5	256.99	0.05	24288	10	0
PVC23DRT	PVC pipe, 3.5-	20-JAN-02	188	75	5.87	0.00		50	0
SM-18277	1.25-in. metal	29-DEC-01	172	75	6.99	0.00	21225	50	0
SW-23116	2.5-in. wd. sc	24-DEC-01	237	100	8.45	0.00	21231	100	0
WR3/TT3	Steel matting,	17-JAN-02	18	5	119.95	0.10	25595	10	0
16 rows selected.									
SQL>									

Рис. 5.44. Распечатка таблицы PRODUCT в среде РСУБД Oracle

Используя распечатку, представленную на рис. 5.44, мы разработаем триггер для расчета количества товара на складе (P_ONHAND). Если эта величина становится меньше предварительно заданного минимума (P_MIN), то значение в соответствующем столбце должно принять значение "yes" (т. е. необходимо заказать товар). (Напомним, что число "1" в столбце P_REORDER означает "yes" — да.)

Синтаксис триггера в Oracle выглядит так:

```
CREATE OR REPLACE TRIGGER <название триггера>
  [BEFORE / AFTER] [DELETE / INSERT / UPDATE OF <название столбца>]
  ON <название таблицы>
  [FOR EACH ROW]
BEGIN
  Операторы PL/SQL;
...
END;
```

Примечание

В код триггера нельзя включать операторы SQL, управляющие ходом транзакции, такие как COMMIT или ROLLBACK. Oracle рекомендует использовать триггеры в следующих случаях:

- в целях контроля (создание журнала проверки);
- автоматическое создание значений производных атрибутов;
- выполнение ограничений безопасности и бизнеса;
- создание реплик таблиц для создания резервных копий.

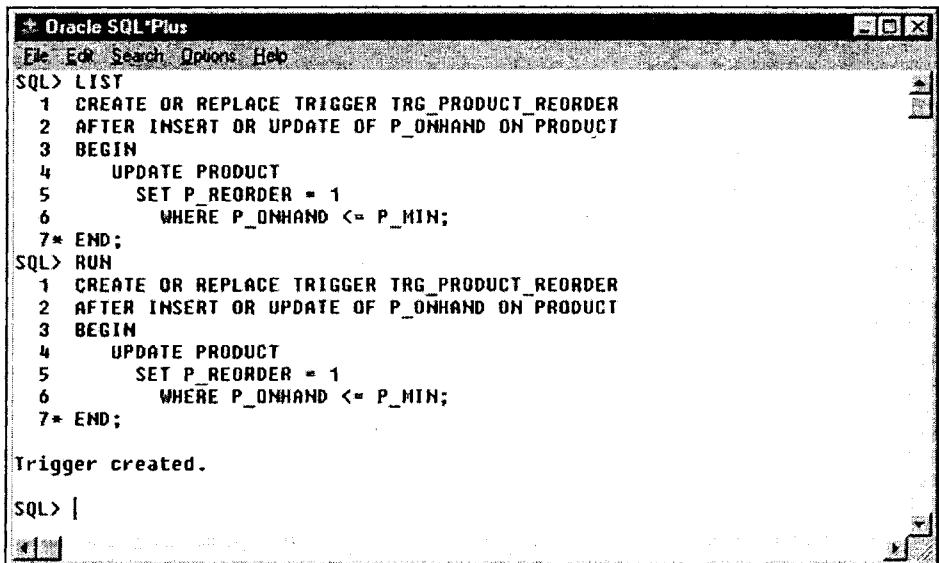


Рис. 5.45. Создание триггера для таблицы PRODUCT в PCYБД Oracle

Мы создадим триггер, неявно выполняемый после обновления столбца P_ONHAND. Действие триггера будет состоять в сравнении значения P_ONHAND с

значением атрибута P_MIN. Если значение P_ONHAND станет равно или меньше, чем P_MIN, триггер установит атрибут P_REORDER в значение 1. При создании триггера будем использовать язык SQL*Plus СУБД Oracle. Код триггера представлен на рис. 5.45.

Для тестирования триггера обновим количество товара "11QER/31" на складе до значения 4. После обновления триггер автоматически выполнится и присвоит атрибуту P_REORDER значение 1 для всех товаров, количество которых на складе меньше заданного минимума.

Примечание

С точки зрения документирования триггеры лучше именовать так, чтобы в их именах нашли отражение таблицы, на которые воздействует триггер. Кроме того, в именах триггеров используется префикс TRG. Поэтому правильное имя триггера должно иметь формат TRG_TABLE_NAME. Например, имя триггера TRG_PRODUCT_REORDER указывает, что таблица PRODUCT является источником для триггера, названного REORDER.

На первый взгляд триггер, представленный на рис. 5.46, работает верно. Но что произойдет, если мы уменьшим минимально допустимое значение количества товара "2232/QWE"? На рис. 5.47 показано, что после того, как количество товара станет меньше его допустимого минимального значения, признак необходимости дополнительного заказа все равно остается равным 0. Почему?

```

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT * FROM PRODUCT WHERE P_CODE = '11QER/31';

P_CODE  P_DESCRIPT  P_INDATE  P_ONHAND  P_MIN  P_PRICE  P_DISCOUNT  U_CODE  P_MIN_ORDER  P_REORDER
-----
11QER/31 Power painter, 03-DEC-01      8      5  109.99      0.00  25595          25          0

SQL> UPDATE PRODUCT
2      SET P_ONHAND = 4
3      WHERE P_CODE = '11QER/31';

1 row updated.

SQL> SELECT * FROM PRODUCT WHERE P_CODE = '11QER/31';

P_CODE  P_DESCRIPT  P_INDATE  P_ONHAND  P_MIN  P_PRICE  P_DISCOUNT  U_CODE  P_MIN_ORDER  P_REORDER
-----
11QER/31 Power painter, 03-DEC-01      4      5  109.99      0.00  25595          25          1

SQL>
  
```

Рис. 5.46. Поле P_REORDER таблицы PRODUCT, обновленное в результате действия триггера

Ответ прост: мы обновляли столбец P_MIN, а наш триггер работает только после обновления столбца P_ONHAND. Чтобы избежать этого несоответствия, мы должны изменить триггер так, чтобы включить в него и поле P_MIN. Обновленный код триггера представлен на рис. 5.48.

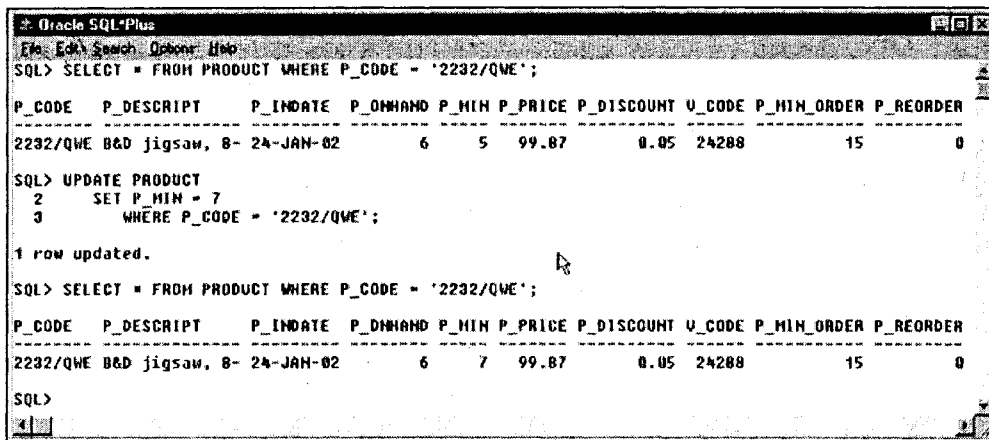


Рис. 5.47. Неверное значение P_REORDER

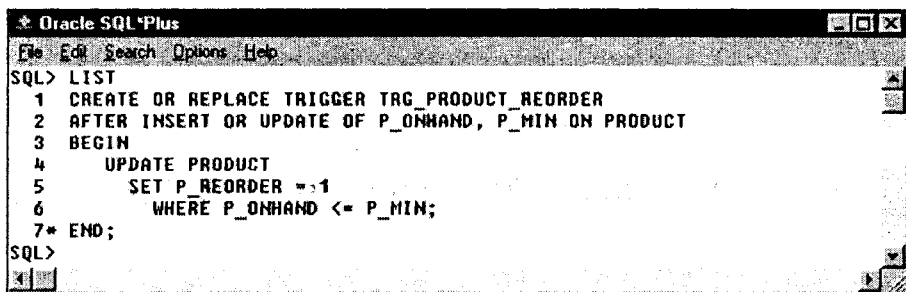


Рис. 5.48. Вторая версия триггера PRODUCT_REORDER

Для тестирования нового триггера изменим минимальное значение товара "23114-AA" на 8. После этого обновления триггер сработает правильно, и признак необходимости заказа будет установлен должным образом для всех товаров в таблице PRODUCT (рис. 5.49).

Кажется, что вторая версия триггера работает правильно. Но что произойдет, если мы изменим значение атрибута P_ONHAND товара "11QER/31", как показано на рис. 5.50? Ничего! Признак необходимости заказа так и остался равным 1. Почему триггер не изменил значение этого признака на 0?

Это происходит вследствие того, что в нашем триггере не учтены все возможные варианты. Давайте обсудим недостатки этого триггера.

- ☐ Триггер выполняет обновление для всех строк таблицы PRODUCT, даже если эту операцию необходимо выполнить только для одной строки. Это может повлиять на эффективность базы данных.
- ☐ Триггер устанавливает значение P_REORDER только в 1 и никогда не устанавливает его в 0, даже если это необходимо.

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> UPDATE PRODUCT
2 SET P_MIN = 10
3 WHERE P_CODE = '23114-AA';

1 row updated.

SQL> SELECT * FROM PRODUCT;

P_CODE P_DESCRPT P_INDATE P_ONHAND P_MIN P_PRICE P_DISCOUNT U_CODE P_MIN_ORDER P_REORDER
-----
11QER/31 Power painter, 03-DEC-01 4 5 109.99 0.00 25595 25 1
13-Q2/P2 7.25-in. pwr. 13-JAN-02 32 15 14.99 0.05 21344 50 0
14-Q1/L3 9.00-in. pwr. 13-JAN-02 18 12 17.49 0.00 21344 50 0
1546-QQ2 Hrd. cloth, 1/ 15-DEC-01 15 8 39.95 0.00 23119 35 0
1558-QW1 Hrd. cloth, 1/ 15-JAN-02 23 5 43.99 0.00 23119 25 0
2232/QTY B&D jigsaw, 12 30-DEC-01 8 5 109.92 0.05 24288 15 0
2232/QWE B&D jigsaw, 8- 24-JAN-02 6 7 99.87 0.05 24288 15 1
2238/QPD B&D cordless d 20-NOV-01 12 5 38.95 0.05 25595 12 0
23109-HB Claw hammer 20-JAN-02 23 10 5.95 0.10 21225 25 0
23114-AA Sledge hammer, 02-FEB-02 8 10 14.40 0.05 21344 25 0
54778-2T Rat-tail file, 15-JAN-02 43 20 4.99 0.00 21344 25 0
09-WRE-Q Hicut chain sa 07-DEC-01 11 5 256.99 0.05 24288 10 0
PUC23DRT PUC pipe, 3.5- 20-JAN-02 108 75 5.87 0.00 21225 50 0
SM-10277 1.25-in. metal 29-DEC-01 172 75 6.99 0.00 21225 50 0
SW-23116 2.5-in. wd. sc 24-DEC-01 237 100 0.45 0.00 21231 100 0
WR3/TT3 Steel matting, 17-JAN-02 18 5 119.95 0.10 25595 10 0

16 rows selected.

SQL>

```

Рис. 5.49. Успешное выполнение триггера после обновления значения P_MIN

```

+ Oracle SQL*Plus
File Edit Search Options Help
SQL> UPDATE PRODUCT
2 SET P_ONHAND = P_ONHAND + P_MIN_ORDER
3 WHERE P_CODE = '11QER/31';

1 row updated.

SQL> SELECT * FROM PRODUCT;

P_CODE P_DESCRPT P_INDATE P_ONHAND P_MIN P_PRICE P_DISCOUNT U_CODE P_MIN_ORDER P_REORDER
-----
11QER/31 Power painter, 03-DEC-01 29 5 109.99 0.00 25595 25 1
13-Q2/P2 7.25-in. pwr. 13-JAN-02 32 15 14.99 0.05 21344 50 0
14-Q1/L3 9.00-in. pwr. 13-JAN-02 18 12 17.49 0.00 21344 50 0
1546-QQ2 Hrd. cloth, 1/ 15-DEC-01 15 8 39.95 0.00 23119 35 0
1558-QW1 Hrd. cloth, 1/ 15-JAN-02 23 5 43.99 0.00 23119 25 0
2232/QTY B&D jigsaw, 12 30-DEC-01 8 5 109.92 0.05 24288 15 0
2232/QWE B&D jigsaw, 8- 24-JAN-02 6 7 99.87 0.05 24288 15 1
2238/QPD B&D cordless d 20-NOV-01 12 5 38.95 0.05 25595 12 0
23109-HB Claw hammer 20-JAN-02 23 10 5.95 0.10 21225 25 0
23114-AA Sledge hammer, 02-FEB-02 8 10 14.40 0.05 21344 25 0
54778-2T Rat-tail file, 15-JAN-02 43 20 4.99 0.00 21344 25 0
09-WRE-Q Hicut chain sa 07-DEC-01 11 5 256.99 0.05 24288 10 0
PUC23DRT PUC pipe, 3.5- 20-JAN-02 108 75 5.87 0.00 21225 50 0
SM-10277 1.25-in. metal 29-DEC-01 172 75 6.99 0.00 21225 50 0
SW-23116 2.5-in. wd. sc 24-DEC-01 237 100 0.45 0.00 21231 100 0
WR3/TT3 Steel matting, 17-JAN-02 18 5 119.95 0.10 25595 10 0

16 rows selected.

SQL>

```

Рис. 5.50. Признак P_REORDER принимает неправильное значение после увеличения P_ONHAND

Короче говоря, вторая версия триггера все еще работает неправильно. Посмотрим, как мы можем изменить триггер, чтобы учесть все варианты обновления. (Такой модифицированный триггер представлен на рис. 5.51.)

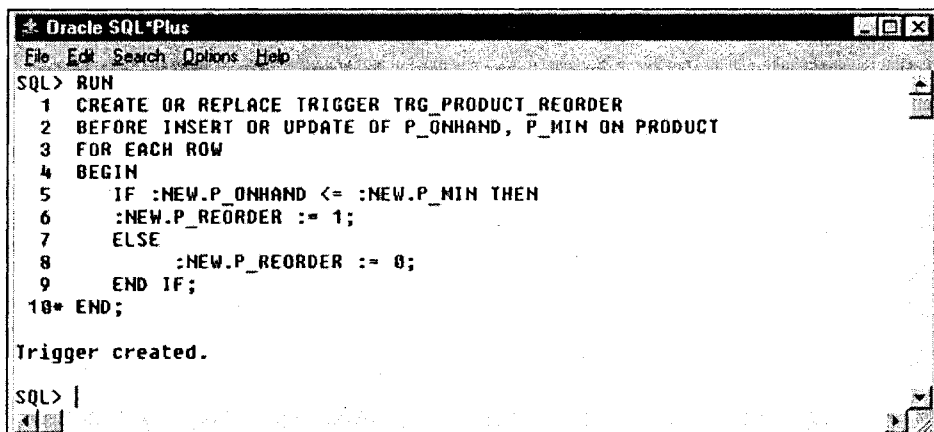


Рис. 5.51. Третья версия триггера PRODUCT_REORDER

Триггер, представленный на рис. 5.51, обладает некоторыми новыми возможностями:

- ☐ этот триггер начинает работать до того, как выполняется оператор, после которого он должен фактически сработать. На рис. 5.51 запуск триггера определен в строке 2: BEFORE INSERT OR UPDATE. Здесь явно указано, что его инициирует вставка (INSERT) или обновление (UPDATE);
- ☐ оператор FOR EACH ROW указывает, что триггер выполняется один раз для каждой строки, к которой применялся оператор, запускающий триггер, а не для всех строк таблицы;
- ☐ если триггер запускается перед (BEFORE) выполнением оператора, он имеет доступ к значениям столбцов данной таблицы как *до* (OLD), так и *после* (NEW) выполнения команды. Например, когда мы указываем:

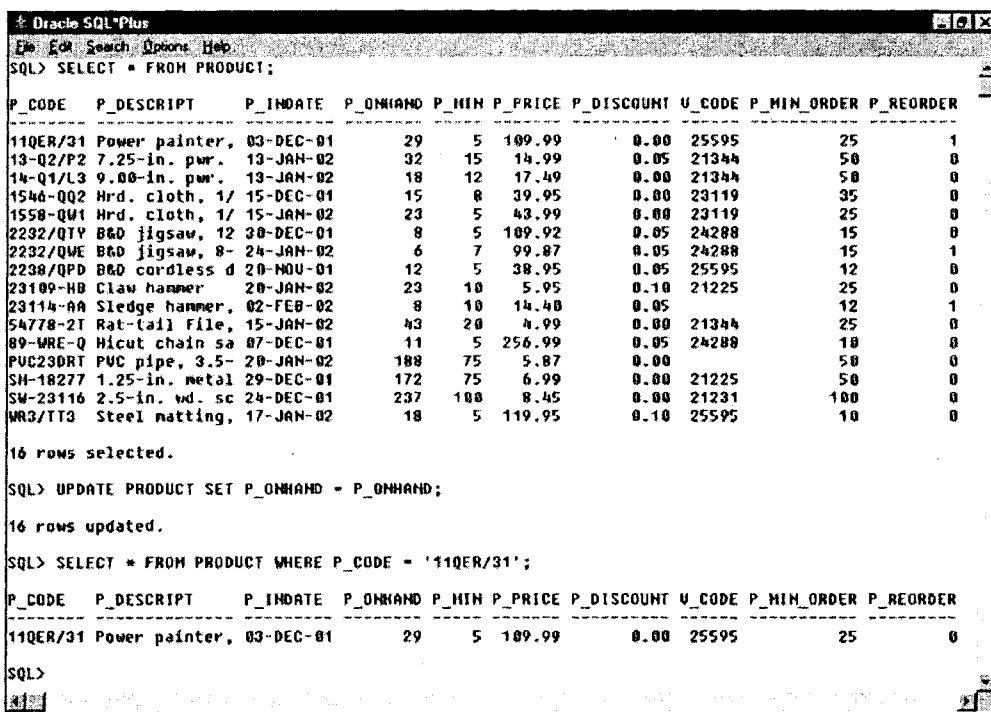
```
IF:NEW.P_ONHAND <= :NEW.P_MIN
```

то сравниваются новые значения, которые будут помещены в эти два столбца. Можно также использовать выражение

```
:OLD.< имя столбца >
```

для ссылки на предыдущее значение в этом столбце.

Перед тем как протестировать новый триггер, убедитесь, что текущее количество товара "11QER/31" выше допустимого минимального значения, а значение признака необходимости заказа равно 1, в то время как при таких условиях оно должно быть равно 0. После создания нового триггера выполним оператор UPDATE (обновление) для того, чтобы инициировать триггер, как показано на рис. 5.52.



```

SQL> SELECT * FROM PRODUCT;

P_CODE  P_DESCRIPTOR  P_INDATE  P_ONHAND  P_MIN  P_PRICE  P_DISCOUNT  U_CODE  P_MIN_ORDER  P_REORDER
-----
11QER/31 Power painter, 03-DEC-01      29      5  109.99      0.00  25595      25      1
13-Q2/P2 7.25-in. pwr. 13-JAN-02      32     15   14.99      0.05  21344      50      0
14-Q1/L3 9.00-in. pwr. 13-JAN-02      18     12   17.49      0.00  21344      50      0
1544-Q02 Hrd. cloth, 1/ 15-DEC-01      15      8   39.95      0.00  23119      35      0
1558-Q01 Hrd. cloth, 1/ 15-JAN-02      23      5   43.99      0.00  23119      25      0
2232/QTY B&D jigsaw, 12 30-DEC-01       8      5  109.92      0.05  24288      15      0
2232/QWE B&D jigsaw, 8- 24-JAN-02       6      7   99.87      0.05  24288      15      1
2238/QPD B&D cordless d 20-MAR-01      12      5   38.95      0.05  25595      12      0
23109-HB Claw hammer 20-JAN-02      23     10    5.95      0.10  21225      25      0
23114-AA Sledge hammer, 02-FEB-02       8     10   14.40      0.05      12      1
54778-2T Rat-tail file, 15-JAN-02      43     20    4.99      0.00  21344      25      0
89-WRE-Q Hicut chain sa 07-DEC-01      11      5  256.99      0.05  24288      10      0
PUC23DRT PVC pipe, 3.5- 20-JAN-02     188     75    5.87      0.00      50      0
SH-18277 1.25-in. metal 29-DEC-01     172     75    6.99      0.00  21225      50      0
SW-23116 2.5-in. wd. sc 24-DEC-01     237    100    8.45      0.00  21231     100      0
WR3/TT3 Steel matting, 17-JAN-02      18      5  119.95      0.10  25595      10      0

16 rows selected.

SQL> UPDATE PRODUCT SET P_ONHAND = P_ONHAND;

16 rows updated.

SQL> SELECT * FROM PRODUCT WHERE P_CODE = '11QER/31';

P_CODE  P_DESCRIPTOR  P_INDATE  P_ONHAND  P_MIN  P_PRICE  P_DISCOUNT  U_CODE  P_MIN_ORDER  P_REORDER
-----
11QER/31 Power painter, 03-DEC-01      29      5  109.99      0.00  25595      25      0

SQL>

```

Рис. 5.52. Результат работы третьей версии триггера

На рис. 5.52 обратите внимание, что хотя оператор UPDATE и не менял никаких значений, триггер все же корректно обновил значение P_REORDER для каждой строки, на которую воздействовал "фиктивный" оператор UPDATE.

5.8.2. Хранимые процедуры

Хранимая процедура (stored procedure) представляет собой именованный набор процедурных операторов и SQL-операторов. Этот именованный набор хранится в базе данных. Хранимые процедуры вызываются по имени.

Хранимые процедуры выполняются как единый программный компонент. Поэтому когда требуется определить транзакцию с множеством обновлений, можно создать хранимую процедуру, которая будет храниться в БД. На выполнение будет передаваться сразу все содержимое хранимой процедуры, тем самым мы избегаем поэтапного выполнения отдельных операторов SQL по сети. Следовательно, использование хранимых процедур значительно уменьшает сетевой трафик и повышает производительность БД.

При создании хранимой процедуры используется следующий синтаксис:

```
CREATE OR REPLACE PROCEDURE имя_процедуры (аргумент IN/OUT тип_данных,
и т. д.)
```

```
IS/AS BEGIN
  DECLARE имя и тип переменной
  PL/SQL- или SQL-операторы;
END;
```

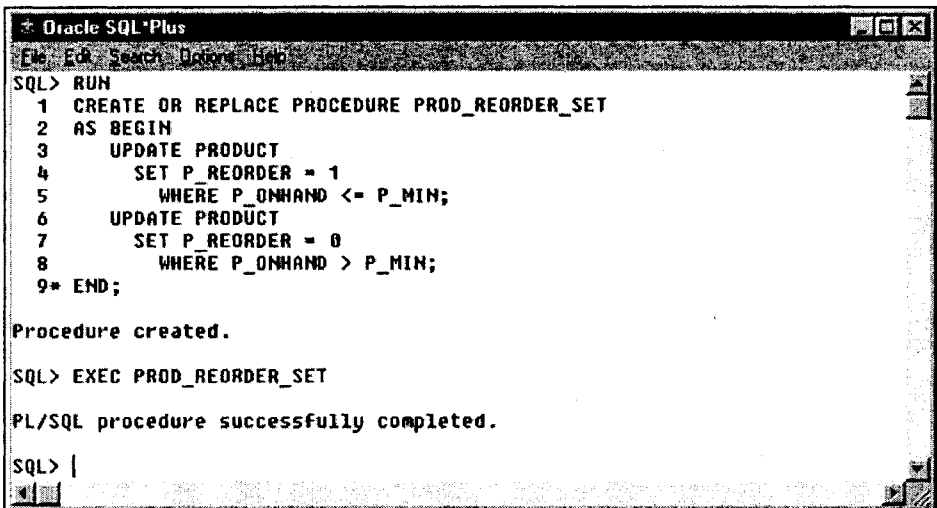
Чтобы помочь вам разобраться в синтаксисе хранимой процедуры, отметим, что:

- ☐ оператор **DECLARE** используется для определения переменных, применяемых внутри процедуры. Нужно определить как имя переменной, так и ее тип;
- ☐ "аргумент" определяет параметры, которые должны быть переданы в хранимую процедуру;
- ☐ **IN/OUT** указывает, предназначен ли данный параметр для ввода (**INPUT**) или для вывода (**OUTPUT**), или для того и другого;
- ☐ "тип данных" представляет собой один из типов данных процедурного языка SQL, используемый в РСУБД. "Тип данных" обычно соответствует типам данных в операторе создания таблицы.

Мы создадим хранимую процедуру для установки значений столбца **P_REORDER** в соответствии с правилами, применявшимися в наших предыдущих примерах. Для вызова хранимой процедуры используется следующий синтаксис:

```
EXEC имя_хранимой_процедуры (параметр1, параметр2, ...)
```

Это синтаксис языка SQL*Plus, но можно использовать и синтаксис процедурного языка, из которого вызывается хранимая процедура. Хранимую процедуру можно вызывать и из триггера. На рис. 5.53 показано, как создавать и выполнять хранимую процедуру.



```
Oracle SQL*Plus
File Edit Search Database Help
SQL> RUN
1 CREATE OR REPLACE PROCEDURE PROD_REORDER_SET
2 AS BEGIN
3   UPDATE PRODUCT
4     SET P_REORDER = 1
5     WHERE P_ONHAND <= P_MIN;
6   UPDATE PRODUCT
7     SET P_REORDER = 0
8     WHERE P_ONHAND > P_MIN;
9* END;

Procedure created.

SQL> EXEC PROD_REORDER_SET

PL/SQL procedure successfully completed.

SQL> |
```

Рис. 5.53. Создание и выполнение простейшей хранимой процедуры

Из рис. 5.53 следует, что хранимая процедура PROD_REORDER_SET запускает два запроса обновления (UPDATE). Первый присваивает значение 1 атрибуту P_REORDER каждого товара, количество которого на складе меньше установленного минимума, второй присваивает значение 0 атрибуту P_REORDER каждого товара, количество которого на складе выше установленного минимума.

Затем мы создадим простую хранимую процедуру, которая регистрирует факт продажи товара. Эта процедура будет обновлять значение атрибута, соответствующего количеству данного товара на складе. Процедура использует два параметра: код товара и количество проданного товара. Код этой процедуры хранится в файле prc_prod_sale.sql, содержимое которого представлено на рис. 5.54.

```

Oracle SQL*Plus
SQL> RUN
1 CREATE OR REPLACE PROCEDURE PROD_SALE (CODE IN VARCHAR2, QTY SOLD IN NUMBER)
2 AS BEGIN
3 UPDATE PRODUCT
4 SET P_ONHAND = P_ONHAND - QTY SOLD
5 WHERE P_CODE = CODE;
6* END;

Procedure created.

SQL> SELECT * FROM PRODUCT WHERE P_CODE = '2232/PTY';

P_CODE P_DESCRIPTOR P_INDATE P_ONHAND P_MIN P_PRICE P_DISCOUNT U_CODE P_MIN_ORDER P_REORDER
-----
2232/PTY B&D jigsaw, 12 30-DEC-01      8    5  109.92      0.05  24288      15         0

SQL> EXEC PROD_SALE('2232/PTY',3)

PL/SQL procedure successfully completed.

SQL> SELECT * FROM PRODUCT WHERE P_CODE = '2232/PTY';

P_CODE P_DESCRIPTOR P_INDATE P_ONHAND P_MIN P_PRICE P_DISCOUNT U_CODE P_MIN_ORDER P_REORDER
-----
2232/PTY B&D jigsaw, 12 30-DEC-01      5    5  109.92      0.05  24288      15         1
  
```

Рис. 5.54. Создание и выполнение хранимой процедуры PROD_SALE

Для тестирования хранимой процедуры мы положим, что было продано 3 единицы товара "2232/PTY". Из рис. 5.54 следует, что в настоящий момент времени имеются 8 единиц этого товара (см. сразу после текста хранимой процедуры). Также отметим, что это количество больше заданного минимума, поэтому признак необходимости заказа установлен в 0.

При выполнении хранимой процедуры последний оператор SELECT * показывает, что хранимая процедура PROD_SALE надлежащим образом уменьшает значение атрибута количества товара "2232/PTY" на складе. Также отметим, что признак P_REORDER автоматически устанавливается в 1 триггером TRG_PRODUCT_REORDER, созданным в предыдущем разделе.

Одно из главных достоинств хранимых процедур в том, что их можно использовать для подготовки и выполнения транзакции в целом. Например, мы можем создать

хранимую процедуру для отслеживания продажи товара, обновления кредита или регистрации группы. Таким образом, мы можем инкапсулировать несколько SQL-операторов внутри одной хранимой процедуры и выполнять их впоследствии как единую транзакцию.

Использование хранимых процедур повышает производительность системы, поскольку все транзакции выполняются локально в РСУБД, и отдельный SQL-оператор не "бегает" каждый раз по линиям связи. Еще одно преимущество хранимых процедур в том, что мы можем уменьшить дублирование кода посредством его изоляции (создания уникальных модулей), что снижает вероятность возникновения ошибок.

Можно считать, что триггеры нужно использовать в том случае, когда при вставке новых столбцов в таблицу необходимо выполнить какие-то дополнительные действия. А хранимые процедуры обычно используются для выполнения бизнес-операций, требующих обновления многих таблиц, или для обработки сложных бизнес-правил и ограничений.

5.8.3. Хранимые функции PL/SQL

С помощью программного или процедурного языка SQL можно также создавать собственные хранимые функции. Хранимые процедуры и функции очень похожи. *Хранимая функция* представляет собой группу операторов процедурного языка и операторов SQL, которые возвращают какое-то значение (определенное в операторе RETURN внутри программного кода). При создании функции используется следующий синтаксис:

```
CREATE FUNCTION имя_функции (аргумент IN тип_данных и т. д.)  
RETURN тип_данных  
AS BEGIN  
    Операторы PL/SQL;  
    RETURN (значение);  
END;
```

Хранимые функции можно вызывать только из хранимых процедур или триггеров, но не из SQL-операторов, где используются специальные встроенные SQL-функции (MIN, MAX, AVG). (Не путайте встроенные SQL-функции и хранимые функции.)

5.9. Конвертирование ER-модели в структуру базы данных

Конвертирование любой ER-модели в набор таблиц базы данных требует выполнения особых правил, которым подчиняется такое преобразование. Использование этих правил требует в свою очередь понимания воздействия, которое оказывает на БД обновление и удаление таблиц. Перед тем как обсудить эти правила подробно, кратко рассмотрим простую модель, ее схему и SQL-команды, используемые при создании таблиц.

Эта модель представляет собой базу данных ARTIST (художники), в которой нашли отражение следующие условия.

- Художник может написать много картин. В контексте базы данных ARTIST художник должен написать, по крайней мере, одну картину. Бизнес-правило устанавливает, что в связи между PAINTER (художник) и PAINTING (картина) мощность равна (1,N).
- Каждая картина написана одним (и только одним) художником.
- Картина может быть выставлена (или нет) в галерее, т. е. GALLERY (галерея) — необязательная сущность для сущности PAINTING.

Используя это описание, создадим простую ER-модель и несколько соответствующих таблиц базы данных ARTIST (рис. 5.55).

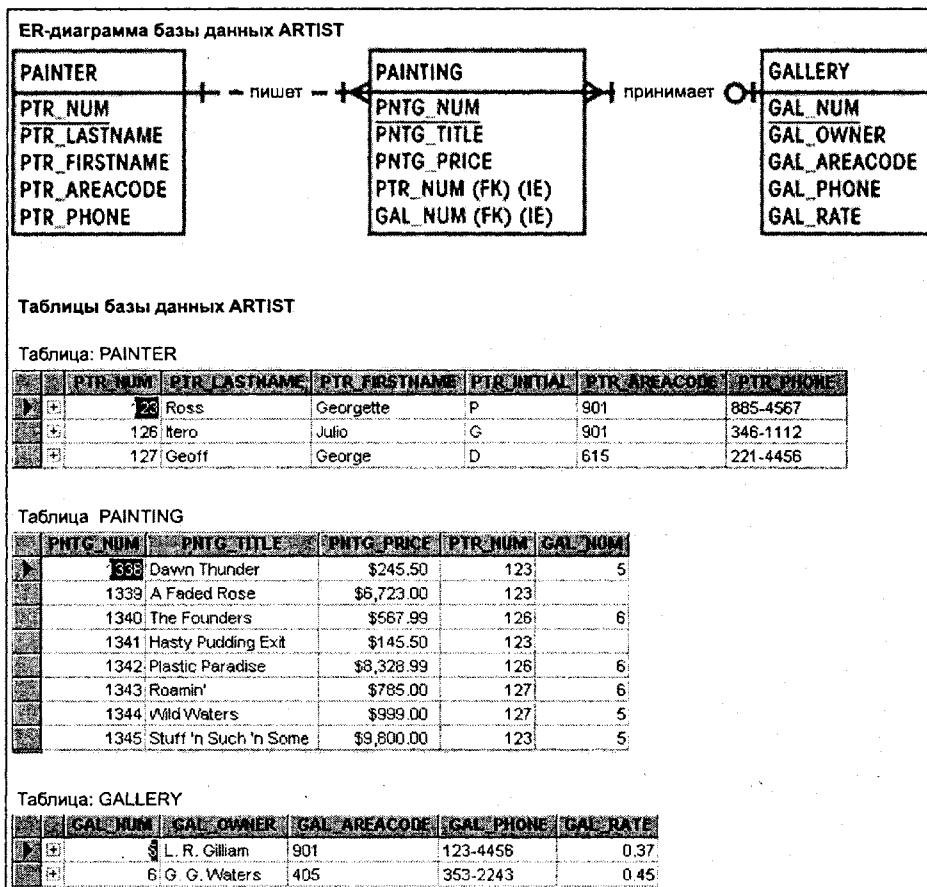


Рис. 5.55. ER-диаграмма и таблицы базы данных ARTIST

Таблицы и атрибуты, соответствующие этой ER-модели, представлены в сводной табл. 5.8.

Таблица 5.8. Словарь данных для базы данных ARTIST

Таблица	Атрибут	Содержимое	Тип	Формат	Диапазон	Обязательность	РК или FK	Таблица, на которую ссылается FK
PAINTER	PTR_NUM	Номер художника	CHAR(4)	9999	1000—9999	Да	РК	
	PTR_LASTNAME	Фамилия художника	VCHAR(15)	Xxxxxx...		Да		
	PTR_FIRSTNAME	Имя художника	VCHAR(15)	Xxxxxx...		Да		
	PTR_INITIAL	Отчество (инициал) художника	CHAR(1)	X				
	PTR_AREACODE	Код города художника	CHAR(3)	999				
GALLERY	PTR_PHONE	Телефон художника	CHAR(8)	999—9999				
	GAL_NUM	Номер галереи	CHAR(4)	9999	1000—9999	Да	РК	
	GAL_OWNER	Хозяин галереи	VCHAR(35)	Xxxxxx...				
	GAL_AREACODE	Код города галереи	CHAR(3)	999		Да		
PAINTING	GAL_PHONE	Телефон галереи	CHAR(8)	999—9999		Да		
	GAL_RATE	Комиссионные (процент)	NUMBER(4,2)	99.99	0.00—60.00	Да		
	PNTG_NUM	Номер картины	CHAR(4)	9999	1000—9999	Да	РК	

Таблица 5.8 (окончание)

Таблица	Атрибут	Содержимое	Тип	Формат	Диапазон	Обязательность	PK или FK	Таблица, на которую ссылается FK
	PNTG_TITLE	Название картины	VCHAR(35)	Xxxxxx...				
	PNTG_PRICE	Цена картины	NUMBER(9,2)	99,999.99	10.00—99,999.99	Да		
	PNTG_DATE_REC VD	Дата поступления картины	DATE	DD/MM/YY		Да		
	PTR_NUM	Номер художника	CHAR(4)	9999	1000—9999	Да	FK	PAINTER
	GAL_NUM	Номер галереи	CHAR(4)	9999	1000—9999		FK	GALLERY

Примечание:

FK — внешний ключ

PK — первичный ключ

CHAR — символьные данные фиксированной длины от 1 до 255 символов

VARCHAR (VCHAR) — символьные данные переменной длины (от 1 до 2000 символов). В Oracle обозначаются как VARCHAR2

NUMBER — числовые данные; NUMBER(9,2) используется для отображения чисел с двумя знаками после запятой общим количеством до 9 знаков, включая десятичную точку. Некоторые СУБД допускают использование типа данных MONEY или CURRENCY

Используя информацию, представленную на рис. 5.55 и в табл. 5.8, мы можем записать следующую структуру таблиц и специальных условий:

PAINTER (PTR_NUM, PTR_LASTNAME, PTR_FIRSTNAME, PTR_INITIAL,
PTR_AREACODE, PTR_PHONE)

GALLERY (GAL_NUM, GAL_OWNER, GAL_AREACODE, GAL_PHONE,
GAL_RATE)

PAINTING (PNTG_NUM, PNTG_TITLE, PNTG_PRICE, PTR_NUM, GAL_NUM)

Примечание

PTR_NUM является внешним ключом, который ссылается на таблицу PAINTER. Поскольку связь между PAINTER и PAINTING обязательна, внешний ключ PTR_NUM должен классифицироваться как NOT NULL. Атрибут GAL_NUM является внешним ключом, который ссылается на таблицу GALLERY. Поскольку связь между PAINTING и GALLERY необязательна, внешний ключ GAL_NUM может иметь пустые значения. Атрибут PNTG_NUM является первичным ключом и должен поэтому классифицироваться как NOT NULL.

Используя приведенную структуру таблиц базы данных ARTIST, посмотрим, что будет происходить при выполнении следующих действий.

- ❑ *Удаление художника (строки) из таблицы PAINTER.* Если мы удалим строку (художника) из таблицы PAINTER, то в таблице PAINTING будет иметься ссылка на несуществующего художника, т. е. мы имеем дело с аномалией удаления. (Картина не перестает существовать только потому, что художник отсутствует в таблице.) Принимая во внимание эту ситуацию, будет разумно запретить удалять строки из таблицы, если в другой таблице имеется внешний ключ, который ссылается на нее. То есть мы должны наложить ограничение на удаление (DELETE RESTRICT) на такую таблицу. Ограничение означает, что мы можем удалить художника из таблицы PAINTER только в том случае, если отсутствует внешний ключ в другой таблице, который требует наличия этого художника.

Фактический эффект этого ограничения очень простой. Предположим, что мы хотим удалить художника с номером PTR_NUM = 123 из базы данных ARTIST. Ограничение DELETE RESTRICT требует, чтобы, прежде всего, были удалены все строки в таблице PAINTING, в которых PTR_NUM = 123 используется в качестве внешнего ключа. Этим самым мы предохраняем пользователя от возможных последствий удаления художника (PAINTER). Можно, конечно, заставить СУБД удалить в таблице PAINTING все строки, соответствующие удаляемому художнику (DELETE CASCADE), но мы так не поступаем по только что указанным причинам

- ❑ *Добавление художника (строки) в таблицу PAINTER.* Добавление художника не вызывает каких-либо проблем, поскольку таблица PAINTER не имеет каких-либо зависимостей в других таблицах.
- ❑ *Изменения значений в ключевых столбцах таблицы PAINTER.* Изменение ключевых значений вызывает проблемы в базе данных, поскольку некоторые картины в таблице PAINTING могут ссылаться на этот ключ. Решение простое: нужно позаботиться о том, чтобы изменения атрибута PTR_NUM таблицы PAINTER автоматически вызвали необходимые изменения в ключе PTR_NUM, имеющемся в других

таблицах. Поскольку одно изменение каскадом отражается во всех таблицах, такой процесс называется каскадным обновлением (UPDATE CASCADE). Другими словами, требование UPDATE CASCADE означает, что все ссылки внешнего ключа на первичный ключ обновляются при обновлении первичного ключа.

- *Удаление галереи (строки) из таблицы GALLERY.* Удаление строки в таблице GALLERY создает проблемы в БД, если в таблице PAINTING имеются строки, которые ссылаются на первичный ключ строки таблицы GALLERY.

Поскольку таблица GALLERY необязательна для таблицы PAINTING, мы можем присвоить всем полям удаляемой строки GUL_NUM пустое значение или предупредить пользователя БД о возможных проблемах, определив оператор DELETE RESTRICT в таблице GALLERY. Выражение DELETE RESTRICT означает, что удаление строки в таблице GALLERY разрешается только при отсутствии в другой таблице внешнего ключа (GUL_NUM), требующего наличия этой строки в таблице GALLERY.

- *Добавление галереи (строки) в таблицу GALLERY.* Добавление новой строки не вызывает каких-либо проблем в БД, поскольку таблица GALLERY не имеет каких-либо зависимостей в других таблицах. (На новую строку не ссылается какой-либо внешний ключ, указывающий на таблицу GALLERY.)
- *Обновление первичного ключа таблицы GALLERY.* Изменение значения первичного ключа в строке таблицы GALLERY требует, чтобы все внешние ключи, которые на него ссылаются, тоже были обновлены. Поэтому мы должны в этом случае использовать выражение UPDATE CASCADE.

С учетом этих рассуждений мы теперь можем преобразовать ER-модель в набор таблиц с помощью следующих SQL-команд.

1. Создание таблицы PAINTER.

```
CREATE TABLE PAINTER (  
    PTR_NUM          CHAR(4)      NOT NULL    UNIQUE,  
    PTR_LASTNAME     CHAR(15)     NOT NULL,  
    PTR_FIRSTNAME    CHAR(15),    NOT NULL  
    PTR_INITIAL      CHAR(1),  
    PTR_AREACODE     CHAR(3),  
    PTR_PHONE        CHAR(8),  
    PRIMARY KEY (PTR_NUM));
```

2. Создание таблицы GALLERY.

```
CREATE TABLE GALLERY (  
    GAL_NUM          CHAR(4)      NOT NULL    UNIQUE,  
    GAL_OWNER        CHAR(35),  
    GAL_AREACODE     CHAR(3)      NOT NULL,  
    GAL_PHONE        CHAR(8)      NOT NULL,  
    GAL_RATE         NUMBER(4,2),  
    PRIMARY KEY (GAL_NUM));
```

3. Создание таблицы PAINTING.

```
CREATE TABLE PAINTING (  
    PNTG_NUM          CHAR(4)      NOT NULL    UNIQUE,  
    PNTG_TITLE         CHAR(35),  
    PNTG_PRICE         NUMBER(9,2),  
    PTR_NUM            CHAR(4)      NOT NULL,  
    GAL_NUM            CHAR(4),  
    PRIMARY KEY (PNTG_NUM),  
    FOREIGN KEY (PTR_NUM) REFERENCES PAINTER  
ON DELETE RESTRICT  
ON UPDATE CASCADE,  
    FOREIGN KEY (GAL_NUM) REFERENCES GALLERY  
    ON DELETE RESTRICT  
ON UPDATE CASCADE;
```

Примечание

Целостность на уровне сущностей требует, чтобы первичный ключ был уникален и не имел пустых значений. Программное обеспечение РСУБД должно поддерживать целостность на уровне сущностей, поэтому в объявлении первичного ключа использованы выражения NOT NULL и UNIQUE. Тем не менее, эти выражения рекомендуются только для гарантии того, что словарь данных точно отражает требования к информации.

После того как таблицы созданы, можно вводить данные, выполнять запросы и создавать отчеты. Обратите внимание, что решение, принятое проектировщиком по управлению целостностью данных, отражено в правилах первичного ключа. Реализация решения варьируется в зависимости от существа задачи.

5.10. Общие правила, обеспечивающие связь таблиц в базе данных

Наши опыты с простой БД ARTIST позволяют нам теперь сформулировать общие правила, помогающие при создании любых табличных структур БД, где необходимо обеспечить ограничения целостности.

- Любой первичный ключ необходимо классифицировать как NOT NULL.

Примечание

Если прикладное программное обеспечение не поддерживает выражение NOT NULL, вам придется обеспечить выполнение этого условия с помощью различных приемов программирования. Это еще один аргумент в пользу программного обеспечения, удовлетворяющего требованиям стандарта ANSI SQL.

- Определяйте все внешние ключи в соответствии со следующими требованиями к бинарным связям.

Связи 1:M

Создавайте внешний ключ, разместив первичный ключ стороны "один" в таблице стороны "многие" связи. Сторона "один" называется *родительской таблицей*, а сторона "многие" — *зависимой таблицей*. Обратите внимание на следующие правила внешнего ключа.

- Если обе стороны внешнего ключа **ОБЯЗАТЕЛЬНЫ**:

Правила внешнего ключа: NOT NULL
 ON DELETE RESTRICT
 ON UPDATE CASCADE

- Если обе стороны внешнего ключа **НЕОБЯЗАТЕЛЬНЫ**:

Правила внешнего ключа: NULL ALLOWED
 ON DELETE SET NULL
 ON UPDATE CASCADE

- Если одна из сторон **ОБЯЗАТЕЛЬНА**, а другая **НЕОБЯЗАТЕЛЬНА**:

- ◊ если компоненты стороны "многие" и обязательные компоненты связи находятся по одну сторону ER-диаграммы, то условие NULL ALLOWED должно определяться для внешнего ключа зависимой таблицы. Правила внешнего ключа выглядят так:

Правила внешнего ключа: NULL ALLOWED
 ON DELETE SET NULL или
 ON DELETE RESTRICT
 ON UPDATE CASCADE

- ◊ если компоненты стороны "многие" и обязательные компоненты связи находятся по разные стороны ER-диаграммы, то для внешнего ключа зависимой таблицы необходимо определить условие NOT NULL. Удаление и обновление в родительской таблице внешнего ключа должно подчиняться ограничениям DELETE RESTRICT и UPDATE CASCADE.

Слабая сущность.

- Расположите ключ родительской таблицы (сильная сущность) в слабой сущности. Этот ключ слабой сущности будет составным ключом, состоящим из ключа родительской таблицы и потенциального ключа слабой сущности, если таковой имеется. (Проектировщик может принять решение о создании нового уникального идентификатора сущности.)
- В связи слабой сущности используются те же правила, что и в связи 1:M, за исключением следующих ограничений на внешний ключ:

NOT NULL
ON DELETE CASCADE
ON UPDATE CASCADE

Связи M:N

Конвертируйте связь M:N в составную (переходную) сущность, состоящую (по крайней мере) из первичного ключа родительской таблицы. Первичный ключ составной сущности, таким образом, является составным ключом и подлежит ограничению NOT NULL.

Связи 1:1

Если обе сущности находятся в обязательной связи и не участвуют в других связях, вероятнее всего, обе эти сущности являются частью одной сущности.

Следующие примеры помогут глубже понять связи M:N, 1:M и 1:1.

Вариант 1: связь M:N, обе стороны обязательны (как определено в бизнес-правилах)

Исходную связь M:N нужно разделить на две связи 1:M, как показано на рис. 5.56.

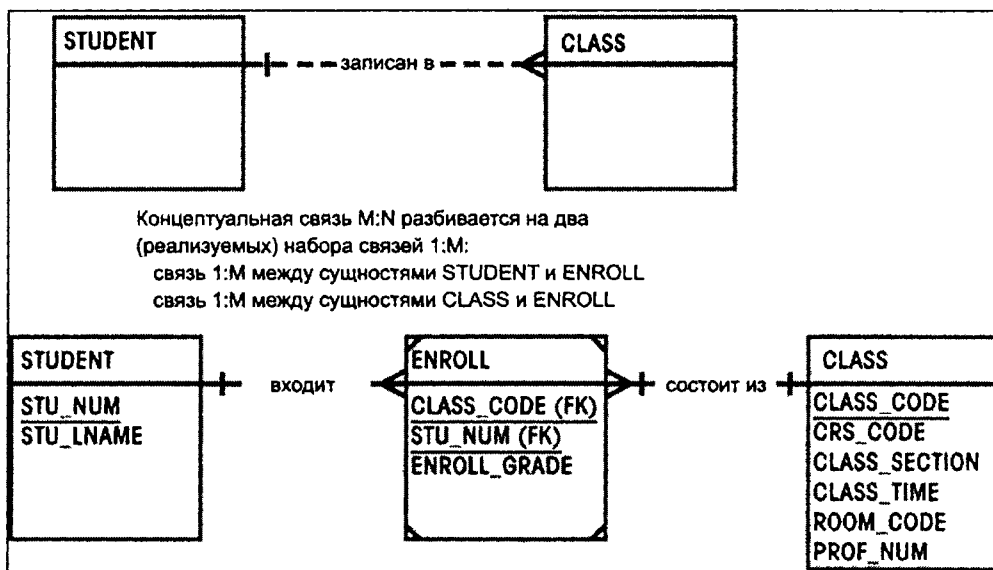


Рис. 5.56. Связи сущностей, M:N, обе стороны обязательны

Для таблицы ENROLL:

Внешний ключ = CLASS_CODE

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: мы не можем удалить группу, если в ней есть хотя бы один студент. Для удаления группы нужно, прежде всего, перевести всех студентов в другую группу.

Внешний ключ = STU_NUM

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: мы не можем удалить студента, если он один в группе, поскольку в обязательной связи в CLASS (группу) должен входить, по крайней мере, один студент (STUDENT).

Вариант 2: связь M:N, обе стороны необязательны (как определено в бизнес-правилах)

Исходную связь M:N нужно разделить на две связи 1:M, как показано на рис. 5.57.

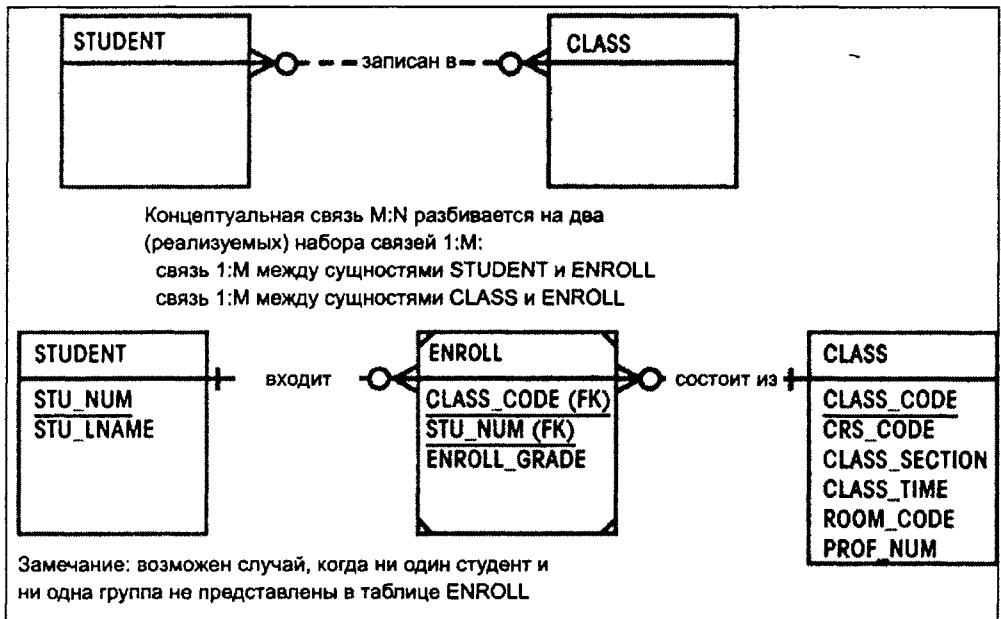


Рис. 5.57. Связи сущностей, M:N,
обе стороны необязательны

Для таблицы ENROLL, оба внешних ключа:

NOT NULL

ON DELETE CASCADE

ON UPDATE CASCADE

Пояснение: если связи необязательны, группа может существовать вообще без студентов. Точно так же студент может существовать, даже если он не записан ни в одну из групп.

Вариант 3: связь M:N, одна из сторон необязательна (как определено в бизнес-правилах)

Исходную связь M:N нужно разделить на две связи 1:M, как показано на рис. 5.58.

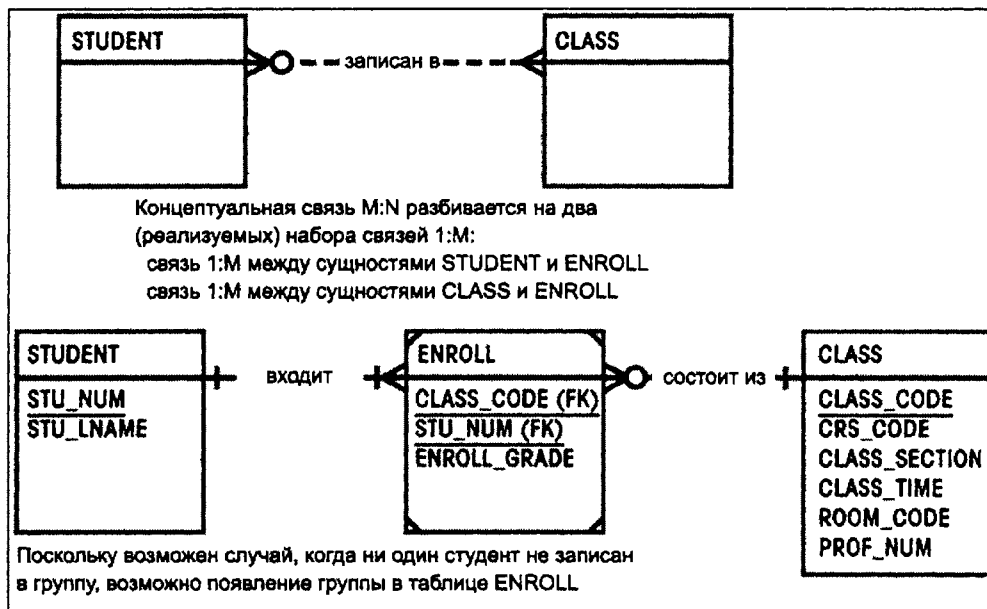


Рис. 5.58. Связи сущностей, M:N,
одна из сторон необязательна

Для таблицы ENROLL, действия на необязательной стороне:

Внешний ключ = STU_NUM

NOT NULL

ON DELETE CASCADE

ON UPDATE CASCADE

Пояснение: возможно, что в группе вообще нет студентов. Следовательно, удаление студента может сопровождаться удалением этого студента из группы (CLASS).

Для таблицы ENROLL, действия на обязательной стороне:

Внешний ключ = CLASS_CODE

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: нельзя удалять студента из группы, если это единственная группа, куда записан этот студент. (Студент должен быть записан, по крайней мере, в одну группу).

Вариант 4: связь 1:M, обе стороны обязательны

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.59.)

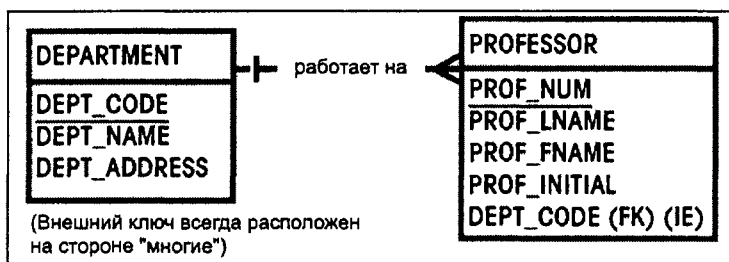


Рис. 5.59. Связь сущностей, 1:M, обе стороны обязательны

Примечание

Созданное в приложении Visio обозначение (IE) означает "Inversion Entry" (инверсионный компонент), который, как указывалось в гл. 3, в Visio определяется как "неуникальный идентификатор доступа к компоненту"). Это означает, что значение внешнего ключа не обязательно должно быть уникальным. И именно поэтому IE появляется в сильных сущностях/слабых (неидентифицируемых) связях (см. рис. 5.58). Если сущность слабая, то FK является частью PK, поэтому он не может быть пустым.

Сильная (идентифицируемая) связь отображается на диаграммах сплошной линией связи, слабая (неидентифицируемая) связь — штриховой линией.

Действия над таблицей PROFESSOR (преподаватель):

Внешний ключ = DEPT_CODE

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: нам не нужно удалять преподавателя при удалении факультета. (Прежде всего, необходимо перевести преподавателя на другой факультет.)

Вариант 5: связь 1:M, обе стороны необязательны

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.60.)

Действия над таблицей STUDENT (студент):

Внешний ключ = ROOM_CODE

NULL ALLOWED

ON DELETE SET NULL
или RESTRICT

Пояснение: если комната удаляется, то студент не может оставаться в этой комнате и будет вообще не иметь комнаты (NULL) до тех пор, пока ему не будет предоставлена другая комната.

ON UPDATE CASCADE

Объявление RESTRICT может использоваться для предупреждения пользователя о переназначении студенту другой комнаты.

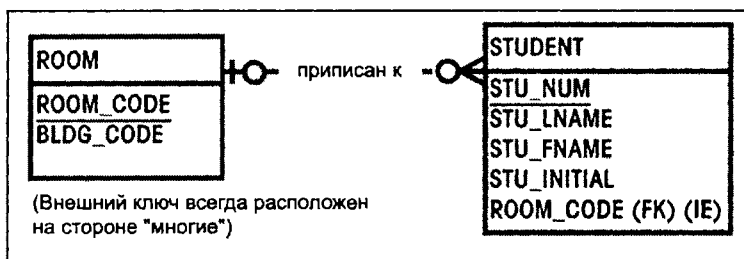


Рис. 5.60. Связь сущностей, 1:M, обе стороны необязательны

**Вариант 6: связь 1:M,
сторона "многие" необязательна,
сторона "один" обязательна**

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.61.)

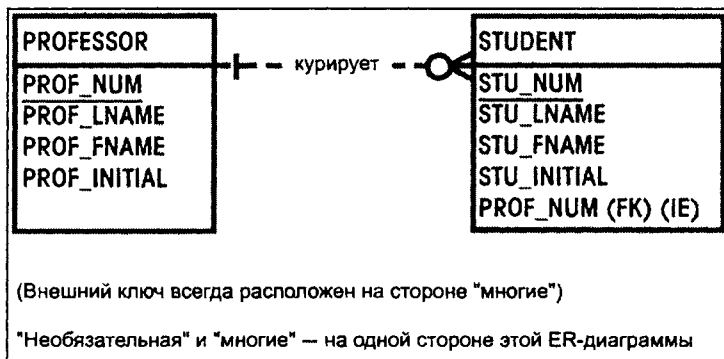


Рис. 5.61. Связь сущностей, 1:M, сторона "многие" необязательна, сторона "один" обязательна

Действия над таблицей STUDENT:

Внешний ключ = PROF_NUM

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: мы не хотим удалять студента при удалении преподавателя. (Прежде всего, передайте студентов другому куратору.)

Вариант 7: связь 1:M, сторона "один" необязательна, сторона "многие" обязательна

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.62.)

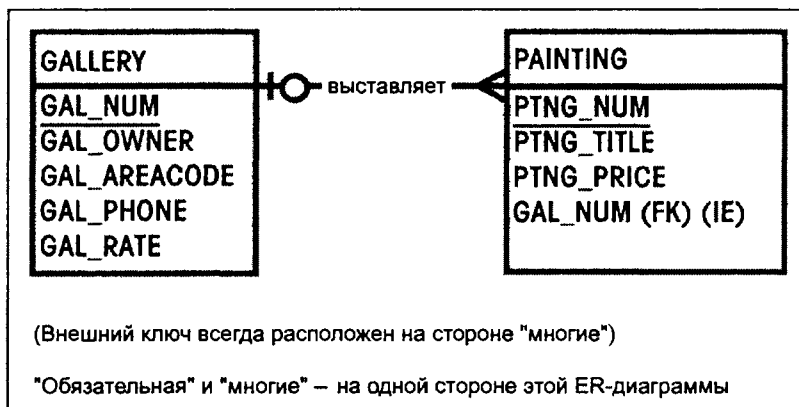


Рис. 5.62. Связь сущностей, 1:M,
сторона "один" необязательна, сторона "многие" обязательна

Действия над стороной PAINTING:

Внешний ключ = GAL_NUM

NULL ALLOWED

ON DELETE SET NULL или RESTRICT

ON UPDATE CASCADE

Пояснение: мы не хотим удалять картины при удалении галереи. GAL_NUM в таблице PAINTING может принимать пустое значение. Объявление RESTRICT может использоваться для предупреждения пользователя о возможном удалении галереи, в которой есть картины.

Вариант 8: связь 1:1, обе стороны обязательны

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.63.)

Внешний ключ = ROOM_CODE

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: мы не хотим удалять комнату, если в нее поселен преподаватель. (Нужно, прежде всего, переселить преподавателя в другую комнату.)

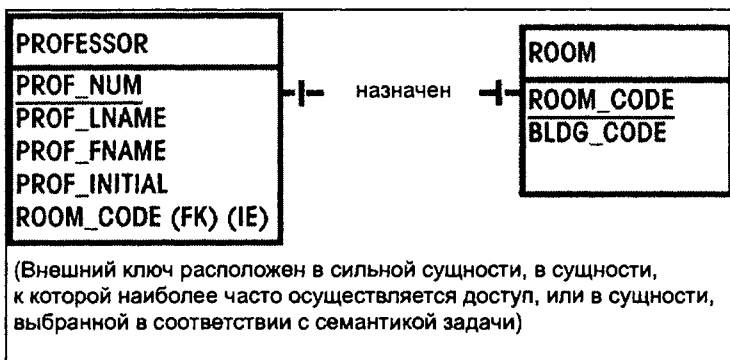


Рис. 5.63. Связь сущностей, 1:1, обе стороны обязательны

Вариант 9: связь 1:1, обе стороны необязательны

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.64.)

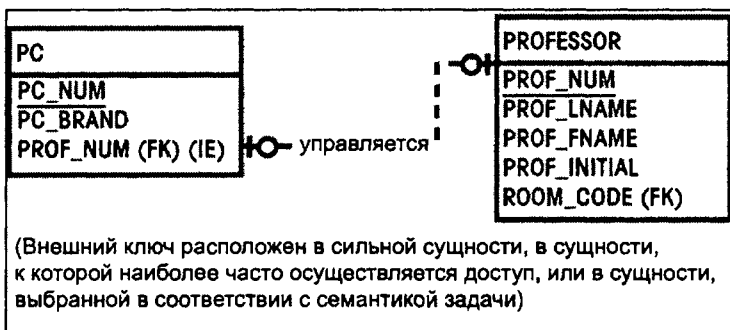


Рис. 5.64. Связь сущностей, 1:1, обе стороны необязательны

Внешний ключ = PROF_NUM

NOT ALLOWED

ON DELETE SET NULL

ON UPDATE CASCADE

Пояснение: преподаватель не обязательно работает на персональном компьютере, и не обязательно, что компьютер выдан преподавателю. Если преподавателя удаляют, атрибуту PROF_NUM присваивается пустое значение в сущности PC.

Вариант 10: связь 1:1, одна сторона необязательна, другая обязательна

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.65.)

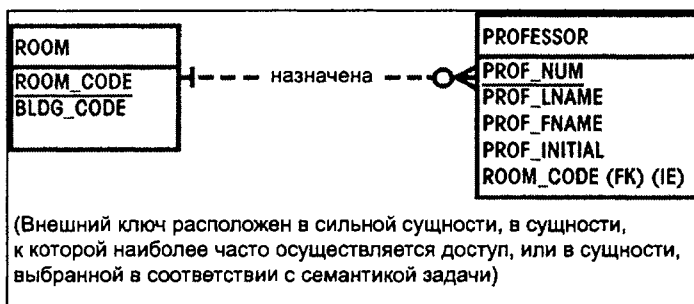


Рис. 5.65. Связь сущностей, 1:1,
одна сторона необязательна,
а другая сторона обязательна

Примечание

Выберите один из представленных внешних ключей. Если выбран первый, то ROOM_CODE в таблице PROFESSOR будет внешним ключом к таблице ROOM, а PROF_NUM в таблице ROOM будет удален. Если выбран второй, PROF_NUM останется внешним ключом в таблице ROOM, а ROOM_CODE в таблице PROFESSOR будет удален.

- ❑ Внешний ключ на необязательном элементе:

Внешний ключ = ROOM_CODE

NOT NULL

ON DELETE RESTRICT

ON UPDATE CASCADE

Пояснение: атрибут ROOM (комната) обязателен для преподавателя, но комната может быть пустой. Если преподавателя, поселенного в определенную комнату, удалить, то комната становится вакантной (SET NULL).

- ❑ Внешний ключ на обязательном элементе:

Внешний ключ = PROF_NUM

NOT ALLOWED

ON DELETE SET NULL

ON UPDATE CASCADE

Пояснение: мы не хотим удалять комнату, если в нее поселен преподаватель. (Прежде всего, необходимо переселить преподавателя в другую комнату, а затем удалить комнату.)

Вариант 11: слабая сущность (внешний ключ размещен в слабой сущности)

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.66.)

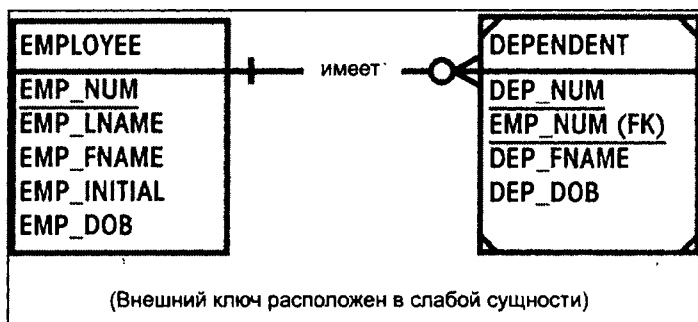


Рис. 5.66. Связь сущностей, слабая сущность

Внешний ключ = EMP_NUM

NOT NULL

ON DELETE CASCADE

ON UPDATE CASCADE

Пояснение: при увольнении сотрудника его иждивенцев также нужно удалить из базы данных.

Вариант 12: многозначные атрибуты (новая таблица в связи 1:M, в новой таблице внешний ключ CAR_VIN)

(Используйте фрагмент ER-диаграммы, представленный на рис. 5.67.)

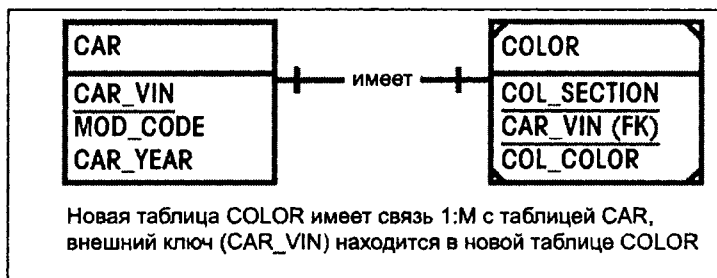


Рис. 5.67. Связь сущностей, многозначные атрибуты

Все 12 приведенных вариантов правил сведены в табл. 5.9.

Таблица 5.9. Правила внешнего ключа

Связь	Местоположение внешнего ключа	Участие сущностей в связи	Условие на ключ	Правила внешнего ключа	
				удаление	обновление
M:N	Новая сущность: составной ключ	Обе обязательны	NN	R	C
		Обе необязательны	NN	C	C
		Одна обязательна, а другая нет:			
		обязательная сторона	NN	R	C
		необязательная сторона	NN	SN или R	C
1:M	Сторона "многие"	Обе обязательны	NN	R	C
		Обе необязательны	NA	SN или R	C
		Одна обязательная, а другая нет:			
		обязательная сторона	NA	SN или R	C
		необязательная сторона	NN	R	C
1:1	Размещение внешнего ключа зависит от точки зрения. Разместите FK на необязательной стороне ER-диаграммы, на сильной сущности, на стороне, обращение к которой происходит чаще всего, или на стороне, выбор которой диктуется семантическими проблемами. Не размещаете FK на обеих сторонах!	Обе обязательны	NN	R	C
		Обе необязательны	NA	SN	C

Таблица 5.9 (окончание)

Связь	Местоположение внешнего ключа	Участие сущностей в связи	Условие на ключ	Правила внешнего ключа	
				удаление	обновление
Слабая Многозначные атрибуты	Слабая сущность Создайте набор новых таблиц, находящихся в связи 1:М. Далее действуйте в соответствии с правилами для слабой сущности	Одна обязательна, а другая нет: обязательная сторона необязательная сторона	NA	SN	C
			NN	R	C
			NN*	C	C
			NN	C	C

Примечание:

NN — NOT NULL

SN — SET TO NULL

NA — NULL ALLOWED

C — CASCADE

R — RESTRICT

* — Наследуется от родительской сущности

Резюме

SQL-команды можно подразделить на три основных набора процедур: определение данных, управление данными и манипулирование данными (запросы).

Определение данных

- ❑ Создание схемы базы данных. В стандартном SQL используется следующее определение схемы:

```
CREATE SCHEMA AUTHORIZATION <автор>
```

- ❑ Создание таблиц базы данных. В стандартном SQL используется следующая структура команды:

```
CREATE TABLE <имя таблицы>
```

Управление данными и запросы

- ❑ Оператор INSERT используется для вставки данных в таблицу.
- ❑ Оператор UPDATE используется для изменения (обновления) данных в таблице.
- ❑ Оператор DELETE используется для удаления строк таблицы.
- ❑ Оператор SELECT — основа запросов SQL.

В большинстве реализаций SQL требуется, чтобы SQL-команды разделялись точкой с запятой, однако в некоторых версиях SQL это требование отсутствует. Ознакомьтесь с руководством пользователя, чтобы выяснить правильный синтаксис команд.

В приведенной ниже сводке команд SQL рассматриваются SQL-запросы на основе таблицы STUDENT, которая содержит атрибуты STU_NUM, STU_LNAME, STU_FNAME, STU_INITIAL, STU_DOB, STU_STATE, STU_GPA и DEPT_CODE. Атрибут STU_NUM является первичным ключом таблицы STUDENT.

- ❑ *Создание таблицы STUDENT.* Выражение NOT NULL используется в сочетании с первичным ключом для обеспечения целостности на уровне сущностей.

```
CREATE TABLE STUDENT (  
    STU_NUM          FCHAR(6)          NOT NULL    UNIQUE,  
    STU_LNAME        VARCHAR(15)       NOT NULL,  
    STU_FNAME        VARCHAR(15)       NOT NULL,  
    STU_INITIAL      FCHAR(1),  
    STU_DOB          DATE,  
    STU_STATE        FCHAR(2),  
    STU_GPA          NUMBER(4,1),  
    DEPT_CODE        VARCHAR(8),  
    PRIMARY KEY (STU_NUM),  
    FOREIGN KEY DEPT_CODE REFERENCES DEPARTMENT);
```

❑ *Ввод данных в таблицу STUDENT.*

```
INSERT INTO STUDENT
VALUES ('231234', 'Smith', 'Jennie', 'B', '11/12/1976', 'TN', 3.67,
'INFS');
```

❑ *Список всех атрибутов таблицы по всем строкам (символ * называется метасимволом и означает "все строки")*

```
SELECT *
FROM STUDENT;
```

❑ *Список выбранных атрибутов по всем строкам.*

```
SELECT STU_LNAME, STU_GPA, DEPT_CODE
FROM STUDENT;
```

❑ *Список выбранных атрибутов для студента по имени George W. Batman.*

```
SELECT STU_LNAME, STU_GPA, DEPT_CODE
FROM STUDENT
WHERE STU_LNAME = 'Batmann'
AND STU_FNAME = 'George'
AND STU_INITIAL = 'W'
```

❑ *Сохранение таблицы.*

```
COMMIT STUDENT;
```

❑ *Внесение изменений.*

```
UPDATE STUDENT
SET STU_LNAME = 'Bronner', STU_FNAME = 'Jane', STU_INITIAL = 'G',
STU_DOB = '10/12/1973', STU_STATE = 'KY', STU_GPA = 2.95,
DEPT_CODE = 'ACCTG';
```

❑ *Удаление строки.*

```
DELETE FROM STUDENT
WHERE STU_NUM = '123456';
```

❑ *Восстановление содержимого таблицы.*

```
ROLLBACK;
```

❑ *Использование математических операций для ограничения вывода по запросу.*
К математическим операциям относятся:

- = (равно)
- < (меньше)
- > (больше)
- <= (меньше или равно)

- `>=` (больше или равно)
- `<>` (не равно)

Математические операции могут применяться к строкам и к числам. Сравнение строк производится слева направо и основывается на кодах ASCII. Из такого соглашения следует, что 'AB' < 'BB'.

```
SELECT *
FROM STUDENT
WHERE STU_GPA >= 3.25;
```

- *Использование атрибута даты для ограничения вывода по запросу.*

```
SELECT *
FROM STUDENT
WHERE STU_DOB < '01/01/1940';
```

- *Использование логических операторов для ограничения вывода по запросу.* К логическим операторам относятся AND, OR и NOT.

```
SELECT STU_FNAME, STU_LNAME, STU_GPA
FROM STUDENT
WHERE STUDENT_DOB < '01/01/1940'
AND STU_GPA >= 3.25
OR STU_GPA < 2.00;
```

К логическим операторам применимо правило старшинства. Поэтому приведенная команда отличается от следующей команды:

```
SELECT STU_FNAME, STU_LNAME, STU_GPA
FROM STUDENT
WHERE STUDENT_DOB < '01/01/1940'
AND (STU_GPA >= 3.25 OR STU_GPA < 2.00);
SELECT STU_FNAME, STU_LNAME, STU_GPA
FROM STUDENT
WHERE STU_STATE NOT 'TN';
```

- *Задание диапазона атрибута внутри запроса.* Команда BETWEEN.

```
SELECT *
FROM STUDENT
WHERE GPA BETWEEN 3.00 AND 3.50;
```

Некоторые реализации SQL не поддерживают оператор BETWEEN. Тот же запрос можно выполнить с помощью математических операций сравнения:

```
SELECT *
FROM STUDENT
WHERE STU_GPA >= 3.00
AND STU_GPA <= 3.50;
```

☐ Проверка пустых значений.

```
SELECT *  
FROM STUDENT  
WHERE STU_DOB IS NULL;
```

☐ Проверка на вхождение значения в некоторое подмножество. Следующий запрос выводит строки для студентов с номерами 123456, или 234567, или 456789.

```
SELECT *  
FROM STUDENT  
WHERE STU_NUM IN ('123456', '234567', '456789');
```

☐ Проверка наличия определенных значений внутри атрибута.

```
SELECT *  
FROM STUDENT  
WHERE STU_NUM EXISTS;
```

☐ Изменение структуры таблицы.

- Изменение свойств атрибута

```
ALTER TABLE STUDENT  
MODIFY (STU_GPA NUMBER(5,3));
```

Примечание

В большинстве реализаций SQL нельзя изменять тип данных атрибута, если столбец не пустой.

- Добавление столбца

```
ALTER TABLE STUDENT  
ADD (STU_PHONE CHAR(8));
```
- Обновление нескольких столбцов

```
ALTER TABLE STUDENT  
ADD (STU_PHONE CHAR(8));  
STU_HONOR CHAR(12));
```

☐ Обновление данных

- Одиночный элемент

```
UPDATE STUDENT  
SET STU_GPA = 2.79  
WHERE STU_NUM = '213-32-0433';
```
- Многократное обновление с помощью присвоения

```
UPDATE STUDENT  
SET STU_HONOR = 'President's list'  
WHERE STU_GPA > 3.75;
```

□ *Копирование части таблицы.*

- Создание новой табличной структуры.

```
CREATE TABLE TEMP (STU_NUMBER CHAR(6) NOT NULL, STU_LNAME CHAR(15));
```

- Копирование значений совпадающих столбцов в новую таблицу.

```
INSERT INTO TEMP (STU_NUMBER, STU_LNAME)
SELECT STU_NUM, STU_LNAME
FROM STUDENT
```

□ *Удаление таблицы из базы данных.*

```
DROP TABLE <имя таблицы>;
```

- *Чувствительность к регистру.* SQL различает символы верхнего и нижнего регистров (считает разными фамилии 'Smith' и 'SMITH'). В некоторых реализациях SQL поддерживается функция UPPER, позволяющая выполнять сравнение независимо от регистра.

```
SELECT *
FROM STUDENT
WHERE UPPER (STU_LNAME) = 'SMITH';
```

- *Частичное совпадение строк.* Для этого используются метасимволы % или _.

```
SELECT *
FROM STUDENT
WHERE STU_LNAME LIKE 'Smith%';
```

```
SELECT *
FROM STUDENT
WHERE STU_LNAME LIKE 'J_ns_n';
```

```
SELECT *
FROM STUDENT
WHERE STU_LNAME LIKE 'J_ns%';

SELECT *
FROM STUDENT
WHERE STU_LNAME NOT LIKE 'Smith%';
```

- *Упорядоченный список.* Для упорядочивания выводимого списка используется выражение ORDER BY <атрибут(ы)> или, при необходимости вывода в убывающем порядке, выражение ORDER BY <атрибут(ы)> DESC.

```
SELECT *
FROM STUDENT
ORDER BY STU_NAME, STU_GPA;

SELECT *
FROM STUDENT
ORDER BY STU_NAME, STU_GPA DESC;
```

- ❑ *Устранение дублирования в распечатке.*

```
SELECT DISTINCT STU_GPA
FROM STUDENT;
```

- ❑ *Числовые функции SQL.* Сюда относятся функции COUNT, MIN, MAX, AVG, SUM. Следующая команда выводит список студентов с различными средними баллами (GPA).

```
SELECT COUNT(DISTINCT STU_GPA)
FROM STUDENT;
```

Следующая команда выводит список студентов с лучшими средними баллами.

```
SELECT MAX(STU_GPA)
FROM STUDENT;
```

Следующая команда выводит средний балл по всем студентам.

```
SELECT AVG(STU_GPA)
FROM STUDENT;
```

Следующая команда выводит общую сумму средних баллов студентов.

```
SELECT SUM(STU_GPA)
FROM STUDENT;
```

Чтобы определить атрибуты, ассоциированные с результатами применения этих числовых функций, можно использовать вложенный запрос.

```
SELECT SUM(STU_GPA)
FROM STUDENT
WHERE STU_GPA = (SELECT MAX(STU_GPA)
FROM STUDENT);
```

- ❑ *Группировка данных.*

```
SELECT STU_GPA, STU_STATE
FROM STUDENT
GROUP BY STU_GPA;
```

Выражение GROUP BY обычно используется в сочетании с арифметическими функциями. Если вывод ограничивается определенными записями, то оператор WHERE использовать нельзя, вместо него используется оператор HAVING.

- ❑ *Создание представления (виртуальной таблицы).* Оператор VIEW.

```
CREATE VIEW TEMP_2
SELECT STU_NAME, STU_GPA
FROM STUDENT
WHERE STU_STATE = 'TN';
```


□ *Создание SQL-индекса.*

```
CREATE INDEX NUMDEX
  ON STUDENT (STU_NUM);
```

Если индекс используется для предотвращения дублирования данных, можно применить модификатор UNIQUE.

```
CREATE UNIQUE INDEX NUMDEX
  ON STUDENT (STU_NUM);
```

□ *Соединение (объединение) таблиц.* Это действие выполняется при получении данных из двух и более таблиц. Источник данных (таблица) должен быть обязательно идентифицирован. Предположим, что имеются две таблицы — PROD и VENDOR. Атрибуты таблицы PROD — P_NUM (первичный ключ), P_DESCRIPTOR и V_CODE. Атрибуты таблицы VENDOR — V_CODE (первичный ключ) и V_NAME. Содержимое атрибутов P_DESCRIPTOR и V_NAME можно получить с помощью команды:

```
SELECT PROD.P_DESCRIPTOR, VENDOR.V_NAME
  FROM PROD, VENDOR
 WHERE PROD.V_CODE = VENDOR.V_CODE;
```

К объединенным таблицам можно применить любую из перечисленных ранее SQL-команд и функций.

Процедурный SQL позволяет совмещать процедурный код и SQL в одном исполняемом файле. Процедурный код выполняется СУБД — опосредованно или напрямую — при инициировании его конечным пользователем.

Процедурный SQL (PL/SQL) можно использовать при создании триггеров, хранимых процедур и функций PL/SQL. Триггером называется код процедурного SQL, автоматически вызываемый СУБД при определенном событии манипулирования данными — обновлении, вставке, удалении. Триггер всегда ассоциирован с таблицей базы данных. У каждой таблицы БД может быть один или более триггеров. Триггер выполняется как часть транзакции, которая его инициирует.

Хранимая процедура — это именованный набор операторов процедурного SQL. Поскольку хранимые процедуры выполняются как единый модуль, их использование уменьшает объем сетевого трафика, что способствует повышению производительности системы в целом.

При конвертировании ER-модели в набор таблиц базы данных необходимо следовать правилам внешнего ключа (см. табл. 5.9).

Основные термины

ALTER	COMMIT	CREATE TABLE	DROP
AND	COUNT	CREATE VIEW	DISTINCT
AVG	CREATE DOMAIN	DELETE	EXIST
BETWEEN	CREATE INDEX	DOMAIN	GROUP BY

IN	MAX	OR	SELECT
INSERT	MIN	ORDER BY	SUM
IS NULL	NOT	ROLLBACK	UPDATE
LIKE			

Булева алгебра, алгебра логики — Boolean algebra

Вложенный запрос — nested query

Внешнее соединение — outer JOIN

Зависимая таблица — dependent table

Зарезервированные слова — reserved words

Каскадная упорядоченная последовательность — cascading order sequence

Левостороннее внешнее соединение — left outer JOIN

Метасимвол — wildcard character

Обновляемое представление — updatable view

Пакетная процедура обновления — batch update routine

Правило старшинства — rule of precedence

Правостороннее внешнее соединение — right outer JOIN

Процедурный SQL — procedural SQL (PL/SQL)

Псевдоним — alias

Пустое значение — null

Родительская таблица — parent table

Тип данных — data type

Триггер — trigger

Хранимая процедура — stored procedure

Хранимая функция — stored function

Язык структурированных запросов — SQL (Structured Query Language)

Вопросы

Все вопросы основаны на данных, представленных на рис. 5.68. Эта таблица базы данных называется EMP_1. Структура таблицы представлена в табл. 5.10.

Таблица 5.10. Содержимое таблицы EMP_1

Атрибут (поле)	Тип данных
EMP_NUM	CHAR(3)
EMP_LNAME	VARCHAR(15)
EMP_FNAME	VARCHAR(15)

Таблица 5.10 (окончание)

Атрибут (поле)	Тип данных
EMP_INITIAL	CHAR(1)
EMP_HIREDATE	DATE
JOB_CODE	CHAR(3)

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-94	502
102	Senior	David	H	12-Jul-87	501
103	Arbough	June	E	01-Dec-96	500
104	Ramores	Anne	K	15-Nov-85	501
105	Johnson	Alice	K	01-Feb-93	502
106	Smithfield	William		23-Jun-90	500
107	Alonzo	Maria	D	10-Oct-91	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501

Рис. 5.68. Содержимое таблицы EMP_1

Используя приведенные данные, ответьте на следующие вопросы.

1. Напишите SQL-код, создающий табличную структуру.
2. По табличной структуре, созданной в вопросе 1, напишите SQL-код, который вводит первые две строки данных в таблицу EMP_1.
3. Вставьте оставшиеся записи (см. рис. 5.63) в таблицу EMP_1, затем напишите SQL-код, который выводит список всех атрибутов для кода специальности EMP_NUM = 502.
4. Напишите SQL-код, который сохраняет таблицу EMP_1.
5. Напишите SQL-код для изменения кода специальности на 501 для персоны с идентификационным номером 106. После этого проверьте результаты, а затем отмените изменение кода специальности (верните прежнее значение).
6. Напишите SQL-код удаления строки для сотрудника с именем William Smithfield, который был принят на работу 06/23/90, код специализации 500. (Подсказка: используйте логические операции, в которые включите всю исходную информацию этой задачи.)
7. Напишите SQL-код, который восстанавливает данные в исходном виде: т. е. таблица должна содержать информацию, которая в ней имелаась до того, как вы сделали изменения в соответствии с вопросами 5 и 6.
8. Создайте копию таблицы EMP_1 с именем EMP_2, затем добавьте в эту структуру атрибуты EMP_PCT и PROJ_NUM. В атрибуте EMP_PCT хранится процент премиальных, выплачиваемых каждому сотруднику. Свойства новых атрибутов представлены ниже.

EMP_PCT NUMBER(4,2)
 PROJ_NUM CHAR(3)

(Примечание: если ваша реализация SQL позволяет, можно использовать формат DECIMAL(4,2) вместо NUMBER(4,2)).

9. Напишите SQL-код ввода значения EMP_PCT = 3.85 для сотрудника с персональным номером EMP_NUM = 103. Предположим, что все элементы EMP_PCT в настоящий момент времени определены, и что таблица EMP_2 теперь содержит данные, представленные на рис. 5.69.

	EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_PCT	PROJ_NUM
▶	101	News	John	G	08-Nov-94	502	5.00	
	102	Senior	David	H	12-Jul-87	501	8.00	
	103	Arbough	June	E	01-Dec-96	500	3.85	
	104	Ramoras	Anne	K	15-Nov-85	501	10.00	
	105	Johnson	Alice	K	01-Feb-93	502	5.00	
	106	Smithfield	William		23-Jun-90	500	6.20	
	107	Alonzo	Maria	D	10-Oct-91	500	5.15	
	108	Washington	Ralph	B	22-Aug-89	501	10.00	
	109	Smith	Larry	W	18-Jul-99	501	2.00	

Рис. 5.69. Содержимое таблицы EMP_2

10. Используя простую командную последовательность, напишите SQL-код с помощью которого можно ввести номер проекта PROJ_NUM = 18 для всех сотрудников? имеющих специальность JOB_CODE = 500.
11. Используя простую командную последовательность, напишите SQL-код, с помощью которого можно ввести номер проекта PROJ_NUM = 25 для всех сотрудников, имеющих специальность JOB_CODE = 502 и выше. После решения вопросов 10 и 11 в таблице EMP_2 будет содержаться информация, представленная на рис. 5.70.

	EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE	EMP_PCT	PROJ_NUM
▶	101	News	John	G	08-Nov-94	502	5.00	25
	102	Senior	David	H	12-Jul-87	501	8.00	
	103	Arbough	June	E	01-Dec-96	500	3.85	18
	104	Ramoras	Anne	K	15-Nov-85	501	10.00	
	105	Johnson	Alice	K	01-Feb-93	502	5.00	25
	106	Smithfield	William		23-Jun-90	500	6.20	18
	107	Alonzo	Maria	D	10-Oct-91	500	5.15	18
	108	Washington	Ralph	B	22-Aug-89	501	10.00	
	109	Smith	Larry	W	18-Jul-99	501	2.00	

Рис. 5.70. Содержимое таблицы EMP_2 после модификации

На этом этапе лучше сохранить эту таблицу еще раз.

12. Напишите SQL-код, с помощью которого можно ввести значение PROJ_NUM = 14 для сотрудников, принятых на работу до 1 января 1992 года и код специальности которых не менее 501. (Можно считать, что эта таблица соответствует условиям вопроса 11.)

13. Напишите две командных последовательности SQL, которые необходимы для:
 - создания временной таблицы с именем TEMP_1, структура которой составлена из атрибутов EMP_NUM и EMP_PCT таблицы EMP_2;
 - копирования совпадающих значений таблицы EMP_2 в таблицу TEMP_1.
14. Напишите SQL-команду, с помощью которой можно удалить из базы данных вновь созданную таблицу TEMP_1.
15. Напишите SQL-код, позволяющий вывести список всех сотрудников, фамилия которых начинается со "Smith". Другими словами, в этот список должны войти строки и Smith, и Smithfield.
16. Предположим, что ваша БД содержит таблицу PROJECT, представленную на рис. 5.71. Атрибут EMP_NUM в таблице PROJECT ссылается на сотрудников в таблице EMPLOYEE_2, которые руководят проектом.

	PROJ_NUM	PROJ_NAME	EMP_NUM
►	5	Evergreen	103
	18	Amber Wave	108
	22	Rolling Tide	102
	25	Starflight	107

Рис. 5.71. Содержимое таблицы PROJECT

Используя эту информацию, напишите SQL-код, с помощью которого можно получить результаты, представленные на рис. 5.72.

	PROJ_NAME	EMP_LNAME	EMP_FNAME	EMP_INITIAL	JOB_CODE
►	Evergreen	Arbough	June	E	500
	Amber Wave	Washington	Ralph	B	501
	Rolling Tide	Senior	David	H	501
	Starflight	Alonzo	Maria	D	500

Рис. 5.72. Результаты запроса для вопроса 16

(Подсказка: необходимо соединить таблицы EMP_1 и PROJECT.)

17. Напишите SQL-код, с помощью которого можно создать виртуальную таблицу с именем REP_1, содержащую информацию, представленную в вопросе 16.
18. Напишите SQL-код для поиска среднего значения премиальных в таблице EMP_2.
19. Напишите SQL-код, который выводит список данных таблицы EMP_2 в возрастающем порядке значений премиальных.
20. Напишите SQL-код, который выводит список только отличающихся номеров проекта, найденных в таблице EMP_2.

Задачи

1. Используя базу данных CH2_AVIA_CO, рассмотренную в гл. 2, используйте ее словарь данных в качестве основы для реализации этой БД. (БД должна содержать таблицы AIRCRAFT, CHARTER, CUSTOMER, EMPLOYEE, MODEL и PILOT.) Переименуйте эту базу данных в CH5_AVIA. С помощью функции импорта вашей СУБД импортируйте данные из БД CH2_AVIA_CO (эта база данных — CH5_AVIA — будет основой для первых 15 задач).
2. Используя содержимое таблицы CHARTER, напишите SQL-запрос, результат выполнения которого представлен на рис. 5.73. Обратите внимание, что вывод ограничен указанными атрибутами для самолета 2778V.

	CHAR_DATE	AC_NUMBER	CHAR_DESTINATION	CHAR_DISTANCE	CHAR_HOURS_FLOWN
▶	16-Jan-2002	2778V	BNA	320	1.6
	17-Jan-2002	2778V	GNV	1574	7.9
	19-Jan-2002	2778V	MOB	884	4.8
	20-Jan-2002	2778V	MQY	312	1.5

Рис. 5.73. Выборка атрибутов из таблицы CHARTER для самолета 2778V

3. Создайте виртуальную таблицу (с именем AC2778V), содержащую вывод, представленный в задаче 2.
4. Создайте результат, представленный на рис. 5.74, для самолета 2778V. Обратите внимание, что этот результат включает данные из таблиц CHARTER и CUSTOMER (Подсказка: используйте операцию JOIN.)

	CHAR_DATE	AC_NUMBER	CHAR_DESTINATION	CUS_LNAME	CUS_AREACODE	CUS_PHONE
▶	19-Jan-2002	2778V	MOB	Ramas	615	644-2573
	20-Jan-2002	2778V	MQY	Dunne	713	894-1238
	17-Jan-2002	2778V	GNV	Smith	615	894-2285
	16-Jan-2002	2778V	BNA	Brown	615	297-1228

Рис. 5.74. Выборка из таблиц CHARTER и CUSTOMER

5. Создайте результат, представленный на рис. 5.75. Обратите внимание, что в этот вывод включены данные из таблиц CHARTER, EMPLOYEE и MODEL. (Подсказка: Два оператора JOIN охватывают разные таблицы. Например, "соединение" между CHARTER и MODEL требует существования таблицы AIRCRAFT, поскольку таблица CHARTER не содержит внешнего ключа для таблицы MODEL. Однако в таблице CHARTER содержится внешний ключ для таблицы AIRCRAFT, в которой содержится внешний ключ для таблицы MODEL.)
6. Модифицируйте запрос в задаче 5 с тем, чтобы включить данные из таблицы CUSTOMER. В настоящее время вывод ограничен выбором даты, как показано на рис. 5.76. Используйте выражение WHERE CHAR_DATE >= '5/15/1999' для определения ограничения на дату в запросе.

	CHAR_DATE	AC_NUMBER	MOD_NAME	EMP_LNAME
▶	16-Jan-2002	2778V	Navajo Chieftain	Lewis
	17-Jan-2002	2778V	Navajo Chieftain	Lange
	19-Jan-2002	2778V	Navajo Chieftain	Lewis
	20-Jan-2002	2778V	Navajo Chieftain	Travis

Рис. 5.75. Выборка из таблиц CHARTER, EMPLOYEE и MODEL

	CHAR_DATE	AC_NUMBER	MOD_NAME	EMP_LNAME	CUS_LNAME
▶	21-Jan-2002	1484P	Aztec	Lewis	Orlando
	20-Jan-2002	2289L	KingAir	Lange	Brown
	20-Jan-2002	2778V	Navajo Chieftain	Travis	Dunne
	20-Jan-2002	4278Y	Navajo Chieftain	Duzak	Williams
	21-Jan-2002	4278Y	Navajo Chieftain	Williams	Williams

Рис. 5.76. Выборка из таблиц CHARTER, EMPLOYEE, MODEL и CUSTOMER

7. Модифицируйте запрос задачи 6 для создания вывода, представленного на рис. 5.77. Ограничение по датам то же, что и в задаче 6.

	CHAR_DATE	CHAR_DESTINATION	AC_NUMBER	MOD_CHG_MILE	CHAR_DISTANCE	CUS_LNAME
▶	21-Jan-2002	STL	1484P	\$1.93		508 Orlando
	20-Jan-2002	GNV	2289L	\$2.67		1645 Brown
	20-Jan-2002	MGY	2778V	\$2.35		312 Dunne
	20-Jan-2002	ATL	4278Y	\$2.35		936 Williams
	21-Jan-2002	TYS	4278Y	\$2.35		644 Williams

Рис. 5.77. Выборка из таблиц CHARTER, MODEL и CUSTOMER

8. Модифицируйте запрос задачи 7 для создания вычисляемых (производных) атрибутов "расход топлива в час" и "общая стоимость мили полета" (Подсказка: можно использовать SQL для получения вычисляемых "атрибутов", которые не хранятся в какой-то таблице. Например, безусловно допустим следующий SQL-запрос:

```
SELECT CHAR_DISTANCE, CHAR_GALLONS, CHAR_DISTANCE/CHAR_GALLONS
FROM CHARTER
WHERE CHAR_DATE >= '1/20/02';
```

Используйте подобную технику на объединенных таблицах для создания вывода, представленного на рис. 5.78.

Примечание

Формат вывода определяется используемой РСУБД. В данном случае применяется формат MS Access, когда выводится заголовок с названием "Expr1" для обозначения результата операции деления

```
[CHARTER].[CHAR_FUEL_GALLONS]/[CHARTER].[CHAR_HOURS]
```

созданной строителем выражений MS Access. В Oracle по умолчанию используется полное название операции. Необходимо тщательно изучить руководство пользователя РСУБД. Например, MS Access допускает использование окна свойств для переопределения формата вывода, что позволит обеспечить вывод, представленный на рис. 5.79.

	CHAR_DATE	AC_NUMBER	MOD_NAME	CHAR_HOURS_FLOWN	CHAR_FUEL_GALLONS	Expr1
▶	20-Jan-2002	4278Y	Navajo Chieftain	6.1	302.6	49.606557377049
	20-Jan-2002	2289L	King Air	6.7	459.5	68.582089552239
	20-Jan-2002	2778V	Navajo Chieftain	1.5	67.2	44.8
	21-Jan-2002	1484P	Aztec	3.1	105.5	34.032258064516
	21-Jan-2002	4278Y	Navajo Chieftain	3.8	167.4	44.052631578947

Рис. 5.78. Использование SQL-запроса для получения вычисляемого (производного) атрибута

	CHAR_DATE	AC_NUMBER	MOD_NAME	CHAR_HOURS_FLOWN	CHAR_FUEL_GALLONS	Gallons/Hour
▶	20-Jan-2002	4278Y	Navajo Chieftain	6.1	302.6	49.61
	20-Jan-2002	2289L	King Air	6.7	459.5	68.58
	20-Jan-2002	2778V	Navajo Chieftain	1.5	67.2	44.80
	21-Jan-2002	1484P	Aztec	3.1	105.5	34.03
	21-Jan-2002	4278Y	Navajo Chieftain	3.8	167.4	44.05

Рис. 5.79. Использование системного программного обеспечения для форматирования вывода

9. Модифицируйте запрос, созданный в задаче 8, для создания вывода, представленного на рис. 5.80. Обратите внимание, что в данном случае вычисляемые атрибуты производятся от двух разных таблиц. (*Подсказка:* таблица MODEL содержит стоимость мили полета, а в таблице CHARTER содержится суммарное расстояние. Заметим, что вывод упорядочен по дате, а внутри даты — по фамилии клиента.)

	CHAR_DATE	CUS_LNAME	CHAR_DISTANCE	MOD_CHG_MILE	Mileage Charge
▶	20-Jan-2002	Brown	1645	\$2.67	\$4,392.15
	20-Jan-2002	Dunne	312	\$2.35	\$733.20
	20-Jan-2002	Williams	936	\$2.35	\$2,199.60
	21-Jan-2002	Orlando	508	\$1.93	\$980.44
	21-Jan-2002	Williams	644	\$2.35	\$1,513.40

Рис. 5.80. Использование SQL-запроса для вычисления стоимости мили полета

10. Используйте технологию решения задачи 9 для расчета затрат, представленных на рис. 5.81. Общие затраты для клиента рассчитываются так:

Длина маршрута * стоимость мили полета

Часы ожидания * \$50 в час

Длина маршрута (CHAR_DISTANCE) берется из таблицы CHARTER. Стоимость мили полета (MOD_MILECHARGE) берется из таблицы MODEL, а количество часов простоя (CHAR_WAIT) — из таблицы CHARTER.

	CHAR_DATE	CUS_LNAME	Mileage Charge	Waiting Charge	Total Charge
▶	20-Jan-2002	Brown	\$4,392.15	\$0.00	\$4,392.15
	20-Jan-2002	Dunne	\$733.20	\$0.00	\$733.20
	20-Jan-2002	Williams	\$2,199.60	\$105.00	\$2,304.60
	21-Jan-2002	Orlando	\$980.44	\$0.00	\$980.44
	21-Jan-2002	Williams	\$1,513.40	\$225.00	\$1,738.40

Рис. 5.81. Использование SQL-запроса для расчета затрат клиента

11. Напишите SQL-запрос, создающий список клиентов, у которых имеется неоплаченный остаток. Формат вывода представлен на рис. 5.82. Заметим, что остатки перечислены в порядке убывания.

	CUS_LNAME	CUS_FNAME	CUS_INITIAL	CUS_BALANCE
▶	Olowski	Paul	F	\$1,285.19
	O'Brian	Amy	B	\$1,014.56
	Smith	Kathy	W	\$896.54
	Orlando	Myron		\$673.21
	Smith	Olette	K	\$453.98

Рис. 5.82. Список клиентов с неоплаченными задолженностями

12. Найдите среднее значение неоплаченных задолженностей клиентов, максимальный остаток, минимальный остаток и общую сумму неоплаченных задолженностей. Результаты должны иметь вид, представленный на рис. 5.83.

	Average Balance	Minimum Balance	Maximum Balance	Total Unpaid Bills
▶	\$432.35	\$0.00	\$1,285.19	\$4,323.48

Рис. 5.83. Среднее по неоплаченным задолженностям

13. Используя таблицу CHARTER в качестве источника, сгруппируйте данные по самолетам, затем используйте SQL-функции для создания результатов, представленных на рис. 5.84. (Для заголовков мы использовали специальную программу, поэтому у вас заголовки могут иметь другой вид.)

	AC_NUMBER	Number of Trips	Total Distance	Average Distance	Total Hours	Average Hours
▶	484F	4	1976	494.0	12.0	3.0
	2289L	4	5178	1294.5	24.1	6.0
	2778V	4	3090	772.5	15.8	4.0
	4278Y	6	5268	878.0	30.4	5.1

Рис. 5.84. Суммарные данные по самолетам

14. Напишите SQL-код для создания вывода, представленного на рис. 5.85. Обратите внимание, что в список включены избранные (selected) атрибуты таблицы CHARTER для всех рейсов, в которых нет второго пилота. (Подсказка: попробуйте использовать функцию IS NULL! И помните, что фамилия пилота требует доступа к таблице EMPLOYEE, в то время как модель самолета можно получить из таблицы MODEL.)

	CHAR_DATE	AC_NUMBER	MOD_NAME	CHAR_PILOT	EMP_LNAME	CHAR_HOURS_FLOWN
▶	16-Jan-2002	2778V	Navajo Chieftain	101	Lewis	1.6
	16-Jan-2002	2289L	KingAir	104	Lange	5.1
	17-Jan-2002	4278Y	Navajo Chieftain	109	Travis	2.8
	17-Jan-2002	2289L	KingAir	101	Lewis	5.7
	17-Jan-2002	1484P	Aztec	106	Duzak	2.9
	18-Jan-2002	4278Y	Navajo Chieftain	109	Travis	6.2
	18-Jan-2002	2289L	KingAir	105	Williams	6.6
	18-Jan-2002	1484P	Aztec	106	Duzak	4.1
	19-Jan-2002	4278Y	Navajo Chieftain	105	Williams	3.9
	19-Jan-2002	2778V	Navajo Chieftain	101	Lewis	4.8
	20-Jan-2002	4278Y	Navajo Chieftain	106	Duzak	6.1
	21-Jan-2002	1484P	Aztec	101	Lewis	3.1

Рис. 5.85. Список всех чартерных рейсов без второго пилота

15. Напишите SQL-код, который будет обновлять коды ресурса конструкции самолета и ресурса двигателя в таблице AIRCRAFT с учетом кода налетанных каждым самолетом часов из таблицы CHARTER. (Подсказка: для того чтобы восстановить исходные значения таблицы AIRCRAFT, используйте оператор ROLLBACK после решения этой задачи.)

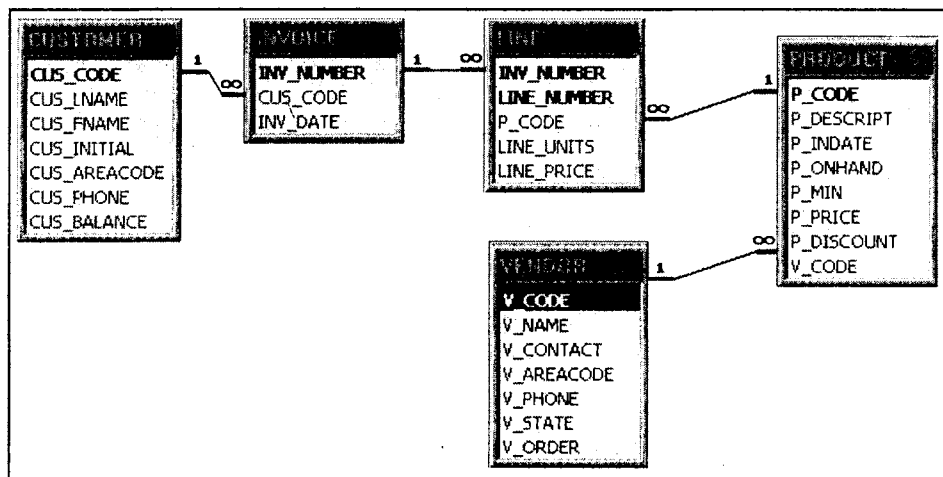


Рис. 5.86. Реляционная схема БД CH5_SALE

При решении задач с 16 по 33 используйте базу данных CH5_SALE_CO. Реляционная схема этой БД представлена на рис. 5.86. На рис. 5.87 представлено содержимое таблиц CUSTOMER, INVOICE, LINE, PRODUCT и VENDOR.

16. Постройте базу данных CH5_SALE_CO и структуру ее таблиц. Используйте в качестве руководства списки данных для определения их свойств. Например, PROD_CODE всегда состоит из 8 символов, поэтому его можно определить как FCHAR(8). (Естественно, если ваша PCУБД не поддерживает формат FCHAR, используйте формат CHAR(8)).
17. Введите данные в таблицы, которые вы определили в задаче 16.
18. Создайте список всех покупок, сделанных клиентами, используя в качестве образца вывод, представленный на рис. 5.88. (Подсказка: в таблице INVOICE используйте оператор ORDER BY CUS_CODE.)
19. Используя вывод, представленный на рис. 5.89, создайте расчет стоимости покупок клиента, включая промежуточное суммирование по строкам счета. (Подсказка: модифицируйте формат запроса задачи 18, удалите столбец INV_DATE, затем добавьте вычисляемый атрибут LINE_UNITS * LINR_PRICE для расчета промежуточных сумм.)

База данных CH5_SALE_CO						
Таблица CUSTOMER						
CUS_CODE	CUS_LASTNAME	CUS_FIRSTNAME	CUS_INITIAL	CUS_AREACODE	CUS_PHONE	CUS_BALANCE
10010	Ramas	Alfred	A	615	844-2573	\$0.00
10011	Dunne	Leona	K	713	894-1238	\$0.00
10012	Smith	Kathy	W	615	834-2285	\$345.86
10013	Olowski	Paul	F	615	894-2180	\$536.75
10014	Orlando	Myron		615	222-1672	\$0.00
10015	O'Brian	Amy	B	713	442-3381	\$0.00
10016	Brown	James	G	615	297-1228	\$221.19
10017	Williams	George		615	290-2556	\$788.93
10018	Farriss	Anne	G	713	362-7185	\$216.55
10019	Smith	Olette	K	615	297-3809	\$0.00
0						\$0.00

Таблица INVOICE						
INV_NUMBER	CUS_CODE	INV_DATE				
1001	10014	Wednesday, August 18, 1999				
1002	10011	Wednesday, August 18, 1999				
1003	10012	Wednesday, August 18, 1999				
1004	10011	Thursday, August 19, 1999				
1005	10013	Thursday, August 19, 1999				
1006	10014	Thursday, August 19, 1999				
1007	10015	Thursday, August 19, 1999				
1008	10011	Thursday, August 19, 1999				

Таблица LINE						
INV_NUMBER	LINE_NUMBER	SP_CODE	LINE_UNITS	LINE_PRICE		
1001	1	13-Q2P2	1	\$14.99		
1001	2	23109-HB	1	\$9.95		
1002	1	54778-2T	2	\$4.99		
1003	1	2238/GPD	1	\$38.95		
1003	2	1548-QQ2	1	\$39.95		
1003	3	13-Q2P2	5	\$14.99		
1004	1	54778-2T	3	\$4.99		
1004	2	23109-HB	2	\$9.95		
1005	1	PVC23DRT	12	\$5.87		
1006	1	SM-18277	3	\$6.99		
1006	2	2232/3TY	1	\$109.92		
1006	3	23109-HB	1	\$9.95		
1006	4	89-VRE-Q	1	\$256.99		
1007	1	13-Q2P2	2	\$14.99		
1007	2	54778-2T	1	\$4.99		
1008	1	PVC23DRT	5	\$5.87		
1008	2	VR3/TT3	3	\$119.95		
1008	3	23109-HB	1	\$9.95		

Рис. 5.87. Содержимое таблиц базы данных CH5_SALE

Таблица PRODUCT

P_CODE	P_DESCRIPTION	P_INDATE	P_ONHAND	P_PRICE
1QER31	Power painter, 15 psi, 3-nozzle	Wednesday, June 02, 1999	8	\$109.99
13-Q2/P2	7.25-in. pwr. saw blade	Monday, July 12, 1999	32	\$14.99
14-Q1/L3	9.00-in. pwr. saw blade	Saturday, June 12, 1999	18	\$17.49
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	Saturday, August 14, 1999	15	\$39.95
1558-QW/1	Hrd. cloth, 1/2-in., 3x50	Saturday, August 14, 1999	23	\$43.99
2232/QTY	B&D jigsaw, 12-in. blade	Thursday, July 29, 1999	8	\$109.92
2232/QWE	B&D jigsaw, 8-in. blade	Friday, July 23, 1999	6	\$99.87
2238/QPD	B&D cordless drill, 1/2-in.	Thursday, August 19, 1999	12	\$38.95
23109-HB	Claw hammer	Thursday, August 19, 1999	23	\$9.95
23114-AA	Sledge hammer, 12 lb.	Sunday, August 01, 1999	8	\$14.40
54778-2T	Rat-tail file, 1/8-in. fine	Wednesday, July 14, 1999	43	\$4.99
89-WRE-Q	Hicut chain saw, 16 in.	Monday, September 06, 1999	11	\$256.99
PVC23DRT	PVC pipe, 3.5-in., 8-ft	Sunday, September 19, 1999	188	\$5.87
SM-18277	1.25-in. metal screw, 25	Tuesday, September 28, 1999	172	\$6.99
SW-23116	2.5-in. wd. screw, 50	Thursday, September 23, 1999	237	\$8.45
WR3/TT3	Steel matting, 4'x8'x1/8", .5" mesh	Monday, August 16, 1999	18	\$119.95

Таблица VENDOR

V_CODE	V_NAME	V_CONTACT	V_AREACODE	V_PHONE	V_STATE	V_ORDER
21225	Bryson, Inc.	Smithson	615	223-3234	TN	Y
21226	SuperLoo, Inc.	Flushing	904	215-8995	FL	N
21231	D&E Supply	Singh	615	228-3245	TN	Y
21344	Gomez Bros.	Ortega	615	889-2546	KY	N
22567	Dome Supply	Smith	901	678-1419	GA	N
23119	Randssets Ltd.	Anderson	901	678-3998	GA	Y
24004	Brackman Bros.	Browning	615	228-1410	TN	N
24268	ORDVA, Inc.	Hakford	615	898-1234	TN	Y
25443	B&K, Inc.	Smith	904	227-0093	FL	N
25501	Damal Supplies	Smythe	615	890-3523	TN	N
25595	Rubicon Sis.	Orton	904	456-0092	FL	Y

Рис. 5.87. Содержимое таблиц базы данных CH5_SALE (окончание)

20. Теперь модифицируйте запрос задачи 19 для создания вывода, представленного на рис. 5.90.
21. Модифицируйте запрос задачи 20 для включения числа покупок отдельных товаров, сделанных каждым клиентом. (Другими словами, если счет клиента включает в себя три товара по одному на каждую строку — LINE_NUMBER, — то вы насчитаете три купленных товара. Если вы исследуете исходные данные счета, обратите внимание, что клиент 10011 инициировал создание трех счетов, содержащих всего шесть строк, каждая из которых представляет собой одну покупку.) Ваши выходные значения должны соответствовать рис. 5.91.

	CUS_CODE	INV_NUMBER	INV_DATE	P_DESCRIPTION	LINE	UNITS	LINE PRICE
▶	0011	1002	16-Jan-02	Rat-tail file, 1/8-in. fine	2		\$4.99
	10011	1004	17-Jan-02	Claw hammer	2		\$9.95
	10011	1004	17-Jan-02	Rat-tail file, 1/8-in. fine	3		\$4.99
	10011	1008	17-Jan-02	Claw hammer	1		\$9.95
	10011	1008	17-Jan-02	PVC pipe, 3.5-in., 8-ft	5		\$5.87
	10011	1008	17-Jan-02	Steel matting, 4'x8'x1/8", .5" mesh	3		\$119.95
	10012	1003	16-Jan-02	7.25-in. pwr. saw blade	5		\$14.99
	10012	1003	16-Jan-02	B&D cordless drill, 1/2-in.	1		\$38.95
	10012	1003	16-Jan-02	Hrd. cloth, 1/4-in., 2x50	1		\$39.95
	10014	1001	16-Jan-02	7.25-in. pwr. saw blade	1		\$14.99
	10014	1001	16-Jan-02	Claw hammer	1		\$9.95
	10014	1006	17-Jan-02	1.25-in. metal screw, 25	3		\$6.99
	10014	1006	17-Jan-02	B&D jigsaw, 12-in. blade	1		\$109.92
	10014	1006	17-Jan-02	Claw hammer	1		\$9.95
	10014	1006	17-Jan-02	Hicut chain saw, 16 in.	1		\$256.99
	10015	1007	17-Jan-02	7.25-in. pwr. saw blade	2		\$14.99
	10015	1007	17-Jan-02	Rat-tail file, 1/8-in. fine	1		\$4.99
	10018	1005	17-Jan-02	PVC pipe, 3.5-in., 8-ft	12		\$5.87

Рис. 5.88. Список покупок клиентов

	CUS_CODE	INV_NUMBER	P_DESCRIPTION	Units Bought	Unit Price	Subtotal
▶	0011	1002	Rat-tail file, 1/8-in. fine	2	\$4.99	\$9.98
	10011	1004	Claw hammer	2	\$9.95	\$19.90
	10011	1004	Rat-tail file, 1/8-in. fine	3	\$4.99	\$14.97
	10011	1008	Claw hammer	1	\$9.95	\$9.95
	10011	1008	PVC pipe, 3.5-in., 8-ft	5	\$5.87	\$29.35
	10011	1008	Steel matting, 4'x8'x1/8", .5" mesh	3	\$119.95	\$359.85
	10012	1003	7.25-in. pwr. saw blade	5	\$14.99	\$74.95
	10012	1003	B&D cordless drill, 1/2-in.	1	\$38.95	\$38.95
	10012	1003	Hrd. cloth, 1/4-in., 2x50	1	\$39.95	\$39.95
	10014	1001	7.25-in. pwr. saw blade	1	\$14.99	\$14.99
	10014	1001	Claw hammer	1	\$9.95	\$9.95
	10014	1006	1.25-in. metal screw, 25	3	\$6.99	\$20.97
	10014	1006	B&D jigsaw, 12-in. blade	1	\$109.92	\$109.92
	10014	1006	Claw hammer	1	\$9.95	\$9.95
	10014	1006	Hicut chain saw, 16 in.	1	\$256.99	\$256.99
	10015	1007	7.25-in. pwr. saw blade	2	\$14.99	\$29.98
	10015	1007	Rat-tail file, 1/8-in. fine	1	\$4.99	\$4.99
	10018	1005	PVC pipe, 3.5-in., 8-ft	12	\$5.87	\$70.44

Рис. 5.89. Расчеты стоимости покупок клиента с промежуточными суммами

22. Используйте вложенный запрос для вычисления средней стоимости товара, купленного каждым клиентом. (Подсказка: возьмите за основу результат задачи 21.) Ваш вывод должен соответствовать результатам, представленным на рис. 5.92. Заметим, что средняя стоимость покупки равна общей стоимости, деленной на число покупок.

	CUS_CODE	CUS_BALANCE	Total Purchases
▶	10011	\$0.00	\$444.00
	10012	\$345.86	\$153.85
	10014	\$0.00	\$422.77
	10015	\$0.00	\$34.97
	10018	\$216.55	\$70.44

Рис. 5.90. Общая стоимость покупок клиента

	CUS_CODE	CUS_BALANCE	Total Purchases	Number of Purchases
▶	10011	\$0.00	\$444.00	6
	10012	\$345.86	\$153.85	3
	10014	\$0.00	\$422.77	6
	10015	\$0.00	\$34.97	2
	10018	\$216.55	\$70.44	1

Рис. 5.91. Общий итог покупок клиента, включая количество сделанных покупок

	CUS_CODE	CUS_BALANCE	Total Purchases	Number of Purchases	Average Purchase Amount
▶	10011	\$0.00	\$444.00	6	\$74.00
	10012	\$345.86	\$153.85	3	\$51.28
	10014	\$0.00	\$422.77	6	\$70.46
	10015	\$0.00	\$34.97	2	\$17.48
	10018	\$216.55	\$70.44	1	\$70.44

Рис. 5.92. Средняя стоимость покупки, сделанной клиентом

23. Создайте запрос для получения общего числа покупок на один счет, чтобы получить результаты, представленные на рис. 5.93. Invoice Total — это общая стоимость купленных товаров в таблице LINE, соответствующая таблице INVOICE.

	INV_NUMBER	Invoice Total
▶	1001	\$24.94
	1002	\$9.98
	1003	\$153.85
	1004	\$34.87
	1005	\$70.44
	1006	\$397.83
	1007	\$34.97
	1008	\$399.15

Рис. 5.93. Общие суммы по счетам

24. Используйте запрос для представления счетов и сумм по счетам, как показано на рис. 5.94. (Подсказка: выполните группирование по атрибуту CUS_CODE.)

	CUS_CODE	INV_NUMBER	Invoice Total
▶	10011	1002	\$9.98
	10011	1004	\$34.87
	10011	1008	\$399.15
	10012	1003	\$153.85
	10014	1001	\$24.94
	10014	1006	\$397.83
	10015	1007	\$34.97
	10018	1005	\$70.44

Рис. 5.94. Суммы по счетам клиентов

25. Напишите запрос для получения числа счетов и общей стоимости покупок на клиента с помощью вывода, представленного на рис. 5.95. (Сравните эти суммы с результатами задачи 24.)

	CUS_CODE	Number of Invoices	Total Customer Purchases
▶	10011	3	\$444.00
	10012	1	\$153.85
	10014	2	\$422.77
	10015	1	\$34.97
	10018	1	\$70.44

Рис. 5.95. Общее число счетов и общая стоимость покупок по клиентам

26. Используя результаты запроса из задачи 25, напишите запрос для создания общего числа счетов, общей суммы по этим счетам, счета с наименьшей суммой, счета с наибольшей суммой и средней суммой по всем счетам. (Подсказка: проверьте число в выводе задачи 25 и обратите внимание, что средняя продажа, представленная на рис. 5.96, получается путем деления суммы (SUM) общей стоимости покупок клиента (Total Customer Purchases) из задачи 25 на сумму (SUM) числа счетов (Number of Invoices)). Вывод задачи должен совпадать с рис. 5.96.

	Total # of Invoices	Total Sales	Minimum Sale	Largest Sale	Average Sale
▶	3	\$1,126.03	\$34.97	\$444.00	\$225.21

Рис. 5.96. Общее число счетов, суммы по счетам, минимум, максимум и среднее по продажам

27. Составьте баланс клиентов, делавших покупки в течение данного периода обработки счетов, т. е. для клиентов, появившихся в таблице INVOICE. Результаты этого запроса представлены на рис. 5.97.
28. Используя результаты запроса, созданного в задаче 27, выполните суммарный баланс клиентов, как показано на рис. 5.98.

	CUS_CODE	CUS_BALANCE
▶	10011	\$0.00
	10012	\$345.86
	10014	\$0.00
	10015	\$0.00
	10018	\$216.55

Рис. 5.97. Текущий баланс клиентов, делавших покупки в течение данного периода обработки счетов

	Minimum Balance	Maximum Balance	Average Balance
▶	\$0.00	\$345.86	\$112.48

Рис. 5.98. Суммарный баланс клиентов, делавших покупки в течение данного периода обработки счетов

29. Создайте запрос для поиска баланса клиента по всем клиентам, включая все просроченные платежи. Результат запроса представлен на рис. 5.99.

	Total Balance	Minimum Balance	Maximum Balance	Average Balance
▶	\$2,089.28	\$0.00	\$768.93	\$208.93

Рис. 5.99. Суммарный баланс по всем клиентам

30. Получите список клиентов, не делавших покупки в течение данного периода обработки счетов. Вывод запроса должен совпадать с представленным на рис. 5.100.

CUS_CODE	CUS_BALANCE
10010	0
10013	536.75
10016	221.19
10017	768.93
10019	0

SQL> |

Рис. 5.100. Баланс для клиентов, не делавших покупки в течение данного периода обработки счетов

31. Найдите суммарный баланс для всех клиентов, не делавших покупки в течение данного периода обслуживания счетов. Результаты запроса представлены на рис. 5.101.
32. Создайте запрос для вывода суммарной стоимости товаров, находящихся на складе. Обратите внимание, что стоимость по каждому товару получается умно-

жением количества единиц товара на стоимость каждой единицы. Результаты выполнения запроса представлены на рис. 5.102.

SUM(CUS_BALANCE)	MIN(CUS_BALANCE)	MAX(CUS_BALANCE)	AUG(CUS_BALANCE)
1526.87	0	768.93	305.374

SQL> |

Рис. 5.101. Суммарный баланс клиентов, не делавших покупки в течение данного периода обработки счетов

	P_DESCRPT	P_ONHAND	P_PRICE	Subtotal
▶	Power painter, 15 psi, 3-nozzle	8	\$109.99	\$879.92
	7.25-in. pwr. saw blade	32	\$14.99	\$479.68
	9.00-in. pwr. saw blade	18	\$17.49	\$314.82
	Hrd. cloth, 1/4-in., 2x50	15	\$39.95	\$599.25
	Hrd. cloth, 1/2-in., 3x50	23	\$43.99	\$1,011.77
	B&D jigsaw, 12-in. blade	8	\$109.92	\$879.36
	B&D jigsaw, 8-in. blade	6	\$99.87	\$599.22
	B&D cordless drill, 1/2-in.	12	\$38.95	\$467.40
	Claw hammer	23	\$9.95	\$228.85
	Sledge hammer, 12 lb.	8	\$14.40	\$115.20
	Rat-tail file, 1/8-in. fine	43	\$4.99	\$214.57
	Hicut chain saw, 16 in.	11	\$256.99	\$2,826.89
	PVC pipe, 3.5-in., 8-ft	188	\$5.87	\$1,103.56
	1.25-in. metal screw, 25	172	\$6.99	\$1,202.28
	2.5-in. wd. screw, 50	237	\$8.45	\$2,002.65
	Steel matting, 4'x8'x1/8", .5" mesh	18	\$119.95	\$2,159.10

Рис. 5.102. Стоимость товаров, имеющихсся на складе

33. Используя результаты запроса задачи 32, найдите общую стоимость товаров на складе. Результат представлен на рис. 5.103.

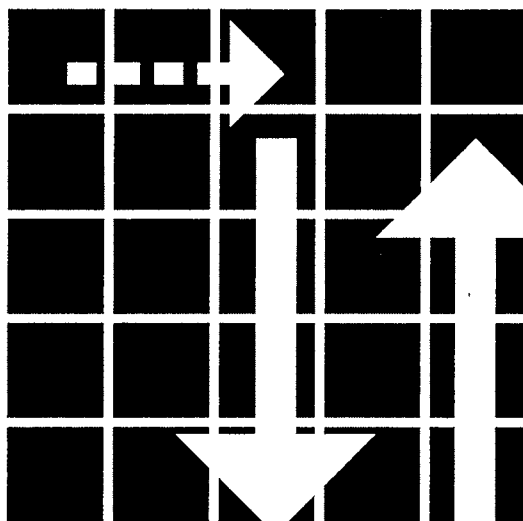
Total Value of Inventory
▶ \$15,084.52

Рис. 5.103. Общая стоимость товаров на складе

34. Используя базу данных CH5_AVIA, представленную в гл. 2 как CH2_AVIA и использованную в задачах с 1 по 15 данной главы, создайте триггер

TRG_CHAR_HOURS, автоматически обновляющий таблицу AIRCRAFT при добавлении новой строки в таблицу CHARTER. Используйте атрибут CHAR_HOURS_FLOWN для добавления значений AC_TTAF, AC_TTEL и AC_TTER в таблицу AIRCRAFT.

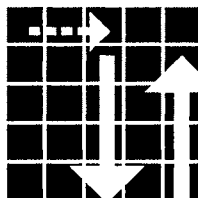
35. Используя базу данных CH5_SALE_CO, создайте триггер TRG_LINE_PROD, автоматически обновляющий код количества товара на складе после каждой продажи.
36. Используя базу данных CH5_SALE_CO, создайте процедуру PRC_CUS_BALANCE_UPDATE, которая получает номер счета в качестве параметра и обновляет баланс клиента. (*Подсказка: можно использовать раздел DECLARE для определения переменной TOTINV числового типа, в которой будут храниться расчетные суммы по счетам.*)
37. Используя процедурный язык SQL, напишите пакетную процедуру обновления, описанную в *разд. 5.7*.



ЧАСТЬ III

ЭФФЕКТИВНОЕ ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ БАЗ ДАННЫХ

Глава 6



Проект базы данных

Из этой главы вы узнаете следующее:

- ☐ что эффективный проект базы данных должен отражать все свойства информационной системы, частью которой он является;
- ☐ что информационные системы разрабатываются в интегрированной среде, называемой Systems Development Life Cycle (SDLC, жизненный цикл разработки систем);
- ☐ что в информационных системах даже самые успешные базы данных являются объектом постоянной оценки и исправлений в интегрированной системе, называемой Database Life Cycle (DBLC, жизненный цикл базы данных);
- ☐ как провести оценку и корректировку баз данных в средах SDLC и DBLC;
- ☐ что существуют стратегии проектирования баз данных: нисходящее или восходящее централизованное и децентрализованное проектирование.

Обзор

Проект базы данных — это фрагмент большой картины, которая называется информационной системой. В рамках информационной системы не только корректируют, хранят и извлекают информацию, но также структурируют и упорядочивают ее. Используя своевременную и значимую информацию, ответственные лица могут принимать решения, способствующие росту и процветанию компании. Проекты баз данных, не позиционирующие себя как составную часть этого общего целого, скорее всего, не будут иметь коммерческого успеха. То есть проектировщики БД должны отдавать себе отчет, что база данных является важным средством достижения цели, но не самой целью. (Менеджеры хотят, чтобы база данных соответствовала их целям и требованиям, но, судя по всему, многие проектировщики баз данных в свою очередь требуют, чтобы менеджеры подстраивались под требования самих баз данных.) Вы ведь не приобретаете автомобиль исключительно для того, чтобы стать его владельцем. Скорее, право собственности на автомобиль является лишь средством обеспечения вас надежным транспортом. Информационные системы не исключение, они являются продуктом тщательно организованного процесса эволюции. Для выработки требований к информационной системе и определения сферы ее действия используются методы системного анализа. Используя эти методы, мы создаем реальную информационную систему, и этот процесс называется разработкой систем.

Любое предприятие не есть нечто застывшее и статичное. Даже собственный экстенсивный рост компании требует расширения информации, поиска ее новых источников. Поэтому как только работа над какой-то информационной системой завершена, ее сразу же необходимо подвергнуть критической оценке. Выполняют ли системные приложения свойственные им функции? Как эти приложения можно изменить и расширить, чтобы они соответствовали изменяющимся требованиям? Должным ли образом выполняет база данных присущие ей функции хранилища информации, которые затем используются прикладными программами?

Рано или поздно даже лучшие информационные системы достигают критической точки. Если такая система более не отвечает изменившимся требованиям бизнеса, то как создать систему, которая отвечала бы этим требованиям? Перефразируя старую как мир истину, можно утверждать, что в сфере разработок постоянно только изменения. Непрерывный процесс разработки, сопровождения, расширения и модернизации представляет собой жизненный цикл разработки систем (Systems Development Life Cycle, SDLC).

Подобно основанным на них приложениям, базы данных тоже должны подвергаться постоянной оценке. После создания базы данных, в процессе сопровождения она постоянно модифицируется. Но наступает время, когда даже модернизация не сможет привести базу данных в соответствие техническим требованиям. Когда БД не может более адекватно выполнять свои функции, ее приходится заменять. Короче говоря, база данных существует в рамках своего жизненного цикла (Database Life Cycle, DBLC), который мы детально будем изучать в этой главе.

6.1. Структурирование данных

В гл. 1 мы говорили о том, что информация хранится в базе данных в неупорядоченном, неструктурированном виде. Например, агентство по кредитованию собирает данные, связанные с возрастом клиента, полом, доходом, трудовым стажем, долгами и т. д. Поскольку корректный сбор данных, их хранение и извлечение являются важнейшими условиями деятельности любого предприятия, эффективный проект базы данных имеет огромное значение. Даже на простейшем уровне розничной продажи поиск и извлечение данных для клиента при выполнении торговых операций является самым минимальным требованием к проекту БД. Необходимо определить, имеется ли товар в наличии, сколько единиц, какие позиции необходимо восполнить и какой поставщик производит товар, запасы которого необходимо пополнить. Фактически современный учет рассматривает информацию как самый ценный актив.

Неупорядоченная информация, не менее ценная, чем структурированные данные, редко бывает нужна руководителям в своем первоначальном виде. (В конце концов, даже если долго смотреть на компьютерные распечатки, где в случайном порядке перечислены все имеющиеся факты, вряд ли можно рассчитывать на внезапное озарение.) В гл. 1 мы говорили, что руководству необходима структурированная, упорядоченная информация, полученная на основе обработки данных и представленная в легко интерпретируемой форме.

Для руководителя структурированная информация так же отличается от неупорядоченных данных, как кирпичное здание от груды кирпичей. И точно так же, как гру-

ду кирпичей можно использовать для постройки здания любого размера, формы и назначения, на основе неупорядоченных данных, собранных вместе и преобразованных надлежащим образом, можно принимать правильные решения. Такие преобразования могут быть очень простыми как, например, табулирование, что позволяет лучше представить себе некоторые фрагменты данных. Например, если руководство считает, что распределение клиентов по возрасту и полу поможет выработать надлежащую стратегию, то вероятно, в этом случае можно применить технологию перекрестной классификации (cross-classification). Результаты такой перекрестной классификации отображаются в перекрестных таблицах (cross-classification tables). Пример перекрестной таблицы, показывающей распределение 9040 клиентов по возрасту и полу, представлен в табл. 6.1. Такие таблицы, наверное, — наиболее часто используемый способ организации данных и (возможно, в нашей практике) используются в широком спектре организационных и управленческих структур.

Таблица 6.1. Простая перекрестная таблица: преобразование данных

	До 25 лет	25–45	45–59	60 и выше	ВСЕГО
Мужчины	119	1892	2641	876	5528
Женщины	48	1117	1805	542	3512
ВСЕГО	167	3009	4446	1418	9040

Термин *преобразование (transformation)* описывает любой процесс структурирования информации. В основе преобразования данных может лежать простое табулирование, позволяющее получить данные в форме, представленной в табл. 6.1, составление подробного отчета или сложный комплекс статистических процедур. Какой бы метод преобразования ни использовался, принятие решения, как правило, основано на структурированных данных, полученных с помощью одного из этих преобразований. Без данных мы не можем выполнить никаких преобразований вообще; без преобразований мы не сможем получить структурированную информацию, которая служит основой принятия решений.

6.2. Информационная система

По существу *база данных* представляет собой тщательно разработанное и сконструированное хранилище сведений. Хранилище сведений есть часть некоего целого, называемого *информационной системой*. Информационная система предназначена для сбора данных, их хранения и извлечения. Она также обеспечивает структурирование данных и управление ими. Таким образом, законченная информационная система состоит из людей, оборудования, управляющего программного обеспечения, базы (или баз) данных, прикладных программ и процедур. *Системный анализ* — это процесс, который определяет необходимость разработки информационной системы и сферу ее действия. Процесс создания информационной системы называется *разработкой систем (systems development)*.

Примечание

Нужно иметь в виду, что эта глава не ставит цель рассмотреть все аспекты системного анализа и разработки систем, которые обычно являются предметом специальных курсов или отдельных книг. Однако эта глава поможет вам глубже понять процесс проектирования базы данных, ее реализацию и проблемы управления, на которые оказывает влияние информационная система, где база данных является важнейшим компонентом.

В структуре разработки систем приложения (applications) упорядочивают информацию и формируют основу принятия решений. С помощью приложений составляются отчеты, выполняется табулирование и создается графическое представление данных, что помогает проникнуть в суть проблемы. Из рис. 6.1 следует, что каждое приложение состоит из двух частей: данных и кода (программных инструкций), с помощью которого данные определенным образом структурируются. Данные и код, действуя совместно, позволяют моделировать реальные бизнес-функции и действия. В любой момент времени физически хранимая информация представляет собой, в сущности, мгновенный снимок предприятия. Но картина будет неполной, если мы не уясним деловые операции, представленные кодом (программой).



Рис. 6.1. Структурирование данных для принятия решений

Эффективность информационной системы зависит от следующей триады:

- ☐ проектирование базы данных и ее реализация;
- ☐ проектирование и реализация приложений;
- ☐ административные процедуры.

Хотя эта книга в основном посвящена первому сегменту этой триады (возможно, самому важному!) — проектированию и реализации базы данных, все же, если не обсудить и другие два сегмента, то мы вряд ли сможем разработать эффективную информационную систему. Разработка хорошей информационной системы — очень тяжелый труд: системный анализ и разработка требуют тщательного планирования, гарантирующего, что все действия будут согласованы, будут дополнять друг друга, и что вся работа будет завершена вовремя.

В широком смысле термин *разработка базы данных* используется для описания процесса проектирования БД и ее реализации. Основная цель проектирования баз данных состоит в создании полных, нормализованных, неизбыточных (по возможности) моделей баз данных, полностью интегрированных на концептуальном, логическом и физическом уровне. Этап реализации включает в себя создание структуры хранения базы данных, загрузку информации в базу данных и обеспечение управления данными.

Чтобы процедуры, описываемые в этой главе, были применимы во всех случаях, мы сосредоточимся на элементах, являющихся общими для всех информационных систем. Поэтому основная часть процессов и процедур не будет зависеть от размера, типа и сложности реализуемых баз данных. Например, вне зависимости от размера, типа и сложности БД необходимо составлять план, проводить анализ и выполнять проектирование. Процесс работы над проектом базы данных большой корпорации или даже ее сегмента нельзя оценить путем простого масштабирования действий, необходимых для создания небольшой базы данных (например, для соседнего обувного магазина). Если вновь использовать строительную аналогию, то можно сказать, что постройка небольшого здания требует отдельного проектирования точно так же, как строительство небоскреба Seattle Space Needle или знаменитого моста Golden Gate, но для строительства Space Needle и Golden Gate все же требуется более сложный и квалифицированный проект, чем для небольшого здания. Например, проектировщик здания может не принимать во внимание комплексное воздействие ветровой нагрузки на конструкцию дома или воздействие на нее близлежащих конструкций.

Хотя все представленные далее технологические процессы и процедуры соответствуют определенным стандартам, в действительности могут возникнуть различные специфические обстоятельства, определяющие иные подходы к проектированию БД. Ознакомившись со стандартными подходами, в последних разделах главы мы сравним нисходящие и восходящие, а также централизованные и децентрализованные стратегии проектирования.

6.3. Жизненный цикл разработки систем (SDLC)

Жизненный цикл разработки систем (Systems Development Life Cycle, SDLC) отслеживает историю (жизненный цикл) информационной системы. Может быть самым важным для системотехника является то, что SDLC предоставляет "полную картину", в рамках которой могут быть спланированы и качественно оценены проекты баз данных и разработка прикладных программ.

Почему мы уделяем такое пристальное внимание SDLC, хотя эта книга посвящена вопросам проектирования и реализации баз данных? Ответ очевиден: проектирование баз данных происходит в рамках проектирования информационной системы. Короче говоря, очень трудно отделить процесс проектирования базы данных от SDLC и наоборот.

Как следует из рис. 6.2, обычно SDLC подразделяется на пять этапов: планирование, анализ, детальное системное проектирование, реализация и сопровождение. Жизненный цикл разработки систем скорее итеративный, чем последовательный

процесс. Например, анализ технико-экономического обоснования поможет уточнить первоначальную оценку проекта, а сведения, полученные в процессе изучения технических требований пользователя к SDLC, помогут уточнить технико-экономическое обоснование.

Базы данных точно так же имеют свой жизненный цикл, как и все информационные системы, частью которых они являются. Поскольку жизненный цикл баз данных (DBLC) очень напоминает жизненный цикл разработки систем, краткое описание SDLC будет очень уместно.

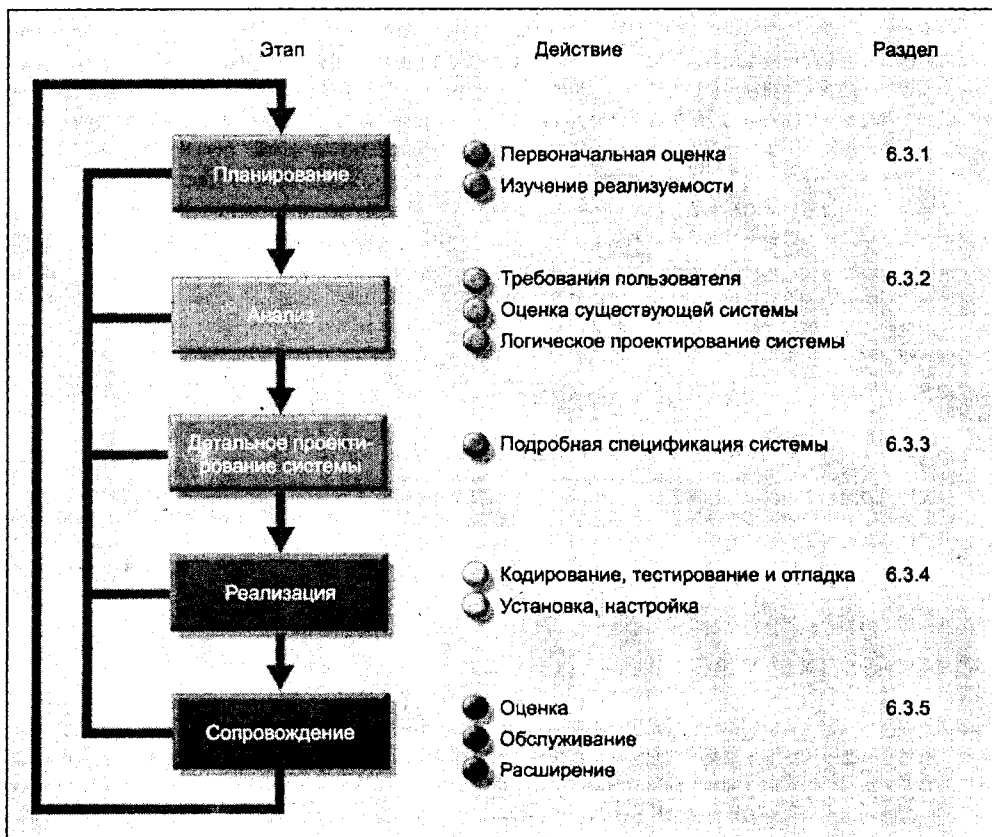


Рис. 6.2. Жизненный цикл разработки систем (SDLC)

6.3.1. Планирование

На этапе планирования SDLC создается общее представление о компании и ее задачах, дается первоначальная оценка информационных потоков на предприятии.

Эти мероприятия должны дать ответ на некоторые важные вопросы.

- ❑ *Стоит ли продолжать эксплуатацию имеющейся системы?* Если генератор информации успешно справляется со своей работой, то нет смысла его модифицировать или заменять. Помните старую истину "Не сломалось — не ремонтируй".
- ❑ *Стоит ли модифицировать имеющуюся систему?* Если первичная оценка определит недостаточный объем информационных потоков, то может потребоваться небольшая (а может быть, и значительная) модернизация. Стороны, задействованные в первичной оценке, во время принятия решения о модификации всегда должны соотносить желаемое и возможное.
- ❑ *Стоит ли менять существующую информационную систему?* В процессе первичной оценки может обнаружиться, что устранить недостатки системы нельзя. Принимая во внимание большой объем работ, который придется выполнить при создании новой системы, нужно тщательно взвесить свои желания и возможности, в данном случае это даже более важно, чем при модификации системы.

Стороны, участвующие в первичной оценке SDLC, должны также изучить возможность принятия альтернативных решений. Если принято решение о необходимости замены информационной системы, то возникает следующий вопрос: "Выполнимо ли это?" На этапе анализа экономической целесообразности необходимо решить следующие проблемы.

- ❑ *Технические аспекты требований к оборудованию и программному обеспечению.* Принятые решения не обязательно должны как-то зависеть от поставщиков, но они должны учитывать характер оборудования (персональный компьютер, мини-компьютер или универсальная машина) и требования к программному обеспечению (однопользовательская или многопользовательская операционная система, тип базы данных, языки программирования, используемые в приложениях, и т. д.).
- ❑ *Стоимость системы.* Известно, что самый простой вопрос "Можем ли мы позволить себе это?" является самым важным. (А ответ на этот вопрос может стать поводом для повторной, более тщательной первоначальной оценки!) Стоит повториться, что глупо решать копейчную проблему за миллион долларов.

6.3.2. Анализ

Проблемы, поставленные на этапе планирования, более детально изучаются на этапе анализа. Необходимо выполнить макроанализ как индивидуальных потребностей, так и потребностей компании в целом, и ответить на следующие вопросы.

- ❑ Каковы конкретные требования конечных пользователей системы?
- ❑ Вписываются ли эти требования в общий набор технических требований к информационной системе?

Фактически стадия анализа SDLC состоит во всесторонней ревизии требований пользователя.

На этапе анализа также изучаются имеющиеся оборудование и программные средства. Результатом анализа должно стать лучшее понимание функционального назначения системы, существующие и потенциальные проблемы, а также сфера ее действия.

Для выработки плана действий и выявления проблемных областей конечные пользователи и системотехники должны работать сообща. Такая кооперация жизненно необходима для определения соответствующих рабочих характеристик, по которым можно оценить новую систему.

Наряду с изучением требований пользователя и имеющихся систем на этапе анализа необходимо создать логический проект системы. С помощью логического проектирования нужно определить необходимую концептуальную модель данных, входные данные, процессы и предполагаемые требования к выходным данным.

При создании логического проекта проектировщик должен использовать такие инструментальные средства, как схемы информационных потоков (data flow diagram, DFD), диаграммы "вход-процесс-выход" (HIPO, hierarchical input process output) или ER-диаграммы. На этом этапе выполняется моделирование данных проекта БД, выявляются и описываются объекты и их атрибуты, а также связи сущностей внутри базы данных.

Логический проект обеспечивает функциональное описание системных компонентов (модулей) для каждого процесса внутри базы данных. Все преобразования данных (процессы) описываются и документируются с помощью таких инструментальных средств анализа, как схемы информационных потоков (DFD). Так проверяется концептуальная модель данных.

6.3.3. Детальное проектирование систем

На этапе детального проектирования систем проектировщик завершает проектирование системных процессов. Сюда включаются все необходимые технические описания экранов, меню, отчетов и других элементов, используемых для повышения эффективности извлечения необходимой информации. Эти шаги предпринимаются для перехода от старой системы к новой. Необходимо также запланировать обучение основным принципам и методам и представить планы обучения на рассмотрение руководства.

Примечание

Поскольку основное внимание мы уделяем деталям процесса проектирования системы, до сих пор не упоминалось о необходимости утверждения всех стадий процесса разработки системы руководством. Такое утверждение необходимо, поскольку решение о продолжении работ требует определенного финансирования. На пути разработки любой системы есть множество моментов, когда приходится принимать решение о продолжении или приостановке работ в выбранном направлении.

6.3.4. Реализация

На этапе реализации устанавливаются оборудование, программное обеспечение СУБД и прикладные программы и реализуется проект БД. На начальной стадии этапа реализации система вступает в цикл кодирования, тестирования и отладки до тех пор, пока она не будет полностью готова к работе. На этом этапе создается действующая база данных, система настраивается путем создания таблиц и представлений, авторизации пользователей и т. д.

Содержимое базы данных может загружаться интерактивно или в пакетном режиме с помощью различных методов и устройств:

- ☐ настраиваемых пользовательских программ;
- ☐ интерфейсных программ базы данных;
- ☐ программ конвертирования, способных импортировать данные из различных файловых структур с помощью пакетных процедур и утилит БД.

До момента ввода в эксплуатацию система должна подвергаться всестороннему тестированию. Как правило, реализация и тестирование новой системы занимают от 50 до 60% общего времени разработки. Однако появление весьма изощренных генераторов приложений и инструментов отладки в значительной степени уменьшило затраты времени на кодирование приложений и их отладку.

После завершения тестирования проверяется и распечатывается окончательный вариант документации и организуется обучение конечных пользователей. В конце этого этапа система полностью введена в строй, но, тем не менее, требует постоянной оценки и настройки.

6.3.5. Сопровождение

Практически сразу же после ввода системы в строй конечные пользователи начинают просить внести в нее изменения. Внесение изменений и исправлений выполняется службой сопровождения системы, работающей в трех направлениях¹.

- ☐ *Корректирующее обслуживание* — как ответ на возникающие ошибки системы.
- ☐ *Адаптивное обслуживание* — как ответ на изменение корпоративной среды.
- ☐ *Усовершенствование* — расширение возможностей системы.

Поскольку любое требование структурных изменений предполагает повторное выполнение каких-либо этапов SDLC, информационная система в известном смысле всегда находится на некотором этапе SDLC!

Каждая система имеет предопределенную эксплуатационную долговечность. Фактическая долговечность системы зависит от ее практической пользы. Есть несколько причин для уменьшения эксплуатационной долговечности некоторых систем, одна из них — быстрые изменения в технологии; это особенно касается систем, в основу которых положены скорость обработки и возможности расширения. Другая распространенная причина — стоимость обслуживания системы.

Если стоимость сопровождения системы слишком высока, то ее польза вызывает сомнение. Технология *автоматизированного проектирования систем* (CASE, computer-assisted systems engineering) — например System Architect или Visio — помогает создавать высококачественные системы в приемлемые сроки и по разумной цене. Кроме того, структурированные, хорошо документированные и особенно стандартизированные реализации систем, созданных с помощью CASE-технологий,

¹ См. "Software Maintenance Revisited: A Product Life Cycle Perspective", E. Reed Doke and Nell E. Swanson, Information Executive 4(1). Winter 1991, стр. 8—11. Дата выпуска этой статьи может вызвать сомнение в ее актуальности, но она все же остается значимой и сегодня. Хотя среда программирования меняется с головокружительной быстротой, многие основополагающие принципы разработки, реализации и управления программным обеспечением обладают поразительной долговечностью.

продлевают срок эксплуатации систем, делая их проще и дешевле в обслуживании и при обновлении².

6.4. Жизненный цикл базы данных (DBLC)

Функционируя внутри сложной информационной системы, база данных обладает собственным жизненным циклом. *Жизненный цикл базы данных* (Database Life Cycle, DBLC) состоит из шести этапов (рис. 6.3): этап начальной разработки базы данных, проектирование БД, реализация и загрузка, тестирование и оценка, функционирование, и, наконец, сопровождение и развитие.

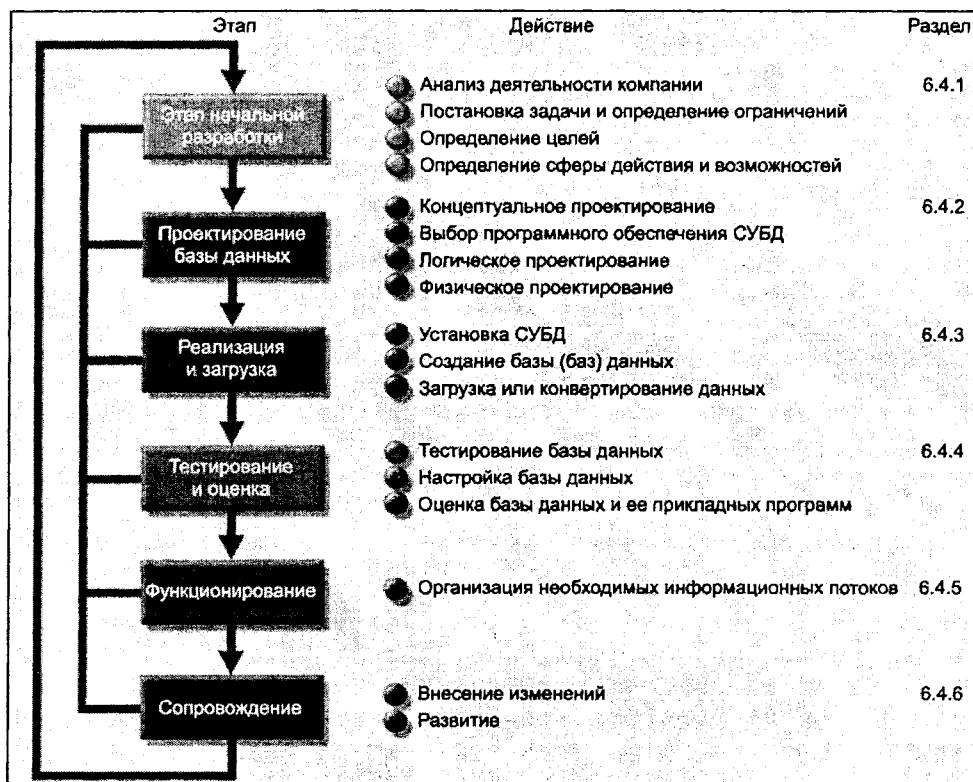


Рис. 6.3. Жизненный цикл базы данных (DBLC)

² Например, если вы используете Visio, то наверняка знаете, что в Visio проверяется внутренняя непротиворечивость ER-диаграмм при обновлении внешних ключей. Visio реализует внешние ключи в соответствии с типами сущностей проекта (слабые или сильные) и природой связи (идентифицируемая или неидентифицируемая) между этими сущностями. Если вы посмотрите на результаты реализации, то сразу увидите, верны ли были ваши действия. Кроме того, если в проекте имеются циклические аргументы, то Visio выведет соответствующее предупреждение. Поэтому вы сможете определить проблемы проекта прежде, чем они войдут в реализацию!

6.4.1. Этап начальной разработки

Если в компанию приглашен проектировщик БД, значит, имеющаяся система не в состоянии выполнять жизненно важные функции (вы же не будете вызывать водопроводчика, если краны в порядке!). Поэтому кроме проверки работоспособности имеющейся системы проектировщик должен определить, вследствие чего система стала работать неудовлетворительно. При этом потребуются переговоры (в основном, для того, чтобы выслушать!) с множеством конечных пользователей. Хотя проектирование баз данных является сугубо техническим бизнесом, оно тоже предназначено для людей. Проектировщики баз данных должны быть очень коммуникабельны.

В зависимости от сложности и масштабов структуры БД проектировщик может быть один или входить в группу системных разработчиков, в которой есть руководитель группы, один или более старших системных аналитиков и один или более младших системных аналитиков. Далее мы будем использовать термин "проектировщик" для обозначения всего спектра специальностей, входящих в группу проекта.

Основные цели на этапе начальной разработки базы данных:

- ☐ анализ деятельности компании;
- ☐ постановка задачи и определение ограничений;
- ☐ определение целей;
- ☐ определение сферы действия и границ возможностей.

На рис. 6.4 представлены интерактивные и итеративные процессы, необходимые для успешного выполнения первого этапа DBLC. Обратите внимание, глядя на рис. 6.4, что этап начальной разработки базы данных приводит к определению целей системы базы данных. Используя рис. 6.4 в качестве образца, исследуем каждый из этих компонентов более детально.

Анализ деятельности компании

Термин "анализ" (от греч. *analysis* — разложение) означает "разделение целого на его составляющие с целью изучения их свойств, функций и т. д.". Понятие "деятельность компании" описывает общие условия, в которых функционирует компания, ее организационную структуру и ее предназначение. Для анализа деятельности компании проектировщик базы данных должен узнать, из каких рабочих органов состоит компания, как они функционируют и как взаимодействуют друг с другом.

На этом этапе необходимо ответить на следующие вопросы.

- ☐ *Что представляет собой рабочая среда компании, каково ее предназначение в рамках этой среды?* Проект должен отвечать требованиям, соответствующим предназначению компании. Например, торгово-посылочная фирма, скорее всего, предъявляет несколько иные требования к базе данных, чем компания обрабатывающей промышленности.
- ☐ *Какова организационная структура компании?* При определении информационных потоков, отчетов и форматов запроса и т. д. необходимо знать, кто управляет компанией, кто и какие выполняет отчеты и для кого.

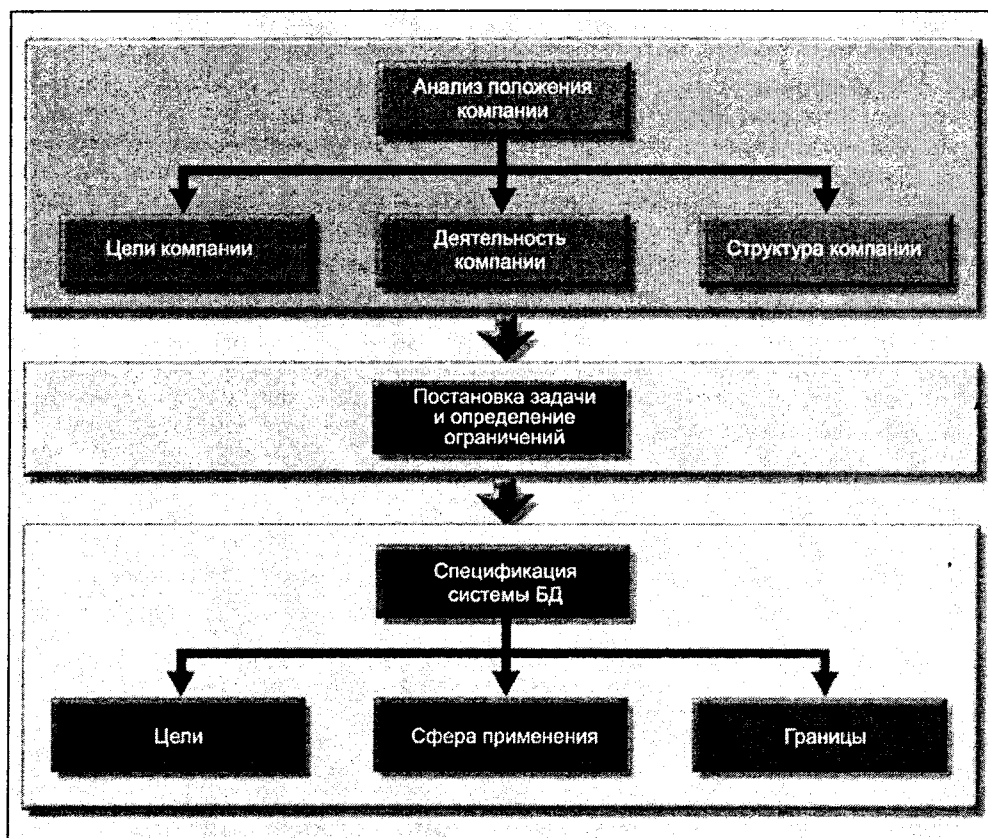


Рис. 6.4. Этап начальной разработки базы данных

Постановка задачи и определение ограничений

Проектировщик имеет как формальные, так и неформальные источники информации. Если компания существует в течение какого-то промежутка времени, она уже имеет какую-то информационную систему (ручную или автоматизированную). Как функционирует имеющаяся система? Какая требуется исходная информация для этой системы? Какие документы создаются системой? Как используется выходная информация системы? Кем? Изучение документации, связанной с деятельностью компании, может дать много полезных сведений. Кроме официальной стороны деятельности предприятия есть ее неформальная сторона; проектировщик должен постараться узнать, в чем их сходство и отличие.

Процесс постановки задачи может вначале показаться несколько хаотичным. Конечные пользователи компании зачастую не способны точно описать сферу интересов компании или указать на реальные проблемы, с которыми сталкивается компания в своей деятельности. Чаше всего представление руководства о деятельности

компании отличается от представления конечных пользователей, выполняющих реальную повседневную работу.

В начальной стадии проектировщик, скорее всего, составит очень длинный список различных проблем. Вот, к примеру, слова президента быстро растущей производственной компании:

"Хотя мы и удовлетворены быстрым ростом, однако руководство обеспокоено тем, что этот рост начинает сказываться на уровне обслуживания клиентов и, хуже того, быстрый рост может привести к ослаблению контроля качества производства".

В процессе постановки задачи быстро набирается целая куча описаний самых насущных проблем. Например, руководитель маркетингового отдела жалуется:

"Мне приходится работать с устаревшей системой файлов. Мы производим более 1700 деталей для станков. Когда к нам обращаются постоянные клиенты, мы не можем быстро посмотреть, что имеется у нас в наличии. Если к нам обращается новый клиент, мы не можем найти нужную деталь по описанию, поэтому нам иногда приходится перебирать весь станок для поиска детали, имеющейся в описи. Это расточительство! Кроме того, многих клиентов раздражает, что мы не можем быстро ответить".

Руководитель производства высказывает свои претензии:

"Мне требуется, в лучшем случае, несколько часов, чтобы подготовить отчеты, необходимые для составления графика работ. Я не могу тратить столько времени на его оперативную корректировку. Трудно управлять тем, о чем не имеешь сведений.

Я не могу быстро реагировать на обстановку. Взять, к примеру, сборку станка. Сейчас механики простаивают в ожидании необходимого комплекта деталей, а я не могу направить их на работы, которые должны выполнять менее квалифицированные работники. При существующей системе планирования у нас слишком много простоев. Я трачу массу времени и постоянно выбиваюсь из графика. Наши сверхурочные доходят до смешного.

Порой мы выпускаем детали, которые уже есть на складе, поскольку мы, кажется, не можем привести в соответствие то, что мы имеем по факту, с тем, что запланировано. В отделе доставки на меня кричат, потому что я отказываюсь выпускать детали, которые у них лежат ярусом ниже. Иногда это стоит нам кучу баксов.

Новые отчеты попадают ко мне в офис спустя дни, даже недели. А мне нужна эта тонна отчетов для управления персоналом, простоями, обучением и т. д. Я не получаю оперативную информацию. Что мне нужно — это возможность быстро получать свежие сведения по проценту брака, проценту переделок и эффективности обучения, мало ли еще что. И такие отчеты мне нужны по сменам, по датам, — по любой характеристике, какая мне вздумается, — чтобы работать по графику, планировать обучение, да все, что угодно".

У механика свои проблемы:

"Бумаги меня тормозят. Мой график срывается из-за того, что Джон не получил документацию вовремя. Я без конца ищущу спецификации, данные по пуску и установке, другие бумаги, теряю два или три часа на все это. Теперь вы понимаете, почему я выбиваюсь из графика. Я пытаюсь нормально работать, но трачу слишком много времени на подготовку к работе".

После сбора такой "живой" информации проектировщик баз данных должен продолжить исследование, чтобы получить дополнительные сведения, помогающие выявить проблемы на более высоком уровне деятельности компании. Как частная проблема пользователя — руководителя отдела маркетинга — вписывается в общую сферу деятельности отдела маркетинга? Как решение проблемы этого пользователя поможет решить задачи отдела маркетинга и задачи компании в целом? Какое отношение имеет отдел маркетинга к другим подразделениям? Последний вопрос особенно важен. Обратите внимание, что в проблемах, которые выдвигают отдел маркетинга и производственный отдел, есть общие элементы. Если можно упорядочить учет материально-технических ценностей, то, по крайней мере, некоторые из проблем обоих подразделений будут разрешены.

Поиск правильных ответов очень важен, особенно в отношении рабочих взаимосвязей между подразделениями компании. Если предложенная система решает проблемы отдела маркетинга, но усложняет жизнь производственного отдела, то какой в ней толк. Допустим, вы обнаружили, что платите за воду слишком много. Причина: течет кран. Ваше решение? Взять да и перекрыть стояк, отключив воду всему дому. Адекватно ли это решение? Может, лучше все же починить кран? Пример может показаться слишком упрощенным, и все же практически любой опытный проектировщик баз данных может столкнуться с подобным вариантом (может быть, более сложным и не столь очевидным) решения проблем базы данных.

Даже самое полное и точное описание проблем не всегда приводит к хорошим решениям. Действительность устанавливает свои правила, ограничивающие даже самый изысканный проект определенными рамками. К таким ограничениям относятся время, бюджет, кадры и многое другое. Если вам в течение месяца требуется найти решение стоимостью не выше \$12 000, то, очевидно, что решение, для которого потребуются два года и \$100 000, здесь не годится. Проектировщик должен точно знать разницу между желаниями и возможностями.

Определение целей

Предложенная система базы данных проектируется для того, чтобы помочь решить, по крайней мере, основные проблемы, выявленные на этапе изучения деятельности компании. По мере составления списка, скорее всего, будут обнаружены некоторые общие источники проблем. Например, и руководитель отдела маркетинга, и руководитель производственного отдела в нашем примере жалуются на неэффективный учет материально-технических средств. Если проектировщик может создать базу данных, которая может стать основой для эффективного учета деталей, то в выигрыше будут оба подразделения. Первоначальной целью, следовательно, может стать создание системы эффективного учета материально-технических ценностей и администрирования.

Примечание

При разработке проектировщик должен найти *источник* проблем. Имеется много (слишком много) примеров баз данных, которые совершенно не удовлетворяют конечных пользователей, поскольку они были спроектированы так, что устраняли лишь *симптомы* проблем, а не их *причину*.

Обратите внимание, что на начальной стадии также предлагается решение проблем. Работа проектировщика состоит в том, чтобы обеспечить соответствие своего видения целей базы данных с представлением конечных пользователей. В любом случае проектировщик базы данных должен ответить на следующие вопросы.

- ☐ Что является изначальной целью предлагаемой системы?
- ☐ Будет ли система взаимодействовать с другими существующими или планируемыми системами компании?
- ☐ Позволяет ли система совместно использовать данные с другими системами или пользователями?

Определение сферы действия и границ возможностей

Проектировщик должен знать о существовании двух ограничений: сфера действия и границы возможностей. Сфера действия системы определяет рамки проекта в соответствии с эксплуатационными требованиями. Будет ли база данных охватывать все предприятие, одно или несколько его подразделений или только одну или несколько функций одного подразделения? Проектировщик должен знать "размер бейсбольной площадки". Знание области действия базы данных поможет определить необходимые структуры данных, тип и количество сущностей, физический размер БД и т. д.

Предлагаемая система должна разрабатываться в определенных *границах*, выход за которые недопустим. Какой проектировщик может сказать: "Наше время неограниченно. Можно тратить любые деньги и нанимать сколько угодно специалистов для выполнения проекта"? Границы возможностей также определяются имеющимся оборудованием и программным обеспечением. В идеальном случае проектировщик может выбирать оборудование и программное обеспечение, которое лучше всего согласуется с целями проектирования. На самом деле, выбор программного обеспечения является важнейшей стороной жизненного цикла разработки систем. К сожалению, в действительности систему часто приходится проектировать на уже имеющемся оборудовании. Сфера действия и границы возможностей, таким образом, становятся факторами, которые втискивают проект в определенный шаблон, и искусство проектировщика заключается в том, чтобы создать лучший проект системы в рамках этих ограничений. (Обратите внимание, что иногда приходится менять постановку задачи и намеченные цели, чтобы привести их в соответствие со сферой действия и границами возможностей.)

6.4.2. Проектирование базы данных

На втором этапе создается модель базы данных, в которой должны быть учтены сфера деятельности компании и ее задачи. Мы приступаем, наверное, к самому главному этапу DBLC: обеспечение соответствия конечного продукта системным требованиям и запросам пользователей. В процессе проектирования базы данных мы должны уделить основное внимание свойствам данных, необходимых для построения модели данных. Говоря кратко, у нас есть два представления о данных в системе: представление о данных со стороны компании, которая является источником получения необходимых сведений, и представление проектировщика о структурах данных, доступе к ним, а также о действиях, которые необходимо выполнить для структурирования данных. На

рис. 6.5 сопоставляются эти два представления. Обратите внимание, что различие в представлениях в итоге выражается двумя терминами *что* и *как*. Определение данных является составной частью второго этапа DBLC.

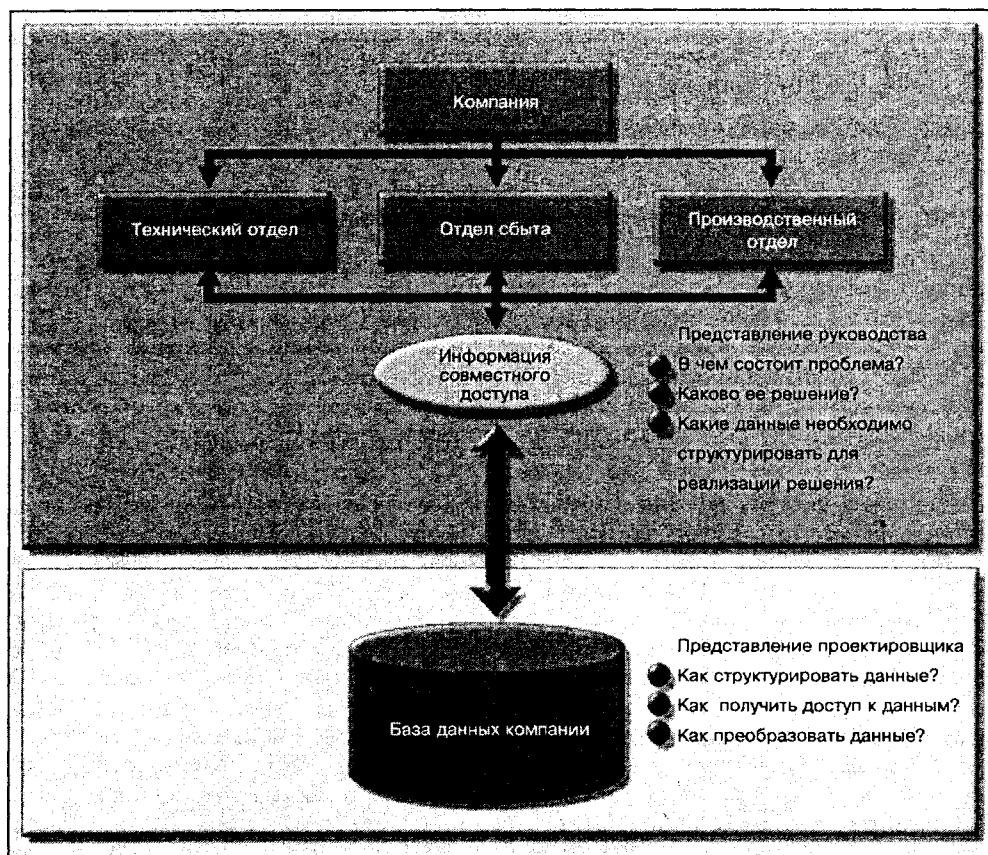


Рис. 6.5. Два представления о данных: руководитель компании и проектировщик

Изучая действия, которые необходимо выполнить на этапе проектирования DBLC, надо иметь в виду следующее.

- ❑ Процесс проектирования базы данных лишь отдаленно напоминает анализ и проектирование больших систем. Информационный компонент — лишь одна из составляющих большой информационной системы.
- ❑ За проектирование остальных компонентов системы отвечают системные аналитики и/или системные программисты. Они создают процедуры, помогающие структурировать информацию внутри базы данных и извлекать из БД нужные сведения.

- Проектирование базы данных не является последовательным процессом. Скорее это итеративный процесс, обеспечивающий постоянную обратную связь для повторной проверки предыдущих этапов.

На втором этапе DBLC (проектирование БД) мы можем проследить процесс проектирования базы данных, представленный на рис. 6.6. Взгляните на последовательность действий, представленных на этом рисунке. В гл. 7 и 8 мы подробно изучим, что происходит на каждой из этих стадий при разработке реальной базы данных.

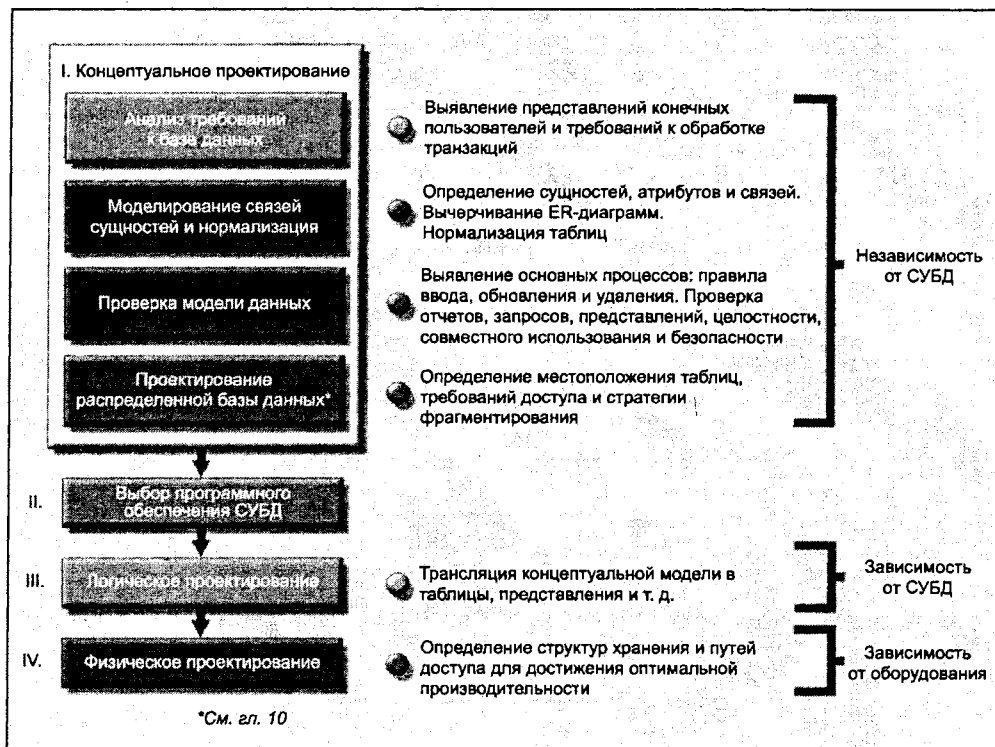


Рис. 6.6. Последовательность действий при проектировании баз данных

Теперь мы можем подробно обсудить каждый компонент, представленный на рис. 6.6. Детальное изучение этих процессов поможет вам проектировать и внедрять эффективные базы данных в реальных условиях.

I. Концептуальное проектирование

На этапе концептуального проектирования для создания абстрактной структуры базы данных применяется моделирование данных, в ходе которого объекты реального мира изображаются по возможности приближенными к жизни. Концептуальная модель должна выражать ясное понимание сферы деятельности предприятия и выполняемых им функций. На этом уровне абстракции тип оборудования и/или используемая мо-

дель базы данных, как правило, не определяются. Поэтому концептуальный проект не должен зависеть от оборудования и программного обеспечения, чтобы систему можно было установить на любых оборудовании и программной платформе, которые будут определены позже.

Помните следующее правило необходимого минимума информации:

Все, что необходимо, здесь есть и все, что здесь есть, необходимо.

Другими словами, убедитесь, что в вашей модели имеются *все* нужные данные, и что все данные, имеющиеся в модели, действительно *нужны*. В модели должны быть определены все элементы данных, необходимые для транзакций базы данных, и каждый элемент данных, определенный в модели, должен использоваться, по крайней мере, в одной транзакции.

Однако применяя правило минимума информации, постарайтесь не скатиться в сторону чрезмерного упрощения. Старайтесь думать не только о сиюминутных потребностях предприятия, но и возможных будущих операциях. То есть в проекте БД надо оставить место для последующих модификаций и добавлений, что гарантирует дальнейшие инвестиции в информационные ресурсы.

Из рис. 6.6 следует, что концептуальное проектирование состоит из четырех шагов.

1. Анализ требований к базе данных.
2. ER-моделирование и нормализация.
3. Проверка модели данных.
4. Проектирование распределенной базы данных.

Рассмотрим каждый из этих шагов в порядке очередности.

Анализ данных и требования

Первым шагом концептуального проектирования является определение свойств элементов данных. Удачно спроектированная база данных представляет собой фабрику информации, производящую ключевые ингредиенты для успешного принятия решений. Именно элементы данных с их свойствами впоследствии могут быть структурированы должным образом и использованы для принятия важных решений. Поэтому усилия проектировщика должны быть сосредоточены на следующем.

- ☐ **Информационные потребности.** Какого сорта информация необходима? То есть какую выходную информацию должна вырабатывать система (отчеты, запросы)? Какую информацию вырабатывает имеющаяся система, и до какой степени ее можно считать адекватной?
- ☐ **Пользователи информации.** Кто будет пользоваться структурированными данными? Каким образом будет использоваться информация? В чем различаются представления конечных пользователей о данных компании?
- ☐ **Источники информации.** Где можно получить информацию? Как можно извлечь информацию из источника?
- ☐ **Состав информации.** Какие элементы данных необходимы для структурирования информации? Каковы атрибуты данных? Как данные связаны друг с другом? Каков объем данных? Как часто используются данные? Какие преобразования данных необходимо выполнять?

Чтобы собрать необходимый материал, проектировщик получает ответы на эти вопросы из различных источников.

- *Выясняя и суммируя представления о данных конечных пользователей.* Проектировщик БД и конечные пользователи тесно взаимодействуют друг с другом, чтобы получить точное описание представления о данных конечных пользователей. В свою очередь, представления конечных пользователей будут использованы для определения основных элементов базы данных.
- *Непосредственно наблюдая имеющуюся систему: существующие выходные данные и желаемые.* Обычно конечные пользователи на рабочих местах пользуются какой-то системой (ручной или автоматизированной). Проектировщик должен исследовать имеющуюся систему для определения данных и их свойств. Проектировщик проверяет входные формы и файлы (таблицы) для того, чтобы определить типы и объемы данных. Если на рабочем месте конечного пользователя уже есть автоматизированная система, то проектировщик должен тщательно исследовать имеющиеся и желаемые отчетные формы, чтобы описать данные, необходимые для получения отчетов.
- *Взаимодействуя с группой системного проектирования.* Как уже говорилось в этой главе, проектирование базы данных является составной частью жизненного цикла разработки систем (SDLC). В некоторых случаях системные аналитики, ответственные за проектирование новой информационной системы, также разрабатывают концептуальную модель данных. (Это справедливо, в частности, для сферы микрокомпьютерного бизнеса.) В других случаях проектирование базы данных рассматривается как часть работы администратора БД. Наличие администратора базы данных (database administrator, DBA) обычно предполагает формальное существование отдела обработки данных. DBA проектирует базу данных в соответствии со спецификациями, разработанными системным аналитиком.

Для того чтобы создать точную модель данных, проектировщик должен иметь полное представление о типах данных компании, объемах информации и способах ее использования. Но данные сами по себе не дают представления о деятельности предприятия. С точки зрения БД набор данных становится значимым только тогда, когда определены бизнес-правила.

Помните, что *бизнес-правила* это краткое и точное повествовательное описание политики, процедур и норм в условиях предприятия. Бизнес-правила, полученные из детального описания операций, помогают выработать и осуществить определенные мероприятия на данном предприятии. Если бизнес-правила заданы корректно, то в них определяются и сущности, и атрибуты, и связи, и связности, и мощность, и ограничения.

Примечание

В известном смысле название "бизнес-правила" не совсем точно: эти правила применимы к *любым* организациям — сферы бизнеса, государственным учреждениям, религиозным группам или исследовательским лабораториям — которые хранят и используют данные для получения необходимой информации. Организация может быть небольшой или крупной, это может быть даже отдельный человек, занимающийся какой-либо деятельностью, требующей хранения данных и управления ими; все они используют определенные бизнес-правила.

Бизнес-правила должны быть не только эффективными, но понятными, и к тому же они должны быть доведены до сведения всех, чтобы каждый человек на предприятии одинаково понимал и интерпретировал эти правила. Бизнес-правила описывают на простом языке основные и точные характеристики данных с точки зрения компании. Ниже приведены примеры типичных бизнес-правил.

- ☐ Клиент в одном счете может указать несколько покупок.
- ☐ Каждая оплата по счету предъявляется только одному клиенту.
- ☐ Механик не может работать более, чем 10 часов в сутки.
- ☐ Авиабилет сотруднику оплачивается только в том случае, если пункт назначения командировки находится не ближе 200 миль.
- ☐ Группа обучения должна состоять не менее, чем из 10 человек, но не более, чем из 30 человек.
- ☐ Самолет компании должен проходить проверку несущих конструкций через каждые 100 часов полета.
- ☐ Лабораторию нельзя использовать в демонстрационных целях более, чем один час в день.
- ☐ Клиент может инициировать несколько счетов.
- ☐ Каждый счет инициируется только одним клиентом.

Основными источниками бизнес-правил являются руководители компании, лица, определяющие политику компании, руководители подразделений и документация — различные методики, стандарты или руководства по эксплуатации. Самый быстрый и непосредственный источник для разработки бизнес-правил — прямые опросы конечных пользователей. К сожалению, поскольку восприятие разных людей сильно различается, конечного пользователя все же нельзя считать абсолютно надежным источником при определении бизнес-правил. Например, механик административно-хозяйственного отдела может полагать, что любой механик может выполнять работы по техническому обслуживанию, тогда как на самом деле такую работу может выполнять только механик, имеющий лицензию на проведение техосмотра. Такое отличие может показаться тривиальным, но оно может иметь важные правовые последствия. Хотя конечные пользователи вносят важный вклад в разработку бизнес-правил, при этом все же приходится делать поправку на возможную разницу восприятия. Мы часто сталкивались с тем, что люди, выполняющие одну и ту же работу, имели очень разное понятие о своих должностных обязанностях. Это может свидетельствовать об "управленческих проблемах", однако этот общий диагноз, тем не менее, никак не поможет проектировщику базы данных. Обнаружив такие проблемы, проектировщик должен согласовать все эти различия в описаниях и проверить результаты этого согласования с тем, чтобы гарантировать адекватность бизнес-правил.

Знание бизнес-правил помогает проектировщику понять, как функционирует предприятие в целом, и какую роль играют данные в деятельности компании. Следовательно, проектировщик должен определить бизнес-правила компании и проанализировать их влияние на природу, роль и сферу действия данных.

Правильное определение бизнес-правил дает некоторые важные преимущества при проектировании новых систем:

- помогает стандартизовать представление о данных компании;
- дает основу взаимодействия между пользователями и проектировщиками;
- позволяет проектировщику понять природу, роль и сферу действия данных;
- позволяет проектировщику глубже понять деятельность предприятия;
- позволяет проектировщику определить надлежащие правила связей и ограничений на внешний ключ (см. гл. 3).

Последний пункт заслуживает особого внимания: определение того, являются ли данные связи обязательными или необязательными, зависит от работоспособности бизнес-правил.

ER-моделирование и нормализация

Перед тем как приступить к созданию ER-модели, проектировщик должен сообщить всем и привести в исполнение стандарты на документацию проекта. Стандарты включают в себя использование диаграмм и символов, стиль написания документации, разметку документов и другие соглашения, которые необходимо соблюдать при документировании. Проектировщики часто не придают значения этому важному требованию, особенно когда они работают в группе. Пренебрежение стандартизацией документов в дальнейшем может помешать обмену информацией. А неудачи при обмене информацией часто приводят к неудачному проекту БД. И наоборот, строгое следование стандартам делает проект проще для понимания и является залогом (но не гарантией) беспрепятственного взаимодействия всех компонентов системы.

Поскольку бизнес-правила обычно определяют природу связи сущностей, проектировщик должен включить их в концептуальную модель. Процесс определения бизнес-правил и разработки концептуальной модели с помощью ER-диаграмм можно описать с помощью действий, представленных в табл. 6.2³.

Таблица 6.2. Разработка концептуальной модели с помощью ER-диаграмм

Шаг	Описание
1	Определение, анализ и уточнение бизнес-правил
2	Выявление основных сущностей на основе результатов Шага 1
3	Определение связей сущностей на основе результатов Шагов 1 и 2
4	Определение атрибутов, первичных ключей и внешних ключей для каждой сущности
5	Нормализация сущностей

³ См. "Linking Rules to Models", Alice Sandifer and Barbara von Halle, Database Programming & Design, 4(3), March 1991, стр.13—16. Хотя источник может показаться устаревшим, он, тем не менее, соответствует современным стандартам. Технологии изменились существенно, но сам процесс остался тем же.

Таблица 6.2 (окончание)

Шаг	Описание
6	Завершение первоначальной ER-диаграммы
7	Проверка конечными пользователями модели, полученной после Шага 6, на основе имеющихся данных и технических требований
8	Модификация ER-диаграммы на основе Шага 7

Некоторые шаги, перечисленные в табл. 6.2, выполняются параллельно. А некоторые, такие как процесс нормализации, могут потребовать создания дополнительных сущностей и/или атрибутов, что может стать причиной пересмотра ER-модели. Например, в процессе выявления двух основных сущностей проектировщик должен идентифицировать переходную промежуточную сущность, представляющую собой связь M:N между двумя основными сущностями.

Для проверки предположим, что создается концептуальная модель для корпорации по прокату видеокассет JollyGood, конечные пользователи которой хотят отслеживать действия клиентов, берущих напрокат кассеты. На простейшей ER-диаграмме, представленной на рис. 6.7, показана составная сущность, которая помогает отслеживать клиентов и взятые ими напрокат видеокассеты. Бизнес-правила определяют необязательную природу связей между сущностями VIDEO (кассета) и CUSTOMER (клиент), представленными на рис. 6.7. (Например, клиенты не требуют проверки видеокассеты. Не нужно проверять наличие видеокассет на полке. Клиент может взять напрокат много видеокассет, а видеокассету могут брать напрокат многие клиенты.) Обратите особое внимание на составную сущность RENTAL (прокат), связывающую две основные сущности.

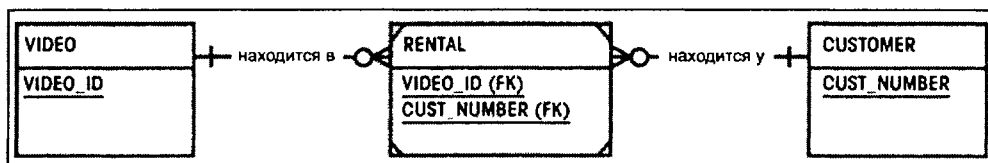


Рис. 6.7. Составная сущность

Вы, вероятно, обратили внимание, что начальная ER-модель может несколько раз подвергаться пересмотру, прежде чем она будет соответствовать требованиям системы. Вы должны помнить, что ER-модель является средством коммуникации, а также схемой проекта базы данных. Поэтому начальная ER-диаграмма должна отвечать на такие вопросы, как "Это действительно то, что вам нужно?" в процессе согласования системы с конечными пользователями. Например, ER-диаграмма, представленная на рис. 6.7, еще далека от завершения. Очевидно, что нужно определить еще много атрибутов и проверить множество зависимостей, перед тем как реализовать проект. Кроме того, этот проект не может обеспечить типичные транзакции проката видеокассет. Например, у каждой видеокассеты, скорее всего, есть несколько копий, доступных в прокате. Однако если сущность VIDEO, представленную на рис. 6.7,

использовать для хранения основных кассет, а также их копий, то в проекте будет иметь место избыточность данных, что продемонстрировано в табл. 6.3.

Таблица 6.3. Избыточность данных в таблице VIDEO

VIDEO_ID	VIDEO_TITLE	VIDEO_COPY	VIDEO_CHG	VIDEO_DAYS
SF-12345FT-1	Приключения на третьей планете	1	2,45	1
SF-12345FT-2	Приключения на третьей планете	2	2,45	1
SF-12345FT-3	Приключения на третьей планете	3	2,45	1
WE-5432GR-1	Кану и Тайлер 2: Путешествие	1	1,99	2
WE-5432GR-2	Кану и Тайлер 2: Путешествие	2	1,99	2

Начальную ER-диаграмму, представленную на рис. 6.7, нужно модифицировать так, чтобы она отвечала на вопрос: "Доступно ли более одной копии для данного фильма?" Необходимо обслуживать платежные операции. (У вас будет возможность модифицировать этот начальный проект в задаче 5 в конце этой главы.)

У вас не должно создаться впечатление, что ER-моделирование (определение сущностей и атрибутов, нормализация, проверка) должно выполняться в строгой последовательности. На самом деле, вполне вероятно, что вы вернетесь к ER-модели после ее завершения и будете повторять все действия до тех пор, пока не убедитесь, что ER-модель в точности соответствует проекту базы данных, способному удовлетворить требования проекта системы в целом. (Действия зачастую выполняются параллельно, и весь процесс носит итеративный характер.) На рис. 6.8 представлен весь этот процесс в целом. На рис. 6.9 сведены воедино все средства проектирования и источники информации, которые проектировщик может использовать при разработке концептуальной модели.

Все объекты (сущности, атрибуты, связи, представления и т. д.) определены в словаре данных, который совместно с процессом нормализации используется для устранения аномалий данных и проблем избыточности. В процессе ER-моделирования проектировщик должен:

- ☐ определить сущности, атрибуты, первичные ключи и внешние ключи (внешние ключи служат основой для связей сущностей);
- ☐ принять решение о добавлении новых первичных ключей для того, чтобы выполнить технологические требования и удовлетворить запросы конечных пользователей;
- ☐ принять решение по обработке многозначных атрибутов;
- ☐ принять решение о размещении внешних ключей в связях типа 1:1;
- ☐ избегать необязательных тернарных связей;

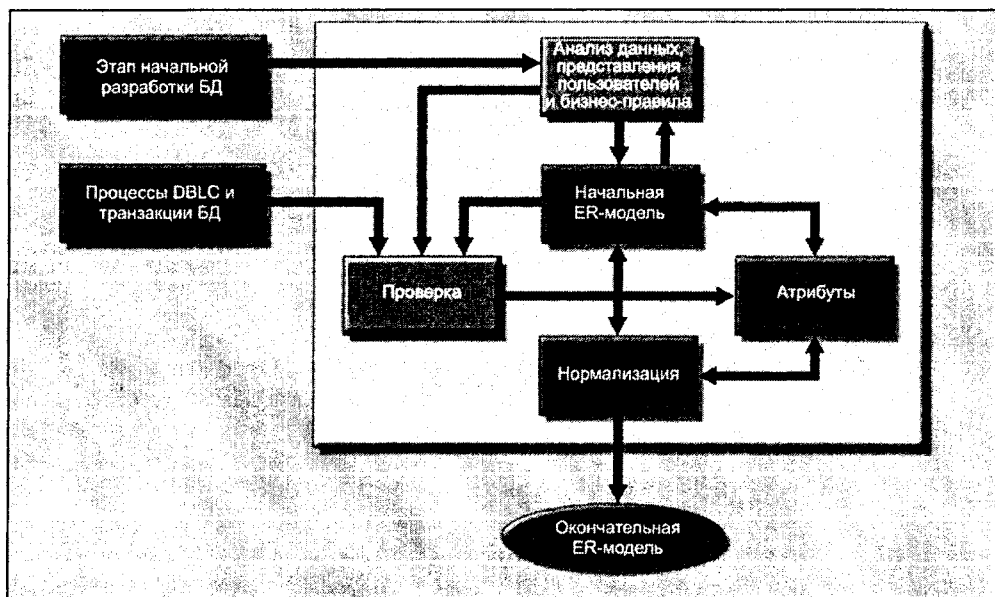


Рис. 6.8. Итеративный процесс ER-моделирования, основанный на множестве операций

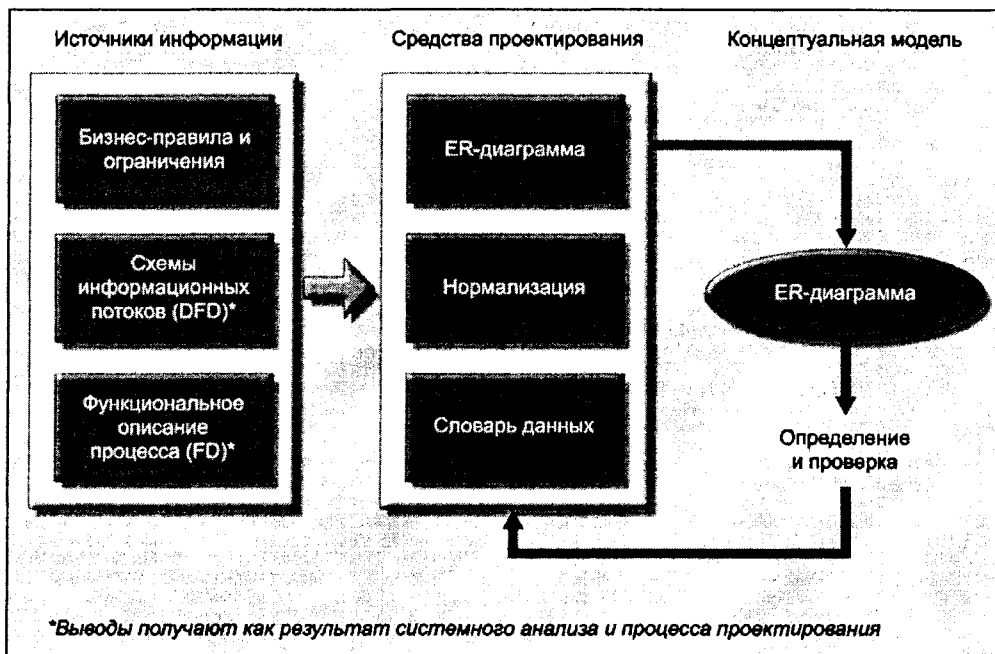


Рис. 6.9. Инструменты концептуального проектирования и источники информации

- нарисовать соответствующую ER-диаграмму;
- нормализовать модель данных;
- включить все определения элементов данных в словарь данных;
- принять решение по стандарту на соглашение об именовании.

Соглашение об именовании — очень важный момент, хотя проектировщики часто избегают его на свой страх и риск. Реальное проектирование базы данных, как правило, выполняется в группе. Поэтому очень важно обеспечить работу членов группы в среде, где определены и выполняются стандарты на соглашения об именовании. Для выполнения проекта очень важна добротная оформленная документация. Поэтому имеет смысл использовать процедуры, способные к самодокументированию.

Хотя мы установили некоторые полезные элементы и атрибуты соглашения об именовании в *гл. 3*, мы все же подробно рассмотрим их еще раз чуть позже. Однако надо иметь в виду, что такие соглашения иногда ограничиваются программным обеспечением СУБД. На СУБД действует правило общего знаменателя в масштабе предприятия. Например, в Microsoft Access название атрибута `LINE_ITEM_NUMBER` считается вполне допустимым. Многие старые СУБД, тем не менее, скорее всего, будут обрезать такие длинные имена при импортировании из одной СУБД в другую, тем самым затрудняя документирование. Поэтому при экспортировании таблиц могут быть установлены ограничения на длину имен. (То же справедливо и для типов данных. Например, многие старые СУБД не поддерживают форматы OLE и memo.)

Хотя мы пока не знаем, какую СУБД будем использовать, все же нужно принять соглашение об именовании, которое подходило бы для возможно более широкого спектра СУБД. Это соглашение об именовании также должно удовлетворять требованиям самодокументирования, насколько это возможно. Поскольку старые СУБД постепенно сходят со сцены, ограничения на соглашения об именовании будут постепенно сниматься. Поэтому будем придерживаться следующих соглашений.

- Нужно использовать наглядные названия для сущностей и атрибутов, когда это возможно. Например, в базе данных университетской компьютерной лаборатории (*см. гл. 7*) сущность `USER` (пользователь) содержит данные о пользователях лаборатории, а сущность `LOCATION` (местоположение) относится к размещению элементов (`ITEM`), которые хочет отслеживать директор лаборатории.
- Составным сущностям обычно присваиваются имена, связанные с атрибутами, которые эти сущности представляют. Например, в базе данных университетской компьютерной лаборатории (*см. гл. 7*) элемент (`ITEM`) может храниться во многих местах (`LOCATION`), а в данном месте (`LOCATION`) может храниться несколько элементов (`ITEM`). Поэтому составная (переходная, промежуточная) сущность, соединяющая `ITEM` и `LOCATION`, называется `STORAGE` (хранилище). Иногда проектировщик считает необходимым показать, какие сущности связаны составной сущностью. В таком случае имя составной сущности может включать фрагменты имен этих сущностей. Например, `STU_CLASS` может быть именем составной сущности, связывающей сущности `STUDENT` и `CLASS`. Однако такой вариант именования может вызвать некоторую путаницу, поэтому его надо применять осторожно. (Мы предпочли промежуточную сущность, связывающую сущности `CLASS` и `STUDENT`, назвать `ENROLL`.)

- Имена атрибутов должны быть наглядными и иметь префикс, указывающий на таблицу, в которой его можно найти. Мы будем использовать префиксы не длиннее пяти символов. Например, таблица `VENDOR` может содержать такие атрибуты, как `VEND_ID` и `VEND_PHONE`. Точно так же таблица `ITEM` может содержать атрибуты с такими именами, как `ITEM_ID` и `ITEM_DESCRIPTION`. Преимущество такого соглашения об именовании в том, что при этом сразу же виден внешний ключ таблицы. Например, если таблица `EMPLOYEE` содержит такие атрибуты, как `EMP_ID`, `EMP_LNAME` и `DEPT_CODE`, то сразу ясно, что `DEPT_CODE` является внешним ключом, связывающим таблицы `EMPLOYEE` и `DEPARTMENT`. Естественно, из-за наличия имен связей и таблиц, начинающихся с одних и тех же символов, приходится иногда немного подправлять соглашение об именовании, как это показано в следующем пункте.
- Если одна из таблиц названа `ORDER` (заказ), а связанная с ней слабая сущность — `ORDER_ITEM`, то для указания на атрибут, находящийся в таблице `ORDER`, будем использовать префикс `ORD`. Префикс `ITEM` мы будем использовать для обозначения атрибутов, расположенных в таблице `ITEM`. Очевидно, что в этом случае для указания на атрибуты таблицы `ORDER_ITEM` префикс `ORD` не подходит, поэтому в качестве префикса имен атрибутов таблицы `ORDER_ITEM` мы будем использовать комбинацию символов, например, `OI`. Несмотря на все эти проблемы, заметим, что всегда можно найти подходящий способ назначения префиксов, указывающих на происхождение атрибута. (Помните, что некоторые РСУБД используют список "зарезервированных слов". Например, `ORDER` может интерпретироваться как зарезервированное слово оператора `SELECT`. В этом случае нужно использовать для таблицы другое имя.)

Надо сказать, что соглашение об именовании не всегда можно соблюсти в точности. Иногда вследствие ограничения на длину приходится использовать не очень наглядные имена атрибутов. К тому же, учитывая многообразие объектов и атрибутов в сложном проекте, вероятно, при назначении подходящих префиксов имен атрибутов придется проявить большую изобретательность. И такие префиксы, конечно, не очень помогут точно определить местонахождение атрибута. Тем не менее, последовательная политика использования префиксов позволит значительно уменьшить двусмысленность. Например, хотя и не совсем очевидно, что префикс `CO` связан с таблицей `CHECK_OUT`, но уж совершенно точно можно сказать, что он не связан с таблицами `WITHDRAW`, `ITEM` или `USER`.

Старайтесь придерживаться представленного здесь соглашения об именовании. На самом деле, большинство приходит к следующей мысли: "Я не понимаю, зачем было так спорить по поводу соглашения об именовании, — теперь, когда я применяю его на практике, его полезность не вызывает у меня сомнений".

Проверка модели данных

ER-модель нужно проверить на соответствие предлагаемым системным процессам, чтобы убедиться, что модель базы данных поддерживает все процессы. При проверке требуется, чтобы модель прошла серии тестов:

- в отношении к представлениям конечных пользователей и выполняемым ими транзакциям: операции `SELECT`, `INSERT`, `UPDATE` и `DELETE`, запросы и отчеты;

- ❑ в отношении путей доступа, безопасности и управления параллельным выполнением;

Примечание

Управление параллельным выполнением (concurrency control) — это возможность, которая допускает одновременный доступ к базе данных, обеспечивая в то же время целостность данных. Необходимость управления параллельным выполнением обоснована в табл. 6.6.

- ❑ в отношении требований и ограничений, предъявляемых к информации компаний.

Проверка исходного проекта базы данных начинается с тщательного пересмотра всех сущностей и последующего подробного исследования атрибутов, описывающих эти сущности. Эти действия обусловлены следующим:

- ❑ появление непредвиденных сведений об атрибутах может привести к пересмотру самой сущности. Возможно, некоторые компоненты, которые поначалу принимались за сущности, станут атрибутами других сущностей. Или элемент, который мы вначале рассматривали как атрибут, может оказаться состоящим из множества вложенных элементов и станет причиной создания одной или более новых сущностей;
- ❑ подробное рассмотрение сведений об атрибутах поможет раскрыть природу связей, определенных первичными и внешними ключами. Неправильное определение связей приводит, прежде всего, к проблемам реализации, а впоследствии — к проблемам при разработке приложений;
- ❑ для выполнения требований технологического процесса и/или конечных пользователей бывает необходимо заменить первичный и внешний ключи новыми. Например, в системе обработки счетов, представленной на рис. 2.29 (см. гл. 2), мы создали первичный ключ, состоящий из атрибутов INV_NUMBER и LINE_NUMBER, чтобы заменить первоначальный первичный ключ, составленный из атрибутов INV_NUMBER и PROD_CODE. Мы сделали это для того, чтобы позиции в счете появлялись в том же порядке, в каком они были введены. Иногда для упрощения выполнения запросов и увеличения производительности системы необходимо создать первичный ключ, состоящий только из одного атрибута, заменив имеющийся первичный ключ, состоящий из нескольких атрибутов;
- ❑ до тех пор, пока сведения о сущностях (т. е. атрибуты и их характеристики) точно не определены, очень трудно оценить глубину нормализации. Знание уровней нормализации помогает избежать нежелательной избыточности;
- ❑ поскольку у нас уже есть, по крайней мере, черновой эскиз проекта, его тщательная проверка, скорее всего, повлечет за собой некоторые изменения. Эти изменения помогут обеспечить соответствие проекта требованиям конечного пользователя.

Так как проектирование базы данных в действительности, как правило, выполняется группой проектировщиков, необходимо постараться выполнить основные компоненты проекта в виде независимых модулей. (*Модуль* — это компонент информационной системы, который служит для выполнения специфических функций, таких

как инвентаризация, заказы, платежные ведомости и т. д.) Создание и использование модулей преследует несколько важных целей:

- ❑ модули и даже их фрагменты могут быть переданы различным подразделениям проектной группы, что позволит ускорить разработку проекта;
- ❑ модули упрощают процесс проектирования. Огромное количество сущностей в сложном проекте может привести в отчаяние. А каждый модуль содержит контролируемое ограниченное количество сущностей;
- ❑ модули можно использовать в качестве прототипа. При разработке и реализации прикладных программ с помощью модулей можно без особого труда обнаружить трудные места (быстрая разработка с помощью прототипов к тому же повышает конфиденциальность);
- ❑ даже если всю систему в целом нельзя ввести в строй достаточно быстро, реализация одного или нескольких модулей продемонстрирует прогресс в разработке и, по крайней мере, часть системы сможет обслуживать пользователей.

Польза модулей еще и в том, что они представляют собой фрагменты ER-диаграмм. Правда, фрагменты являются потенциальным источником проблемы: они не включают в себя все компоненты ER-модели и потому не могут выполнять все необходимые действия. Чтобы избежать этого, нужно проверять модули по отношению к полной ER-модели. Пошаговый процесс проверки (verification) представлен в табл. 6.4.

Таблица 6.4. Проверка ER-модели

Шаг	Описание
1	Выявление основной сущности ER-модели
2	Выявление каждого модуля и его компонентов
3	Определение требований к транзакциям каждого модуля: Внутренние: Обновление/Вставка/Удаление/Запросы/Отчеты Внешние: Интерфейсы модулей
4	Проверка всех процессов в отношении ER-модели
5	Внесение всех изменений, которые необходимо сделать после выполнения Шага 4
6	Повторить Шаги 2—5 для всех модулей

Помните, что процесс проверки требует непрерывной проверки бизнес-транзакций, а также системных требований и требований пользователя. Проверку необходимо повторить для каждого системного модуля. На рис. 6.10 ясно показана итеративная сущность этого процесса.

Процесс проверки начинается с выбора основной (наиболее важной) сущности. Основная сущность определяется ее участием в большинстве связей модели и тем, что она является центром большинства системных операций. Другими словами, для ее выявления проектировщик должен выбрать сущность, которая входит в наибольшее

число связей (на ER-диаграмме это сущность, у которой самое большое число линий связи).

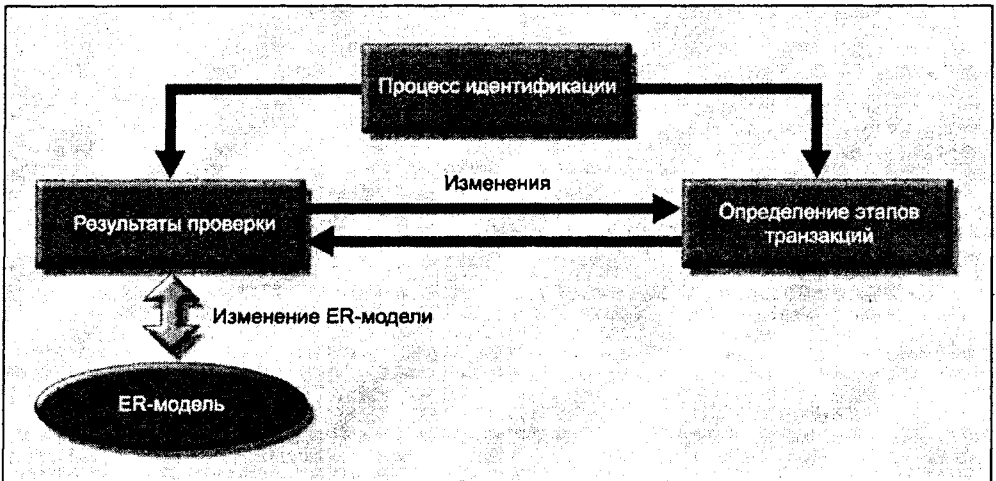


Рис. 6.10. Итеративный процесс проверки ER-модели

Следующий шаг проверки состоит в определении модуля или подсистемы, к которой принадлежит основная сущность, а также в определении области действия модуля и его границ. Сущность принадлежит модулю, который использует ее чаще всего. Как только каждый модуль определен, в структуру модуля помещается основная сущность с тем, чтобы рассмотреть модуль более детально.

В среде основной сущности (в ее модуле) необходимо:

- ❑ убедиться в *сцепленности модуля (module cohesivity)*. Термин "сцепленность" описывает силу связей между сущностями модуля. Модуль должен обладать высокой сцепленностью — то есть сущности модуля должны быть сильно связаны и модуль должен быть завершенным и автономным;
- ❑ проанализировать все связи модулей друг с другом на возможность последующего связывания модулей. Термин *связываемость модулей (module coupling)* описывает степень независимости модулей друг от друга. Модули должны обладать низкой связываемостью, что демонстрирует их слабую зависимость от других модулей. Чем ниже связываемость, тем меньше ненужных связей между модулями, что позволяет создать в полном смысле модульную систему и устранить ненужные связи между сущностями.

Процессы можно классифицировать:

- ❑ по частоте: ежедневные, еженедельные, ежегодные или внеплановые;
- ❑ по типу операций: INSERT (вставка) или ADD (добавление), UPDATE (обновление) или CHANGE (изменение), DELETE (удаление), запросы и отчеты, пакетные процедуры, обслуживание и создание резервных копий.

Все идентифицированные процессы необходимо проверить на ER-модели. При необходимости нужно внести соответствующие изменения. Процесс проверки повторяется для всех модулей модели. Возможно, в процессе проверки придется ввести в концептуальную модель дополнительные сущности и атрибуты.

К этому моменту определена концептуальная модель, т. е. модель, независимая от оборудования и программного обеспечения. Такая независимость гарантирует переносимость системы с платформы на платформу. Переносимость может продлить жизнь базы данных, позволяя перевести ее в другую СУБД и/или на другие технические средства.

Проектирование распределенной базы данных

Фрагменты проекта базы данных могут физически находиться в разных местах. Процессы, требующие доступа к базе данных, также могут варьироваться в зависимости от их местонахождения. Например, процессы системы учета розничной торговли и процессы информационного хранилища, скорее всего, находятся в разных местах. Если процессы базы данных распределены по всей системе, проектировщик должен также разработать распределение данных и стратегии размещения базы данных. Мы подробно исследуем сложности проектирования, возникающие при распределенных процессах, в *гл. 10*.

II. Выбор программного обеспечения СУБД

Выбор программного обеспечения очень важен для успешного выполнения операций информационной системы. Следовательно, преимущества и недостатки программного обеспечения необходимо тщательно изучить. Конечный пользователь должен знать об ограничениях как СУБД, так и базы данных, чтобы избежать непредсказуемых результатов.

Хотя факторы, влияющие на решение о приобретении СУБД, разные в разных компаниях, все же можно выделить наиболее общие из них.

- ☐ *Стоимость.* Стоимость покупки, обслуживания, эксплуатации, лицензии, установки, обучения и конвертирования.
- ☐ *Инструментальные средства и возможности СУБД.* В программное обеспечение некоторых СУБД включены различные инструментальные средства, облегчающие разработку приложений. Например, выполнение запроса по образцу (QBE, Query By Example), системы цветовой раскраски, генераторы отчетов, генераторы приложений, словари данных и т. д. помогают создать более привлекательную программную среду как для пользователей, так и для прикладных программистов. На выбор СУБД оказывают влияние средства обеспечения администрирования БД, обеспечения запросов, простоты использования, эффективности, безопасности, управления параллельным выполнением операций, обработки транзакций, а также поддержка программ сторонних фирм.
- ☐ *Тип основной модели.* Иерархическая, сетевая, реляционная, объектно-реляционная или объектная.
- ☐ *Переносимость.* Независимость от платформы, операционной системы и языка.
- ☐ *Требование СУБД к оборудованию.* Процессор, RAM (оперативная память), дисковое пространство и т. д.

III. Логическое проектирование

Примеры логического проектирования, представленные в гл. 3, мы подробнее изучим в гл. 7 и 8. Логическое проектирование начинается после принятия решения об использовании определенной модели БД (иерархическая, сетевая или реляционная). После того как модель базы данных определена, можно отобразить концептуальное проектирование на логический проект, привязанный к выбранной модели БД. Поэтому логическое проектирование является программно зависимым.

Логическое проектирование используется для перенесения концептуального проекта на внутреннюю модель выбранной системы управления базой данных (СУБД), например, DB2, SQL Server, Oracle, IMS, Informix, Access и т. д. При этом все объекты отображаются на модель с определенной структурой, используемой выбранным программным обеспечением БД. Для реляционных СУБД логическое проектирование включает в себя проектирование таблиц, индексов, представлений, транзакций, авторизации доступа и т. д. Далее мы конвертируем простую концептуальную модель, представленную на рис. 6.11, в логический проект, основанный на реляционной модели.

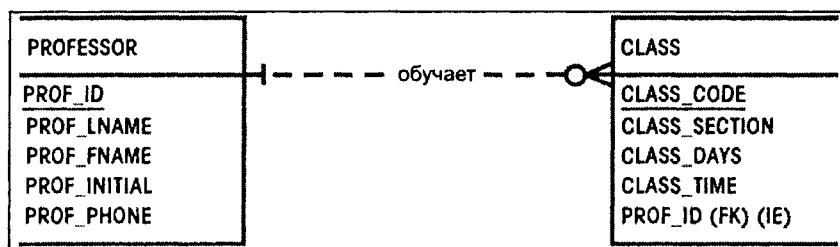


Рис. 6.11. Простая концептуальная модель⁴

Преобразование концептуальной модели, представленной на рис. 6.11, требует определения доменов атрибута, проектирования необходимых таблиц и форматов ограничения доступа. Например, определения доменов для атрибутов CLASS_CODE, CLASS_DAYS и CLASS_TIME, представленных на рис. 6.11, можно записать так:

CLASS_CODE (допустимый код группы)

Тип: числовой

Диапазон: мин. значение = 1000 макс. значение = 9999

Формат отображения: 9999

Длина: 4

⁴ Вы должны помнить из гл. 3, что генерируемая в Visio аббревиатура IE обозначает Inversion Entry (инвертированный компонент), определяемый в Visio как "неуникальный идентификатор доступа к сущности". Это означает, что внешний ключ не обязательно должен быть уникальным. И поэтому так обозначается реализация "строгая сущность-слабая (неидентифицируемая) связь" (рис. 6.11), но "строгая (идентифицируемая) связь-слабая сущность" (см. рис. 6.7). Если сущность слабая, то FK представляет собой часть PK, поэтому он не может иметь пустое значение или быть многозначным.

CLASS_DAYS (допустимое обозначение дня)

Тип: символьный

Допустимые значения: MWF, TTh, M, T, W, Th, F

Формат отображения: XXX

Длина: 3

CLASS_TIME (допустимое время)

Тип: символьный

Формат отображения: 99:99 (24-часовой формат)

Диапазон отображения: от 00:01 до 24:00

Длина: 5

Эти таблицы соответствуют сущностям PROFESSOR и CLASS, представленным на рис. 6.11, а столбцы таблицы соответствуют атрибутам этих сущностей (PROF_ID, PROF_LNAME, PROF_FNAME, PROF_INITIAL, PROF_PHONE, CLASS_CODE, CLASS_SECTION и т. д.). Например, первоначально схема таблицы PROFESSOR может выглядеть так, как представлено в табл. 6.5.

Таблица 6.5. Простая схема таблицы PROFESSOR

PROF_ID	PROF_LNAME	PROF_FNAME	PROF_INITIAL	PROF_PHONE
2134	Darnell	George	D	615-892-9118
2139	Smithson	Anne	M	615-892-1123
2210	Rodriguez	Maria	H	615-892-2368
2214	Vann	James	M	615-890-3246

В процессе логического проектирования также должны быть определены права на пользование базой данных: кто будет иметь право использовать таблицы, какие участки таблицы (или таблиц) доступны и каким пользователям. В реляционной среде для того, чтобы ответить на такие вопросы, требуется определить соответствующие представления и права доступа.

Короче говоря, логическое проектирование переводит программно-независимую концептуальную модель в программно-зависимую модель определением подходящих доменов, необходимых таблиц и соответствующих ограничений на доступ. Теперь наступило время определить физические требования, позволяющие системе функционировать в рамках выбранного оборудования.

IV. Физическое проектирование

Физическое проектирование представляет собой процесс определения характеристик хранилища данных и доступа к данным в БД. Свойства хранилища зависят от типа устройств хранения, типа доступа к данным, поддерживаемого системой, и от СУБД. На этапе физического проектирования определяется не только местоположение данных на устройствах хранения, но и общая производительность системы.

Физическое проектирование является чисто технической задачей, более типичной для клиент-серверных архитектур и систем мэйнфреймов, чем для конфигураций персональных компьютеров. И все-таки даже в сложных системах с миникомпьютерами и мэйнфреймами современное программное обеспечение БД берет на себя основную часть физического проектирования.

Физическое проектирование имело большое значение в устаревших иерархических и сетевых моделях, описанных в гл. 1. Реляционные БД более изолированы от физического уровня, чем старые модели. Все же несмотря на то, что в реляционной модели, как правило, сложные физические свойства скрыты от пользователя, они оказывают большое влияние на производительность реляционных БД. Производительность может определяться характеристиками устройства хранения, например, временем поиска, размером сектора (блока), размером буфера, числом пластин жесткого диска и количеством головок чтения/записи. Кроме того, на производительность реляционной базы данных и на время доступа к данным влияет такой фактор, как правильно созданный индекс.

Необходимо тщательно проанализировать даже тип данных запроса для того, чтобы определить оптимальный метод доступа, который удовлетворял бы установленным требованиям к приложению, объему хранимых данных и предполагаемой производительности. Некоторые СУБД автоматически резервируют пространство, необходимое для хранения определений БД и пользовательских данных на постоянном запоминающем устройстве. Это необходимо для гарантии хранения данных в последовательной форме и в одном месте, что уменьшает время доступа к данным и увеличивает производительность системы.

Физическое проектирование становится еще более сложным, если данные распределены по различным местам, поскольку эффективность будет определяться пропускной способностью средств связи.

С учетом всех этих трудностей становится понятно, почему проектировщики приносят программное обеспечение, которое берет на себя большую часть работ, связанных с физическим проектированием.

В предыдущих разделах этой главы мы отдельно обсуждали логическое и физическое проектирование. На самом деле, два этих процесса выполняются параллельно, таблица за таблицей (или файл за файлом). Логическое и физическое проектирование также параллельно выполняются в рамках иерархической и сетевой модели. Такие действия требуют от проектировщика глубокого понимания программного и аппаратного обеспечения для того, чтобы использовать их с максимальной эффективностью.

6.4.3. Реализация и загрузка данных

В большинстве современных СУБД, таких как DB2 фирмы IBM, Microsoft SQL Server или Oracle, реализация новой базы данных требует создания специальных структурных компонентов для хранения таблиц конечных пользователей. К таким компонентам относятся *группа хранения (storage group)*, *область таблиц (table space)* и сами *таблицы* (рис. 6.12). Обратите внимание, что в области таблиц может находиться больше одной таблицы.

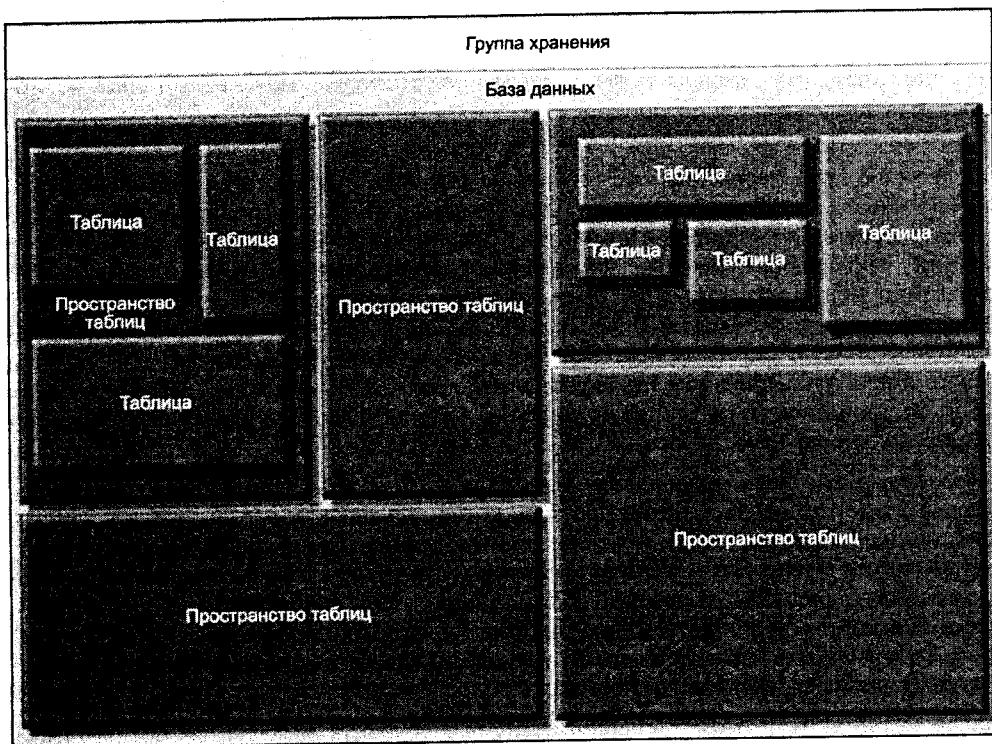


Рис. 6.12. Физическая организация среды базы данных DB2

Например, для реализации логического проекта в СУБД DB2 фирмы IBM потребуется выполнить следующие шаги.

Шаг 1. Создать группу хранения базы данных. Этот шаг (выполняемый системным администратором или SYSADM) необходим для баз данных, развертываемых на мэйнфреймах, например, для DB2. Другие СУБД могут создавать подобные группы хранения автоматически. При создании базы данных (см. Шаг 2) обязательно изучите справочное руководство вашей СУБД с тем, чтобы выяснить, нужно ли вам создавать группы хранения, и если это необходимо, то какой синтаксис команд при этом используется.

Шаг 2. Создать в рамках группы хранения базу данных (также выполняется системным администратором).

Шаг 3. Назначить право на использование базы данных группе системного администратора (DBADM)⁵.

Шаг 4. Создать внутри базы данных область (области) таблиц (обычно выполняется DBADM).

⁵ В других СУБД, упоминаемых в книге (не DB2), администратор базы данных называется DBA.

Шаг 5. Создать таблицу (таблицы) внутри области таблиц (также обычно выполняется DBADM). Общий вид создания таблицы в SQL может выглядеть так:

```
CREATE TABLE PROFESSOR (  
  PROF_ID      NUMBER      NOT NULL,  
  PROF_LNAME   CHAR(15)    NOT NULL,  
  PROF_PHONE   CHAR(8),  
  PRIMARY KEY (PROF_ID));  
  
CREATE TABLE CLASS (  
  CLASS_CODE    NUMBER      NOT NULL,  
  CLASS_SECTION NUMBER      NOT NULL,  
  CLASS_DAYS    CHAR(3)     NOT NULL,  
  CLASS_TIME    CHAR(14)    NOT NULL,  
  PROF_ID       NUMBER,  
  PRIMARY KEY (CLASS_CODE),  
  FOREIGN KEY (PROF_ID) REFERENCES PROFESSOR;
```

Шаг 6. Присвоить права области таблиц и самим таблицам, которые в них определены (еще одна обязанность DBADM). Права доступа могут быть ограничены представлениями, а не доступом к самим таблицам. Создание представлений не требует доступа к базе данных в реляционной среде, и представления предпочтительны с точки зрения обеспечения безопасности.

Права доступа к таблице с названием PROFESSOR могут быть предоставлены лицам с идентификационным именем PROB с помощью такой команды:

```
GRANT USE OF TABLE PROFESSOR  
TO PROB;
```

Вместо таблицы PROFESSOR можно использовать представление с именем PROF:

```
CREATE VIEW PROF  
  SELECT PROF_LNAME  
  FROM PROFESSOR;
```

После создания базы данных необходимо загрузить данные в таблицы. Если данные хранятся к этому моменту в формате, отличном от того, который требует СУБД, то перед загрузкой их необходимо преобразовать в подходящий формат.

При использовании иерархической модели реализация должна включать в себя язык определения данных (DDL) для каждой базы данных. Кроме того, иерархическая база данных, например, IMS, требует создания *блоков спецификации программ* (PSB, program specification block), чтобы программы могли получать доступ к базе данных.

При использовании сетевой модели требуется создание DDL, в который должны быть включены все типы записей, способы размещения, множества, способы вставки, выборки множеств и т. д. Сетевые базы данных также требуют использования *языка опи-*

сания физической организации (DMCL, device media control language) и всех необходимых представлений для того, чтобы программы могли получить доступ к БД.

Для вас, видимо, не стало неожиданностью то, что большинство операций, описанных в жизненном цикле базы данных (DBLC), напоминают операции жизненного цикла разработки систем (SDLC). Все-таки именно SDLC представляет оперативную среду, в которой разрабатывается DBLC. Сводка всех схожих операций, выполняемых в среде SDLC и DBLC, представлена на рис. 6.13.

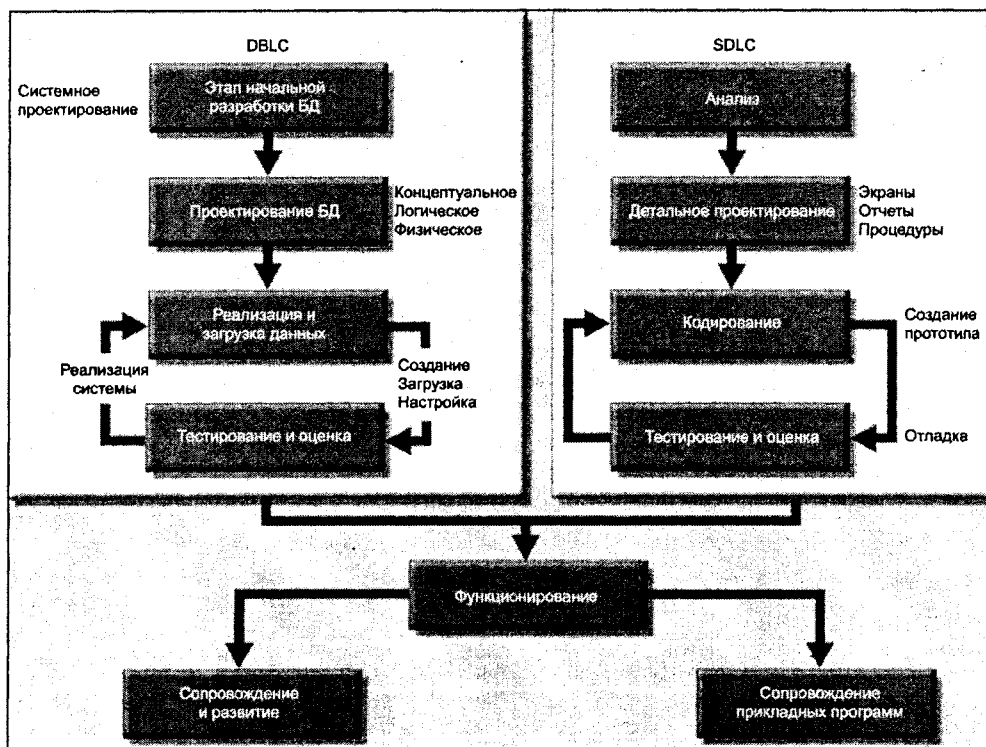


Рис. 6.13. Сходные операции в среде DBLC и SDLC

Примечание

Приведенная сводка операций по реализации базы данных предполагает использование сложных СУБД. Все современные базы данных для мэйнфреймов и для архитектуры клиент-сервер предлагают возможности, которые будут обсуждаться в дальнейшем. Ко времени написания этой книги лишь несколько баз данных для персональных компьютеров предлагали подобные возможности. Более сложные БД для персональных компьютеров, которые поддерживают такие возможности, как правило, происходят от БД для мэйнфреймов (например, DB2, Oracle и SQL Server в настоящее время используются и на персональных компьютерах).

В процессе реализации и загрузки данных также необходимо обратить внимание на производительность, безопасность, создание резервных копий и восстановление, целостность, стандарты компании и управление параллельным выполнением. Эти вопросы будут обсуждаться далее (эти проблемы также обсуждаются подробно в гл. 9 и 10).

Производительность

Производительность базы данных это один из наиболее важных факторов для некоторых реализаций баз данных. Однако не у всех СУБД есть встроенные средства наблюдения за производительностью, что затрудняет оценку эффективности работы СУБД.

Оценка производительности затрудняется еще и тем, что отсутствуют стандартные система оценок определения эффективности баз данных. Производительность меняется в зависимости от оборудования и программного обеспечения. Естественно, размер базы данных также влияет на производительность: поиск по 10 кортежам, конечно же, выполняется быстрее, чем по 100 000.

К важным факторам, влияющим на производительность БД, относятся также параметры конфигурации системы и базы данных, такие как размещение данных, определение путей доступа, использование индексов и размер буфера.

Хотя производительность очень важна, она не является единственным (и даже не самым главным) фактором оценки базы данных. Не менее важными являются проблемы, которые мы будем обсуждать ниже.

Безопасность

Данные, хранящиеся в базе данных компании, должны быть защищены от неавторизованного доступа (нетрудно вообразить, что произойдет, если студенты получат доступ к базе данных студентов или сотрудник получит доступ к платежной ведомости!). Следовательно, необходимо обеспечить (по крайней мере) следующие мероприятия.

- ❑ *Физическая безопасность (physical security)*. Разрешает физический доступ только авторизованным лицам. Поскольку физическая безопасность зависит от типа реализации БД, однако ее не всегда можно осуществить на практике. Например, исследовательская база данных университета не лучший пример физической безопасности. Существование больших мультисерверных, многотерминальных сетей часто не позволяет обеспечить физическую безопасность.
- ❑ *Защита с помощью паролей (password security)*. Позволяет устанавливать права доступа определенным авторизованным пользователям. Защита с помощью паролей обычно осуществляется во время регистрации пользователя в системе.
- ❑ *Права доступа (access rights)*. Могут быть установлены с помощью программного обеспечения БД. Права доступа могут ограничивать выполнение операций (CREATE, UPDATE, DELETE и т. д.) на предопределенных объектах, таких как базы данных, таблицы, представления, запросы и отчеты.
- ❑ *Контрольные журналы (audit trails)*. Предоставляются СУБД для контроля нарушений прав доступа. Хотя контрольные журналы являются средством "пост-фактум", одно лишь их существование препятствует неавторизованному доступу.

- ❑ *Шифрование данных (data encryption)*. Нужно для того, чтобы неавторизованные персоны, возможно, преодолевшие некоторые уровни защиты БД, не могли использовать данные напрямую.
- ❑ *Бездисковые рабочие станции (diskless workstation)*. Позволяют конечным пользователям получать доступ к БД, не загружая какую-либо информацию на свою рабочую станцию.

Резервное копирование и восстановление

Своевременность информации — критично важная вещь в современном информационном пространстве. К сожалению, в базах данных нередко происходит потеря информации вследствие непреднамеренного удаления, аварийного выключения питания и т. д. Процедуры резервного копирования и восстановления являются своего рода предохранительным клапаном, позволяющим администратору БД обеспечить доступность и непротиворечивость данных.

Целостность

Целостность данных реализуется с помощью надлежащего использования правил первичного и внешнего ключа.

Стандарты компании

Стандарты базы данных могут частично определяться специфическими требованиями предприятия. Администратор БД должен реализовать и проводить в жизнь эти стандарты.

Управление параллельным выполнением операций

Эта функция позволяет получать одновременный доступ к БД при сохранении целостности данных. Ошибки управления параллельным выполнением могут сильно снизить эффективность базы данных. Рассмотрим сценарий, представленный в табл. 6.6:

1. Начальный хранимый запас = 500 единиц.
2. Процесс А уменьшает запас на 150 единиц; процесс В уменьшает запас еще на 300 единиц.
3. Конечный запас должен быть равен $500 - 150 - 300 = 50$. Однако если использовать неправильную последовательность, приведенную в табл. 6.6, в базе данных будет храниться неправильное значение 200.

Таблица 6.6. Необходимость управления параллельным выполнением операций

Процесс А (в памяти)	Процесс В (в памяти)	База данных	Период
Считывание 500		500	T1
БД = 500 – 150	Считывание 500	500	T2
	Запись	350	T3
	БД = 500 – 300 (запись)	200	T4
	Значение в БД:	200	

Табл. 6.6 показывает, что может произойти в базе данных при проведении транзакций. Вот как можно описать последовательность транзакций, представленных в табл. 6.6.

1. В период T1 процесс А считывает доступный блок из 500 единиц.
2. В период T2 процесс В считывает доступный блок до того, как процесс А уменьшит значение в БД на 150 единиц. То есть процесс В тоже "видит" в БД 500 единиц.
3. В период T3 процесс А уменьшает значение в БД на 150 единиц и записывает оставшиеся 350 единиц в базу данных. *Но процесс В все еще оперирует в памяти исходным значением в 500 единиц.*
4. В период T4 процесс В уменьшает значение в памяти на 300 единиц и готов записать в БД оставшиеся 200 единиц! Поскольку процесс В записывает в БД после процесса А, то в БД будет записано значение (неправильное!) 200 единиц, которое будет использоваться в дальнейших транзакциях.

Такого сценария можно избежать с помощью алгоритма управления параллельным выполнением операций. В гл. 9 подробно описывается использование методов блокировки для управления параллельными операциями.

6.4.4. Тестирование и оценка

После того как данные загружены в БД, администратор БД тестирует и настраивает базу данных на оптимальную производительность, целостность, параллельный доступ и ограничения безопасности. Этап тестирования и оценки происходит параллельно с программированием приложений.

Программисты используют инструментарий БД для создания прототипов приложений во время разработки программ. На этапе создания прототипов прикладным программистам особенно полезны такие инструментальные средства, как генераторы отчетов, генераторы экранов и генераторы меню.

Если реализация БД не отвечает некоторым системным критериям, то для ее улучшения можно применять некоторые специальные действия:

- в целях повышения производительности проектировщик должен настроить специфические системные параметры и параметры СУБД. Сведения по настройке таких параметров лучше всего искать в справочном руководстве по программному и аппаратному обеспечению;
- модифицировать физический проект (например, применение индексирования увеличивает производительность и облегчает позиционирование указателя);
- модифицировать логический проект;
- обновить или изменить программное обеспечение СУБД и/или аппаратную платформу.

6.4.5. Функционирование

После того как база данных прошла стадию оценки, можно начинать ее эксплуатацию. К этому моменту времени БД, ее руководство, пользователи и прикладные программы представляют собой законченную информационную систему.

Начало этапа функционирования неизбежно является отправной точкой процесса развития системы. Как только все конечные пользователи начнут работать с БД, всплывут проблемы, не проявившиеся во время стадии тестирования. Некоторые из таких проблем достаточно серьезны и могут стать причиной разработки специальных программных "заплаток", в то время как другие являются мелкими недочетами. Например, если проект БД реализуется как интерфейс с Интернетом, то большой объем транзакций может вызвать сбой даже в хорошо спроектированной системе. В таком случае проектировщики должны будут идентифицировать источник (или источники) узких мест и выработать альтернативное решение, при этом можно подключить программное обеспечение выравнивания сетевой нагрузки, чтобы распределить транзакции между несколькими компьютерами, увеличить допустимый кэш СУБД и т. д. В любом случае постоянные изменения для проектировщика являются нормой.

6.4.6. Сопровождение и развитие

Администратор БД должен быть готов к процедурам, связанным с постоянным обслуживанием базы данных:

- ☐ профилактическое обслуживание (резервное копирование);
- ☐ корректирующее обслуживание (восстановление);
- ☐ адаптивное обслуживание (повышение производительности, добавление сущностей и атрибутов и т. д.);
- ☐ назначение прав доступа и их обслуживание для новых и прежних пользователей;
- ☐ ведение статистики доступа к БД для улучшения эффективности и полезности системного аудита и для наблюдения за производительностью системы;
- ☐ периодическая проверка безопасности на основе системной статистики;
- ☐ периодические (ежемесячные, ежеквартальные или ежегодные) сводки по использованию системы для выяснения внутренних затрат и планирования бюджета предприятия.

Как уже говорилось, в системе постоянны только изменения. С учетом возможных изменений требований к информации, дополнительные формы отчетности и новые форматы запросов потребуют изменения приложений и, возможно, минимальных изменений компонентов и содержимого БД. Эти изменения можно легко реализовать только в том случае, если проект БД является гибким и если вся документация обновлена, и к ней можно получить оперативный доступ. И наступит момент, когда даже отлично спроектированную БД уже невозможно будет приспособить к новым реалиям, и тогда весь жизненный цикл DBLC начинается заново.

6.5. О стратегии проектирования базы данных

Есть два классических подхода к проектированию базы данных.

1. *Нисходящее проектирование (top-down design)*. Начинается с определения наборов данных, затем определяются элементы данных для каждого из таких наборов.

Этот процесс включает в себя идентификацию различных типов сущностей и определение атрибутов каждой сущности.

2. *Восходящее проектирование (bottom-up design)*. При таком подходе, прежде всего, выявляются элементы данных (предметы), которые затем группируются в наборы данных. Другими словами, прежде всего, определяются атрибуты, которые затем объединяются и формируют сущности.

Оба эти подхода представлены на рис. 6.14. Выбор подхода часто зависит от масштаба задачи или от личных предпочтений проектировщика. Хотя эти две методологии дополняют, а не исключают друг друга, все же для маленьких БД с небольшим количеством объектов, атрибутов и транзакций восходящее проектирование может оказаться более продуктивным. В случае, если количество, разнообразие и сложность сущностей, связей и транзакций велико, лучше использовать нисходящее проектирование. У большинства компаний есть стандарты по разработке систем и проектированию баз данных.

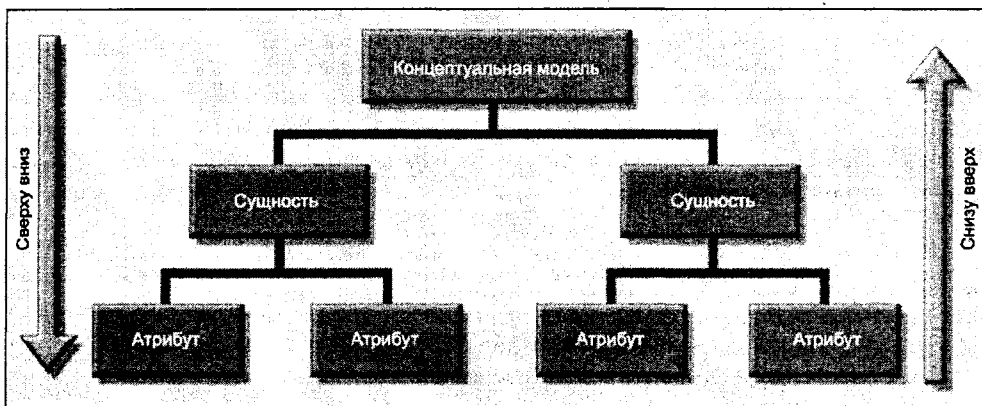


Рис. 6.14. Нисходящий и восходящий подходы к проектированию БД

Примечание

Даже в выборе нисходящего подхода в процессе нормализации, при котором проверяются все существующие табличные структуры, все равно используется восходящий метод. В ER-моделировании используется нисходящая методология, даже в случае, если выбор атрибутов и сущностей выполняется с применением восходящего метода. Поскольку и ER-моделирование, и нормализация являются основой большинства проектов, трудно отделить друг от друга нисходящую и восходящую методологии.

6.6. Централизованное и децентрализованное проектирование

На выбор одного из двух основных методов (нисходящего и восходящего) проектирования БД влияют такие факторы, как сфера действия и размер системы, стиль

руководства компании или структура компании (централизованная или децентрализованная). В зависимости от этих факторов проектирование БД может быть основано на двух совершенно различных стратегиях.

- **Централизованное проектирование.** Эта стратегия эффективна в том случае, когда компоненты данных состоят из относительно небольшого количества объектов и процедур. Централизованное проектирование типично для относительно простых и/или небольших баз данных и может быть выполнено одним человеком (администратором БД) или небольшой неформальной группой проектирования. Деятельность компании и масштаб проблем сильно ограничены, что позволяет даже одному проектировщику определить задачи, создать концептуальный проект, проверить концептуальный проект с точки зрения пользователей, определить системные процессы и ограничения данных, чтобы обеспечить действенность проекта и гарантировать соответствие проекта всем требованиям. (Хотя централизованное проектирование типично для небольших компаний, нельзя считать, что этот метод применим исключительно для них. Даже в сравнительно больших компаниях могут использоваться относительно небольшие базы данных.) На рис. 6.15 приведены основные операции централизованного проектирования. Обратите внимание, что при этом создается и проверяется единственная концептуальная модель.

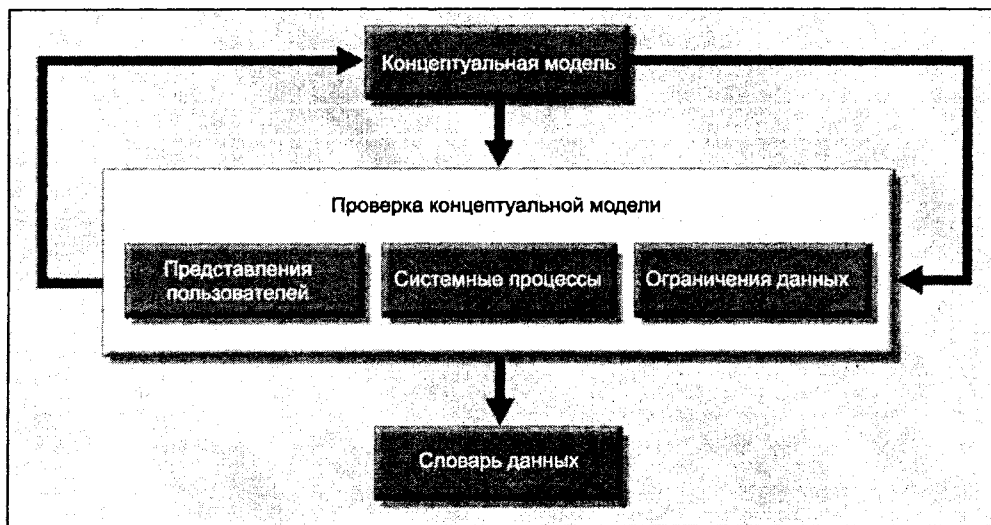


Рис. 6.15. Централизованное проектирование

- **Децентрализованное проектирование.** Используется в том случае, если компоненты данных состоят из большого числа сущностей и сложных связей, и на них выполняются комплексные операции. Децентрализованное проектирование также, скорее всего, будет применяться в случае, если задача сама по себе распределена по нескольким операционным сайтам, а каждый элемент представляет собой подмножество всего набора данных (рис. 6.16).

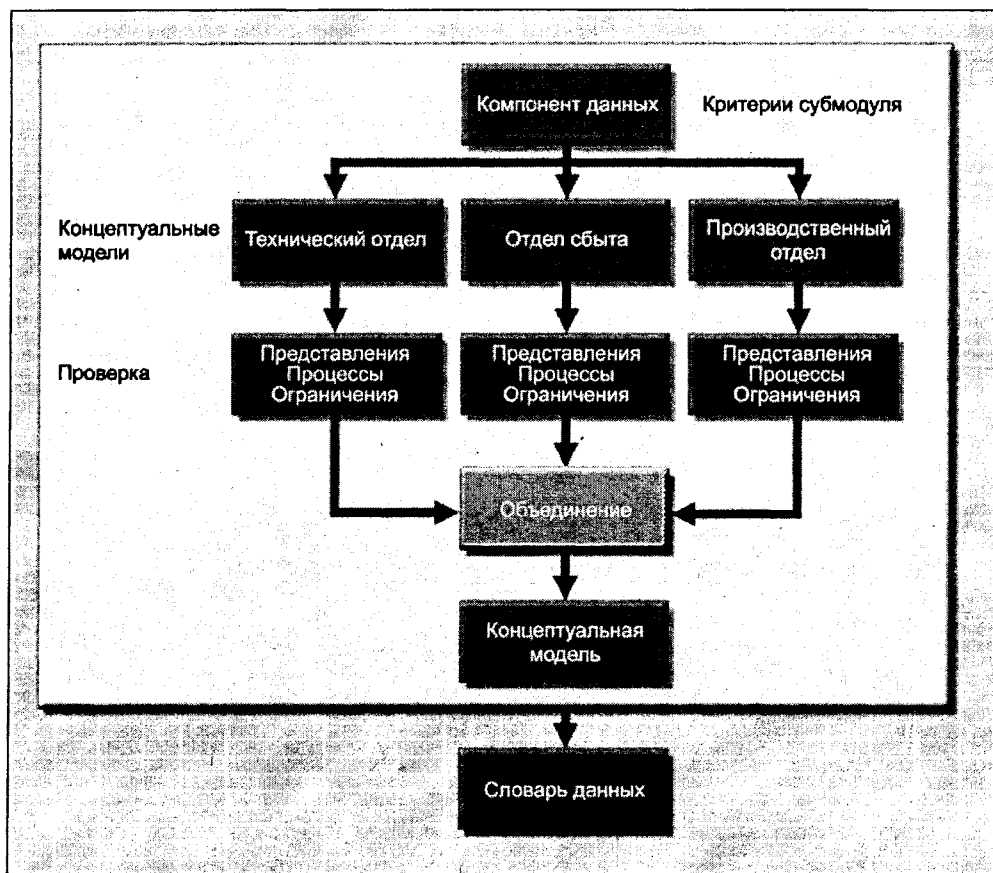


Рис. 6.16. Децентрализованная модель

В больших и сложных системах БД проектируется, как правило, не одним человеком. В таких случаях для проектирования БД, как правило, создается тщательно подобранная команда проектировщиков. В рамках децентрализованного проектирования задача проектирования БД разбивается на несколько модулей. Как только установлены критерии проекта, руководитель команды проектирования определяет группы, ответственные за выполнение фрагментов проекта или модулей.

Поскольку каждая группа проекта работает с отдельным подмножеством системы, определение границ и взаимозависимостей между подмножествами данных должно быть очень тщательным. Каждая группа проектирования создает концептуальную модель данных, соответствующую моделируемому подмножеству. Каждая концептуальная модель затем проверяется индивидуально по отношению к представлению пользователя, процессам и ограничениям каждого модуля. После процесса проверки все модули интегрируются в единую концептуальную модель. Поскольку словарь данных описывает характеристики всех объектов концептуальной модели, его роль в процессе интеграции очень важна. Естественно, после того как все подмножества объединены в укрупнен-

ную концептуальную модель, руководитель команды проекта должен проверить, что составная модель может выполнять все необходимые транзакций.

Необходимо помнить, что создание единой модели может потребовать решения перечисленных ниже проблем, которые могут возникнуть в процессе объединения (рис. 6.17).

- ❑ **Синонимы и омонимы.** Различные подразделения могут давать одним и тем же объектам разные имена (синонимы) или одинаковые имена могут использоваться для различных объектов (омонимы). Объект может быть сущностью, атрибутом или связью.
- ❑ **Сущность и подтипы сущности.** Атрибуты могут быть записаны с использованием различных типов (символьный, числовой) или же для одинаковых атрибутов могут быть определены различные домены. Определения ограничений также могут различаться. Проектировщик должен устранить подобные конфликты.

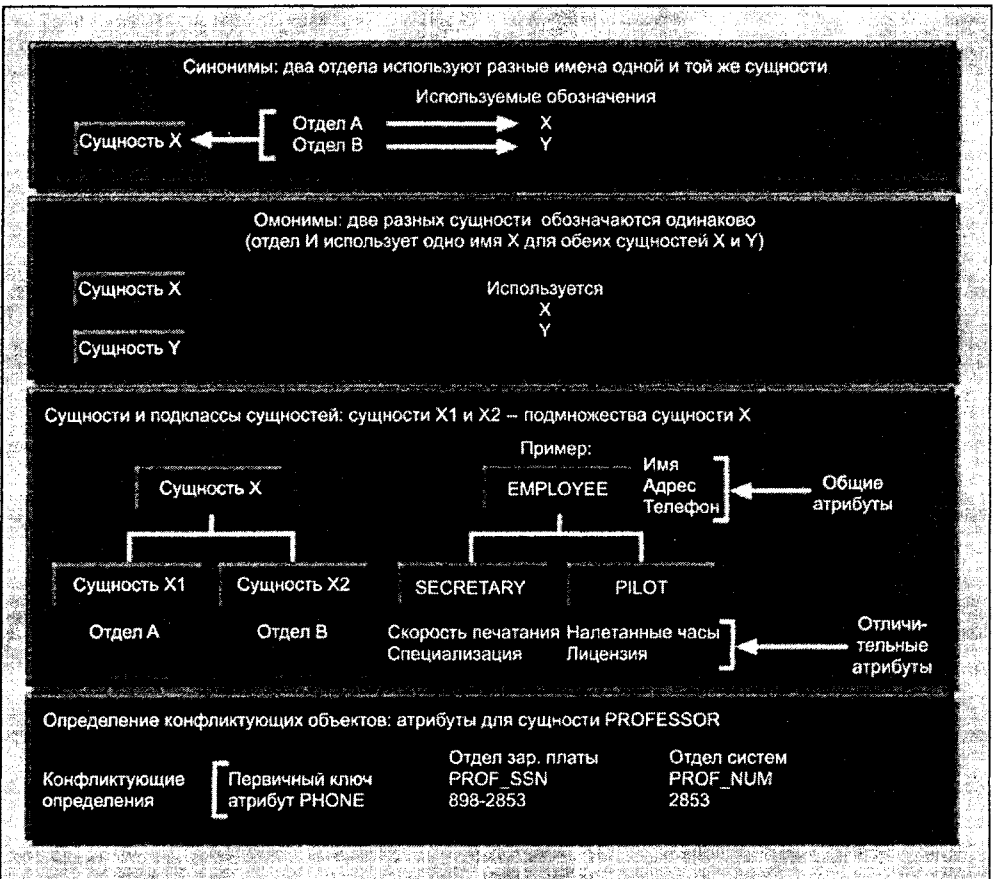


Рис. 6.17. Сводка проблем объединения

Резюме

Структурирование данных происходит в процессе обработки информации приложением. Информационная система разрабатывается с целью облегчения структурирования данных и управления ими. Поэтому база данных является важнейшей частью информационной системы. Системный анализ представляет собой процесс, который устанавливает необходимость разработки и сферу действия информационной системы. Разработка системы — это процесс создания информационной системы.

Жизненный цикл разработки системы (SDLC, System Development Life Cycle) отслеживает историю (жизненный цикл) приложения в информационной системе. SDLC можно подразделить на пять стадий: планирование, анализ, подробное системное проектирование, реализация и сопровождение. SDLC является скорее итеративным, чем последовательным процессом.

Жизненный цикл базы данных (DBLC, Database Life Cycle) описывает историю (жизненный цикл) базы данных внутри информационной системы. DBLC состоит из шести этапов: начальная разработка, проектирование БД, реализация и загрузка данных, тестирование и оценка, функционирование, а также обслуживание и развитие. Так же как и SDLC, процесс DBLC скорее итеративный, чем последовательный.

Поскольку база данных является основным компонентом информационной системы, процесс проектирования и реализации БД представляет для нас особый интерес. Процесс проектирования и реализации БД представляет собой серию определенных этапов.

1. Анализ данных и требований помогает определить представления конечных пользователей и потребность в данных.
2. В результате первого этапа получается определение значимых сущностей, атрибутов и связей, что в свою очередь позволяет получить соответствующую ER-модель. Компоненты ER-модели определяют способ получения нормализованных таблиц.
3. Концептуальная модель проверяется посредством определения ее основных процессов и заданием соответствующих правил INSERT (вставка), UPDATE (обновление) и DELETE (удаление). Кроме того, в процессе проверки просматриваются пользовательские представления и соответствующие ограничения данных.
4. После проверки анализируется проектирование распределенной БД для того, чтобы оценить местоположение таблиц, стратегию фрагментирования БД и требования доступа.
5. В идеальном случае выбор программного обеспечения делается одновременно с разработкой концептуальной модели (этапы 1—4). Однако на практике может случиться так, что программное обеспечение уже имеется и концептуальное проектирование в какой-то мере будет от него зависеть.
6. На основе концептуальной модели и программного обеспечения на этапе логического дизайна происходит перевод концептуальной схемы в определения таблиц, представлений и т. д., зависящих от конкретной СУБД. Логическое проектирование зависит, таким образом, от особенностей СУБД, но не зависит от оборудования.

7. На этапе физического проектирования определяются специфические структуры хранения и пути доступа, что естественно зависит от конкретного оборудования.
8. На последнем этапе реализации загружаются данные и создаются реальные таблицы и представления. При необходимости форматы атрибутов данных конвертируются в форматы, эффективно обрабатываемые имеющейся СУБД. На этом этапе завершается кодирование и тестирование прикладных программ.

Концептуальная часть проекта может выглядеть по-разному, в зависимости от используемой стратегии проектирования.

- *Нисходящая или восходящая стратегия.* При восходящем проектировании, прежде всего, идентифицируются элементы данных (атрибуты), а затем они группируются в наборы (сущности). При нисходящем проектировании сначала идентифицируются сущности, а затем определяются атрибуты каждой сущности. Выбор подхода зависит от области задачи и персональных предпочтений проектировщика.
- *Централизованный и децентрализованный подходы.* Относительно простые базы данных проще создавать на основе централизованного подхода, когда все действия компании хорошо просматриваются с точки зрения единой базы данных. Децентрализованный подход к проектированию имеет смысл применять в случае, когда операции компании распределены между многими сайтами (каждый набор данных на сайте является подмножеством всех операций компании) или когда база данных состоит из большого количества сущностей, между которыми есть сложные связи.

Основные термины

Автоматизированное проектирование систем — computer-assisted systems engineering (CASE)

Бизнес-правила — business-rules

Восходящее проектирование — bottom-up design

Границы — boundaries

Децентрализованное проектирование — decentralized design

Жизненный цикл базы данных — Database Life Cycle (DBLC)

Жизненный цикл разработки систем — Systems Development Life Cycle (SDLC)

Информационная система — information system

Концептуальное проектирование — conceptual design

Логическое проектирование — logical design

Модуль — module

Нисходящее проектирование — top-down design

Правило необходимого минимума информации — minimal data rule

Преобразование — transformation

Разработка баз данных — database development

Разработка систем — systems development

Связываемость модуля — module coupling

Системный анализ — systems analysis

Сфера действия — scope

Сцепленность модуля — cohesivity module

Управление параллельным выполнением операций — concurrency control

Физическое проектирование — physical design

Централизованное проектирование — centralized design

Вопросы

1. Что такое информационная система? Каково ее предназначение?
2. Какова роль системного анализа и разработки систем в информационных системах?
3. В чем различие структурированной и неструктурированной информации?
4. Что означает акроним SDLC. Дайте графическое представление SDLC.
5. Что означает акроним DBLC. Дайте графическое представление DBLC.
6. В чем различие между централизованным и децентрализованным концептуальным проектированием баз данных?
7. Что представляет собой правило необходимого минимума информации в концептуальном проектировании? Для чего оно необходимо?
8. Поясните различия между нисходящим и восходящим подходами к проектированию баз данных.
9. Что такое бизнес-правила? Почему они важны для проектировщика БД?
10. Какова роль словаря данных при проектировании БД?
11. Что такое управление параллельным выполнением операций и какова его цель?

Задачи

1. Центры обслуживания ABC Car Service & Repair Centers принадлежат дилерской фирме SILENT и обслуживают только эти марки автомобилей. Три центра ABC Car Service & Repair Centers обеспечивают обслуживание и ремонт во всем штате. Каждый из трех центров независим, в нем имеются директор, приемщик заказов и, по крайней мере, восемь механиков. В каждом центре есть склад всех необходимых запчастей.

Каждый центр также поддерживает неавтоматизированную систему файлов, в которой хранится история обслуживания каждого автомобиля: выполненный ремонт, использованные запчасти, стоимость, дата обслуживания, владелец и т. д.

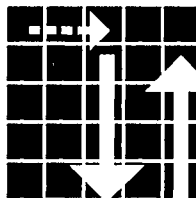
В файлах также хранится информация о запасах на складе, покупках, выручке, рабочем времени и платежных ведомостях.

Вы договорились с руководителем одного из таких центров спроектировать и реализовать компьютеризованную систему учета. Используя вышеприведенную информацию, сделайте следующее:

- определите подходящую последовательность действий, помечая приведенные ниже операции в нужном порядке (например, если вы считаете, что "загрузка базы данных" должна быть первым шагом, то пометьте его единицей).
 - _____ нормализация концептуальной модели
 - _____ получение общего описания деятельности компании
 - _____ загрузка базы данных
 - _____ создание описания каждого процесса системы
 - _____ тестирование системы
 - _____ создание схем информационных потоков
 - _____ создание концептуальной модели с помощью ER-диаграмм
 - _____ создание прикладных программ
 - _____ опрос механиков
 - _____ создание структур таблиц (файлов)
 - _____ опрос директора центра;
 - опишите различные модули, которые, по вашему мнению, должны быть включены в систему;
 - как вам поможет в разработке системы словарь данных? Приведите пример;
 - какие основные рекомендации необходимо дать директору? (Например, если система будет интегрированной, то какие модули будут интегрироваться? Какие преимущества дает такая интегрированная система? И т. д.);
 - какой подход, по вашему мнению, является наилучшим при концептуальном проектировании БД? Почему?
 - назовите и опишите, по крайней мере, четыре отчета, которые по вашему мнению должна выдавать система, и поясните их использование. Кто будет пользоваться этими отчетами?
2. Предположим, вас попросили создать информационную систему для промышленного предприятия, которое производит болты и гайки различной формы, размера и назначения. Какие вопросы вы зададите, и как повлияют ответы на проект БД?
- Как вы представляете себе SDLC?
 - Как вы представляете себе DBLC?
3. Предположим, что вы выполняете ту же работу, что и в задаче 2, для большого информационного хранилища. В чем эти действия будут похожими? В чем они будут различаться?

4. Используя те же процедуры и концепции, что и в задаче 1, как бы вы создавали информационную систему для Tiny College (см. гл. 3)?
5. Напишите корректную последовательность действий проектирования базы данных проката видеокассет (исходная ER-диаграмма представлена на рис. 6.7). Проект должен обеспечивать все необходимые действия по прокату, отслеживание платежей клиентов, распорядок работы сотрудников, а также отслеживание того, кто из сотрудников оформлял прокат видеокассет клиенту. После выработки действий по созданию проекта завершите ER-диаграмму, чтобы обеспечить успешную реализацию проекта БД. (Убедитесь, что проект правильно нормализован, и что он может выполнять надлежащие транзакции.)

Глава 7



Университетская лаборатория: концептуальный проект

Из этой главы вы узнаете:

- ☐ как выполняется начальная разработка базы данных;
- ☐ как составить описание операций;
- ☐ как записать бизнес-правила, на которых будет основываться проект базы данных;
- ☐ как перенести бизнес-правила в сегменты ER-диаграммы;
- ☐ как объединить сегменты ER-диаграммы для создания начальной ER-диаграммы.

Обзор

В этой главе и *гл. 8* мы соберем воедино все фрагменты мозаики проекта базы данных. Мы разработаем концептуальный проект БД с помощью идей и технологий, представленных в *гл. 3*.

Вы проследите эволюцию системы базы данных, пройдя этап начальной разработки БД и затем этап создания первоначальной ER-диаграммы концептуального проекта. В *гл. 8* будут рассмотрены дальнейшая эволюция и преобразование концептуального проекта в логический проект, который может быть реализован в среде любой реляционной СУБД.

На основе опыта преподавания проектирования баз данных мы сделали важный вывод: если человек не пройдет все ступени проектирования базы данных, то навряд ли он сможет успешно спроектировать и реализовать систему БД.

Целью представленного нами проекта будет автоматизация деятельности большой компьютерной лаборатории университета. Поскольку детальное описание процесса проектирования в этой главе основано на реальном проекте, вы столкнетесь с некоторыми жизненными проблемами и получите необходимые навыки их анализа и решения.

Работоспособная система представляет собой результат большого числа последовательных шагов. Чтобы проделать все эти шаги и не упустить некоторые важные детали в качестве плана работ, можно использовать табл. 7.1. В этой таблице представлены операции, выполняемые на первом этапе концептуального проектирования с

помощью ER-модели. Остальные задачи проектирования БД, представленные в табл. 7.1, мы обсудим в гл. 8.

Многие из этих действий на первый взгляд могут показаться тривиальными. Не поддавайтесь искушению пропустить их или не обратить на них внимания. Эти незначительные, казалось бы, детали как раз и отличают хороший проект от неудачного. Позже можно будет легко отказаться от лишних элементов проекта, но не стоит пренебрегать ими на этапе начальной разработки.

Проектирование БД это кропотливый труд. Мы надеемся, что детальное изучение этого примера поможет вам лучше понять суть процесса проектирования, который иногда может казаться слишком неорганизованным.

Таблица 7.1. Карта создания проекта базы данных
Университетской Компьютерной Лаборатории (UCL)

Этапы жизненного цикла БД	Результат	Раздел
<i>ЭТАП НАЧАЛЬНОЙ РАЗРАБОТКИ БД</i>	Предназначение UCL	7.1.1
	Организационная структура	7.1.2
	Описание операций	7.1.3
	Проблемы и ограничения	7.1.4
	Предназначение системы	7.1.5
	Сфера действия и границы	7.1.6
<i>ПРОЕКТИРОВАНИЕ БД</i>		
Концептуальное проектирование	Источники информации и пользователи	7.2.1
	Информационные потребности: требования пользователей	7.2.2
	Начальная ER-диаграмма	7.2.3
	Определение атрибутов и доменов	8.1—8.2
	Нормализация	8.2
	Проверка ER-модели	8.3
Логическое проектирование	Таблицы	8.4.1
	Индексы и представления	8.4.2
Физическое проектирование	Методы доступа	8.5
<i>РЕАЛИЗАЦИЯ</i>	Создание баз данных	8.6.1
	Загрузка и конвертирование БД	8.6.2
	Системные процедуры	8.6.3

Таблица 7.1 (окончание)

Этапы жизненного цикла БД	Результат	Раздел
ТЕСТИРОВАНИЕ И ОЦЕНКА	Измерение производительности	8.7.1
	Оценка безопасности	8.7.2
	Процедуры резервного копирования и восстановления	8.7.3
ФУНКЦИОНИРОВАНИЕ	Действие базы данных	8.8.1
	Операционные процедуры	8.8.2

Примечание

"Когда я слышу — забываю, когда я вижу — запоминаю, когда я делаю — изучаю".

Старая китайская поговорка.

7.1. Этап начальной разработки базы данных

Этап начальной разработки базы данных представляет собой в основном подробное описание имеющейся и предлагаемой структуры базы данных¹. Поэтому стадия начальной разработки включает в себя тщательный анализ структуры организации, выполняемых операций, выявление проблем и ограничений, выяснение предназначения системы, сферы действия системы и ее границы, определение источников информации и пользователей, а также требований конечных пользователей.

Начальная стадия разработки реальной базы данных, по-видимому, будет насчитывать сотни строк записей, поскольку подробное и точное описание имеет очень большое значение. Необходимость этого станет очевидной, когда вы поймете, что процесс реализации проекта БД основывается на бизнес-правилах, которые как раз и формируются на основе этапа начальной разработки БД. Если на начальном этапе упущены некоторые детали или допущены неточности, бизнес-правила, скорее всего, будут неполными и неточными. А это означает, что проект базы данных, основанный на этих бизнес-правилах, будет неудачным. Поскольку объем книги всегда ограничен, невозможно включить в нее все подробности начального этапа разработки. Однако мы постарались упомянуть в этом издании все важнейшие компоненты, с которыми вы столкнетесь на этапе начальной разработки в действительности.

Сведения этапа начальной разработки в основном получаются из опросов основных и конечных пользователей. Эти люди являются основными клиентами БД и их необходимо идентифицировать с особой тщательностью. Основными пользователями

¹ Термин "описание операций" иногда используется как синоним стадии начального изучения базы данных. Однако использование такого синонима годится только в том случае, если "операции" включают всю среду данных предприятия, а не только компоненты транзакций. Мы будем использовать название "описание операций" именно в таком, более узком смысле.

приложения *Университетская Компьютерная Лаборатория* (UCL, University Computer Lab), которое разрабатывается в данной главе, являются:

- заместитель декана (декан) Экономического Колледжа (College of Business);
- директор компьютерной лаборатории (CLD, Computer Lab Director), в чьи обязанности входит оперативное руководство лабораторией;
- ассистенты компьютерной лаборатории (LA, Lab Assistant), в чьи обязанности входит ежедневное обеспечение функционирования лаборатории;
- секретарь компьютерной лаборатории (CLS, Computer Lab Secretary), который помогает выполнять основные административные функции;
- дипломированные ассистенты компьютерной лаборатории (GA, Graduate Assistants), работающие под руководством директора лаборатории и обеспечивающие техническую поддержку и обучение преподавательского состава и сотрудников на оборудовании, принадлежащем Экономическому Колледжу.

Для краткости мы представим только несколько выборок из многочисленных опросов, выполненных в рамках этого проекта.

7.1.1. Задачи UCL

Университетская Компьютерная Лаборатория (UCL) расположена в центре кампуса (территории университета) и доступна всем студентам университета независимо от их специализации. Лаборатория UCL предоставляет доступ к многочисленным ресурсам всем сотрудникам университета. В ее распоряжении имеется 200 компьютеров, лазерные принтеры, сканеры. Лаборатория предоставляет сервис и поддержку группам пользователей, составленных из преподавателей, сотрудников или студентов. В число задач лаборатории входят:

- обеспечение пользователей контролируемым доступом к имуществу UCL, т. е. к микрокомпьютерам, принтерам, материалам, прикладному программному обеспечению и программной документации;
- сопровождение пользователей, работающих на оборудовании UCL, и оперативное решение возникающих проблем. Эти службы первоначально создавались для обучения пользователей основным операциям, выполняемым на компьютере (например, форматирование диска, копирование файлов, установка разрешенного программного обеспечения и включение/выключение компьютера).

7.1.2. Организационная структура

Осмысление организационной структуры UCL поможет проектировщику определить коммуникационные линии и установить необходимые требования к отчетности (рис. 7.1).

Директор лаборатории (CLD) руководит всей деятельностью UCL. Директору помогает секретарь лаборатории (CLS). В UCL работают дипломированные ассистенты (GA), а студенты старших курсов работают в качестве ассистентов (LA). Директор лаборатории отчитывается перед заместителем декана Экономического Колледжа, который подотчетен декану колледжа, а он в свою очередь отчитывается перед вице-

президентом Университета, подчиненным Президенту Университета. Хотя на рис. 7.1 представлена основная цепочка инстанций Экономического Колледжа, проект в основном относится к деятельности UCL. Однако поскольку другие заведующие кафедр периодически получают отчеты о деятельности лаборатории, поддерживают обратную связь с директором UCL и делают определенные денежные взносы в фонд UCL на основе использования лаборатории, они тоже включены в схему, представленную на рис. 7.1.

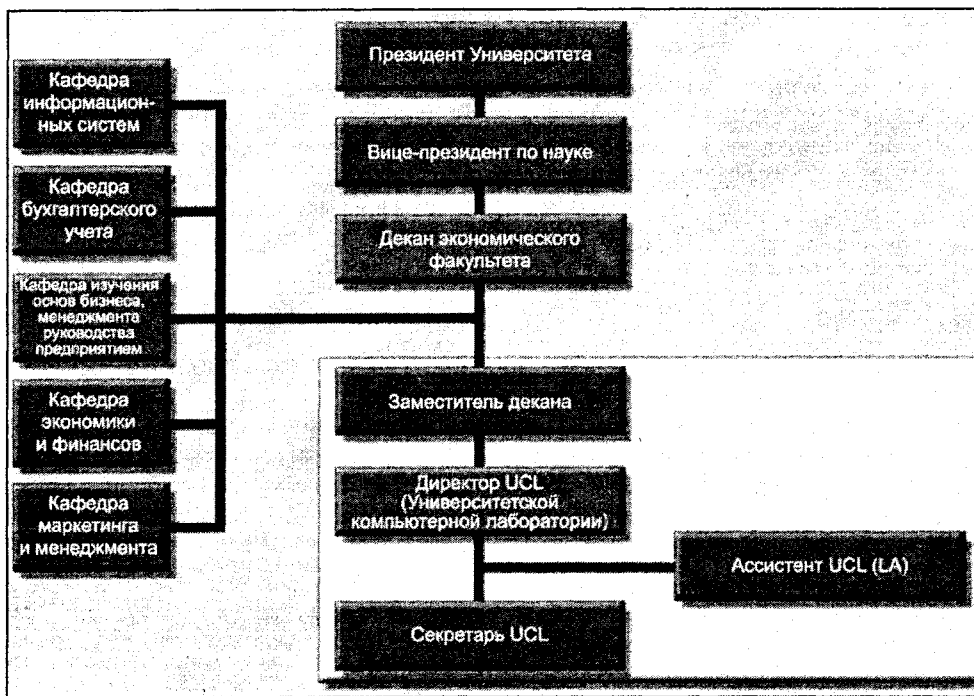


Рис. 7.1. Организационная структура Университетской Компьютерной Лаборатории (UCL)

Примечание

В структуре, представленной на рис. 7.1, опущены детали, которые не оказывают влияния на проектирование базы данных UCL. Например, ни отдел снабжения, ни другие колледжи Университета и их подразделения не представлены на этой схеме, поскольку они не входят в структуру отчетности UCL. По тем же причинам в схему не включены и другие компоненты организационной структуры Университета.

Схема, представленная на рис. 7.1, имеет важное значение по следующим причинам.

- Схема облегчает взаимодействие между конечными пользователями системы и проектировщиком (или проектировщиками) системы. Знание организационной

структуры помогает определить требования к информации (кому и какая требуется информация, в какой форме и когда).

- ☐ Схема помогает конечным пользователям системы определить и уяснить сферу их ответственности (как данный сотрудник вписывается в общую структуру, каковы его должностные обязанности).

Осмысление всей организационной структуры важно даже в том случае, когда система разрабатывается лишь для какого-то одного компонента, поскольку впоследствии система может развиваться и включать в себя другие части общей структуры. Проектировщик должен всегда помнить, что подразделения могут иметь различные (иногда конфликтующие) взгляды на данные и/или системные требования. Работа проектировщика базы данных заключается в выработке общего, удовлетворяющего все подразделения, представления о данных внутри организации.

7.1.3. Описание операций

После определения предназначения UCL и ее организационной структуры можно приступить к изучению ее деятельности. Лаборатория UCL осуществляет шесть видов операций:

- ☐ управление инвентаризацией/хранением/заказами;
- ☐ поддержка оборудования и организация ремонта;
- ☐ выдача и регистрация оборудования;
- ☐ ведение платежных ведомостей ассистентов;
- ☐ резервирование лаборатории для проведения занятий;
- ☐ управление доступом в лабораторию.

Управление инвентаризацией/хранением/заказами

Весь инвентарь UCL можно подразделить на оборудование, программное обеспечение, литературу и материалы.

- ☐ *Оборудование.* Сюда относятся компьютеры, терминалы, принтеры и т. д.
- ☐ *Программное обеспечение.* Включает в себя прикладные программы, например, обработка таблиц, текстовые процессоры, статистические программы и программное обеспечение баз данных.
- ☐ *Литература.* Справочная литература и руководства по использованию программного обеспечения.
- ☐ *Материалы.* Включают в себя все расходные материалы, например, красящие ленты для принтеров, бумага и т. д.

Каждый объект инвентаря определяется типом, а тип инвентаря используется для группирования схожих объектов. Тип инвентаря определяется четырехступенчатой иерархией: категория инвентаря, класс, тип и подтип. Эта иерархия представлена в табл. 7.2.

Таблица 7.2. Иерархия типов инвентаря

Объект	Категория	Класс	Тип	Подтип
Шлюз сети, Pentium 4, 1,5 ГГц	Оборудование	Компьютер	Настольный	Pentium
Бумага для лазерного принтера 8,5 × 11	Материалы	Бумага	Для лазерного принтера	8,5 × 11
Дискеты, 3,5" высокой плотности (HD)	Материалы	Диски	3,5	HD

Для создания полной и работоспособной инвентаризационной иерархической структуры проектировщик базы данных должен действовать совместно с конечными пользователями. Такое сотрудничество также позволит выработать идентификационные коды и описания для каждого типа инвентаря.

Тип инвентаря играет важную роль в способе записи объектов. Например, некоторые объекты инвентаря не требуют отслеживания отдельных составляющих. К таким объектам, называемым *несериализованными объектами* (малоценкой), относятся бумага для принтеров, диски и другие расходные материалы. Термин "несериализованный" означает, что объект не требует присвоения ему инвентарного номера или кода (вряд ли имеет смысл присваивать инвентарные номера отдельным пачкам бумаги для лазерного принтера). С другой стороны, дорогостоящие расходные материалы, такие как компьютеры и принтеры, требуют строгого учета с присвоением инвентарных номеров или кодов. Такие расходные материалы соответственно считаются сериализованными объектами (объектами строгого учета).

Присвоение учетного номера или кода позволяет конечным пользователям отслеживать местоположение объекта, его пользователей, статус и другую значимую информацию. Помните, что хотя оборудование, *как правило*, рассматривается как объект строгого учета, бизнес-правила определяются конечными пользователями и политикой предприятия, которые и формируют рамки классификации объектов.

Объекты инвентаря обновляются в случае, если:

- ☐ получен заказанный объект;
- ☐ объект выдан или зарегистрирован в описи пользователем лаборатории;
- ☐ расходный материал (бумага, чернильный картридж) получен со склада;
- ☐ CLD корректирует инвентарную ведомость. Например, если физическая проверка показала, что недостает коробки бумаги, то объем запаса этой позиции должен быть скорректирован.

Правилами Университета установлено, что заявки на покупку стоимостью более \$500 должны утверждаться расширенной комиссией Университета. После утверждения заявка направляется в отдел снабжения на тендер цен.

ОБЩЕЕ ПРАВИЛО

Расширенная комиссия Университета не требует от CLD указания в заявке на покупку определенного производителя и/или поставщика данного предмета, если только это не связано с проблемами совместимости. Позже вы узнаете, что это общее требование ока-

зывает влияние на выбор атрибута сущности. Например, мы должны определить оборудование по типу инвентаря таким образом:

Категория: *Оборудование*

Класс: *Компьютер*

Тип: *Настольный*

Подтип: *Pentium*

Пример требования на приобретение пяти компьютеров может выглядеть так:

Пять (5) компьютеров со следующими характеристиками: процессор Pentium 4 или эквивалентный, минимальная частота 1,5 ГГц, минимальный объем оперативной памяти повышенного быстродействия 256 Мбайт, 17-дюймовый цветной монитор с зерном 26 dpi, 54X-скоростной CD-ROM, 32 Мбайт видеопамяти, жесткий диск емкостью не менее 16 Гбайт, устройство чтения гибких дисков 3,5", дисковод Zip емкостью не ниже 100 Мбайт, клавиатура со 101 клавишей или эквивалентная, мышь MS IntelliMouse или эквивалентная, модем 56 Кбайт, динамики, MS Windows ME и MS Office XP.

Исключение из общего правила делается только в случае, если предмет покупается по государственному контракту.

Согласованные позиции, которые приобретаются не по государственному контракту, отправляются на тендер. Отдел снабжения отсылает заказ на покупку производителю, выигравшему тендер. Копия заказа отсылается в UCL. После этого UCL отправляет разрешение на оплату в отдел оплаты счетов Университета, чтобы оплатить заказ.

После получения изделия оно может сразу же использоваться в UCL или храниться на складе. Имеется несколько складских помещений, в каждом из которых содержится множество различных предметов, и инвентарь каждого типа может храниться в различных местах. Например, из трех имеющихся в наличии принтеров один может храниться на складе А, а два других — непосредственно в UCL, готовые к немедленному использованию. Материалы получаются со склада по мере необходимости.

Поддержка оборудования и организация ремонта

Компьютерное оборудование время от времени выходит из строя. Неисправное оборудование может ремонтироваться непосредственно в UCL. Если же поломку нельзя устранить на месте, то аппаратура отправляется на ремонт поставщику.

Когда оборудование требует ремонта, директор лаборатории (CLD) делает соответствующую запись в журнале неисправности оборудования (Bad Equipment Log). Если оборудование должно быть отправлено для ремонта поставщику, CLD делает запись в журнале отправки оборудования на ремонт (Hardware Returned for Service Log).

Выдача и учет оборудования

Хотя бюджет UCL и основное административное руководство являются прерогативой Экономического Колледжа, любой студент университета, преподаватель или сотрудник может воспользоваться услугами лаборатории. Проектировщик должен задать следующие вопросы для определения ограничений этого сервиса:

Проектировщик:

"Можно ли забирать оборудование из лаборатории на определенное время?"

Конечный пользователь:

"Брать на время оборудование из лаборатории могут только преподаватели и сотрудники. Для хранения записей о местонахождении оборудования и его использовании CLD должен контролировать выдачу аппаратуры. Преподаватель, который хочет взять оборудование, должен заполнить необходимую учетную форму. Учетная форма требует указания даты получения оборудования и предположительной даты возврата. Если аппаратура не возвращена к назначенному сроку, то преподавателю, имя которого указано в учетной форме, посылается уведомление. Учебные пособия и диски с данными нельзя выносить из лаборатории: они предназначены только для внутреннего использования".

Ведение платежных ведомостей ассистентов

Лаборатория (UCL) оплачивает труд ассистентов (LA) на почасовой основе и отслеживает общее количество часов, отработанное каждым LA в течение оплачиваемого периода (14 дней). У каждого ассистента имеется рабочий график (дни и часы, когда он должен выходить на работу), и он тоже перед тем, как ему будет выписан чек, должен подписывать табель (в котором указано фактическое количество отработанных им часов). Директор проверяет табель и отправляет его в отдел оплаты для дальнейшей обработки. Дипломированные ассистенты (GA) получают ежемесячную зарплату, работают фиксированное количество часов в неделю и не включаются в эти платежные ведомости.

Резервирование лаборатории для проведения занятий

Преподаватели могут резервировать лабораторию для проведения занятий. Преподаватели заполняют специальную форму на резервирование лаборатории, указывая дату, время, факультет и номер курса данной группы обучения. Если преподаватель резервирует лабораторию для небольшой группы, студенты, не записанные в эту группу, могут по усмотрению преподавателя использовать оставшиеся свободными компьютеры. Здесь могут возникнуть такие вопросы:

Проектировщик:

"Есть ли какие-либо ограничения на то, как часто преподаватель может резервировать лабораторию?"

Конечный пользователь:

"Нет, но все определяется ресурсами лаборатории, быть может, сейчас самое время установить такие ограничения".

Проектировщик:

"За какое время до начала занятий необходимо зарезервировать лабораторию?"

Конечный пользователь:

"Преподаватель должен зарезервировать лабораторию, по крайней мере, за одну календарную неделю до момента начала занятий".

Проектировщик:

"Надо ли указывать продолжительность занятия?"

Конечный пользователь:

"Да".

Резервирование делается только для одной группы; лаборатория используется для проведения занятий только в одной группе за данный период резервирования. Резервирование строится на принципе "первым заявил — первым обслуживается" и утверждается директором.

Проектировщик:

"Имеется ли дневной лимит часов на резервирование лаборатории?"

Конечный пользователь:

"В настоящее время нет каких-либо ограничений на число резервирований в день. Учитывая повышенный спрос студентов на время занятий в лаборатории, особенно в период выполнения проектов в группах, мы должны установить ограничения на резервирование времени. Мы собираемся ограничить доступное время резервирования, выделяя на это один час утром и один час во второй половине дня".

Управление доступом в лабораторию

Лабораторией пользуются студенты, преподаватели и сотрудники. При посещении лаборатории пользователь записывается в журнал, находящийся на стойке при входе, и оставляет ассистенту действующую идентификационную карточку Университета (студенческий билет). Когда пользователь уходит из лаборатории, ассистент проверяет, что все материалы (руководства, диски и т. д.) сданы пользователем, после чего возвращает пользователю студенческий билет, а пользователь расписывается в журнале. Пока лаборатория открыта, нет никаких ограничений на размещение пользователей, за исключением случаев, когда лаборатория зарезервирована для занятий группы.

По мере того как вы начинаете вникать в деятельность лаборатории, необходимо создать информационный журнал, помогающий оценить объем данных, которыми должна оперировать система. В табл. 7.3 приведен пример такого информационного журнала, который показывает виды информации и сколько записей ожидается в заданный промежуток времени.

Таблица 7.3. Пример информационного журнала

Вид информации	Ожидаемое количество записей за период
Ассистенты лаборатории	14 за семестр
Рабочий график	8 часов за рабочий день на одного ассистента
Отработанные часы	1 (общее число часов) запись за оплачиваемый период на одного ассистента
Пользователи	
Преподаватели	300
Студенты	15 000
Сотрудники	650

Таблица 7.3 (окончание)

Вид информации	Ожидаемое количество записей за период
Резервирование	4 в неделю
Ежедневные пользователи лаборатории	570 в день
Заказы	20 в месяц
Заказанные позиции	3 на заказ
Типы инвентаря	15
Местоположение	5
Ремонт	20 в месяц
Поставщики	40

7.1.4. Проблемы и ограничения

После того как мы уяснили для себя деятельность лаборатории, необходимо составить список недостатков имеющейся системы. Подробные опросы основных пользователей, вероятно, позволят выявить имеющиеся проблемы. После того как мы выявим проблемы, необходимо изучить их возможные причины: неверные, неадекватные действия или их отсутствие, недостаток оперативного контроля, неправильное применение имеющихся процедур. Выявление источника проблемы помогает проектировщику выработать правильное решение.

Проблемы могут быть *общими* (общесистемными) или *специфичными* (связанными только с некоторой частью системы). Основные пользователи лаборатории UCL указали на следующие общие недостатки:

- ☐ неавтоматизированная система не отвечает современным требованиям и является постоянным источником ошибок, особенно в складском учете;
- ☐ имеется очень много дублированных и противоречивых данных;
- ☐ неавтоматизированная система не может структурировать информацию должным образом. Очень трудно создавать отчеты (отнимает много времени);
- ☐ система не позволяет создавать нерегламентированные запросы;
- ☐ директор тратит слишком много времени на обработку данных вручную;
- ☐ отсутствие компьютеризованной системы складского учета затрудняет управление информацией. Ошибки в управлении информацией приводят к снижению эффективности управления оборудованием лаборатории.

Необходимо определить сферы деятельности, в которых возникают специфические проблемы. В лаборатории UCL мы выявили проблемы в следующих сферах деятельности.

Управление инвентаризацией/хранением/заказами

- ☐ У директора нет готового доступа к важной информации для управления инвентаризацией: например, какие позиции были заказаны, у какого поставщика они были заказаны, и какие позиции были заказаны, но еще не получены.
- ☐ Лаборатории UCL необходимо знать доступный ассортимент и средние показатели использования расходных материалов, таких как бумага и чернильные картриджи для принтеров для эффективного управления, определения оптимального размера заказа и для своевременного размещения необходимых заказов.
- ☐ Директору не всегда известно действительное местонахождение любого предмета в данное время. Имеющаяся система не позволяет отслеживать инвентарь по категориям, по местоположению или по производителю.

Поддержка оборудования и организация ремонта

- ☐ Директор не может без труда проследить историю обслуживания и ремонта каждой детали оборудования.
- ☐ Директор не может без труда определить статус предмета, обслуживание которого необходимо провести в данный момент.

Контроль выдачи оборудования

- ☐ Директору не хватает своевременной и достоверной информации об имуществе лаборатории: какое оборудование выдано на время, кому, когда и т. д. Невозможно составить отчет о загрузке оборудования.

Ведение платежных ведомостей ассистентов

- ☐ Директор тратит много времени, восстанавливая информацию об общем количестве часов, отработанных каждым ассистентом. Такие сводки нужны при определении графика работ, а также при верстке бюджета лаборатории.
- ☐ Директор не может оценить загрузку студентов. Такая оценка необходима для того, чтобы директор мог более равномерно распределить нагрузку между ассистентами.

Резервирование лаборатории для проведения занятий

- ☐ Система ручного резервирования неудобна: уходит много времени на проверку доступности желаемой даты и времени, на работу с бумагами.
- ☐ Имеющаяся система не предоставляет статистическую информацию, необходимую при составлении расписания загрузки лаборатории.

Управление доступом в лабораторию

- ☐ Журнал пользователей не ведется в надлежащем виде.
- ☐ Некоторые студенты не возвращают взятые материалы. Неверные записи в журнале не дают ассистентам возможности своевременно выявлять эти проблемы. Из-за недостатка контроля лаборатория теряет материалы.

- ☐ Имеются проблемы безопасности, от неавторизованного доступа в сеть и неавторизованной инсталляции/деинсталляции программного обеспечения до исчезновения руководств пользователя. Число таких проблем растет.

Выявив множество проблем, мы пришли к выводу, что имеющаяся ручная система учета не отвечает текущим требованиям. Время, затрачиваемое на работу с документами, постоянно растет и, несмотря на то, что собираются горы информации, она становится все менее доступной. Кроме того, упорядочивание информации требует слишком много времени.

Проблемы решаются в рамках двух наборов ограничений: *ограничения деятельности*, определяемые политикой предприятия, и *экономические ограничения*, определяемые финансами предприятия.

Хорошо спроектированная система базы данных должна решить большинство проблем лаборатории. Следовательно, мы должны тщательно определить ограничения, в рамках которых будет проектироваться данная система.

Ограничение по времени

- ☐ Экономический Колледж предлагает, чтобы новая система была введена в строй в течение трех месяцев.

Аппаратное и программное обеспечение

- ☐ Новая система должна разрабатываться (по возможности) на имеющемся оборудовании и программном обеспечении UCL. Система должна функционировать на имеющейся в UCL локальной сети.

Аспекты распределения и наращивания

- ☐ Новая система должна функционировать в многопользовательской среде.
- ☐ Операции системы не должны зависеть от имеющихся в кампусе административных систем.

Стоимость

- ☐ Стоимость разработки новой системы должна быть минимальной. Чтобы минимизировать затраты и обеспечить дополнительное обучение по направлению "Разработка компьютерных информационных систем" (КИС), система должна разрабатываться под эгидой факультета КИС. Проектирование и реализация будут осуществляться как проект группы под руководством преподавателя для того, чтобы минимизировать стоимость разработки.
- ☐ В новой системе будет использоваться не более двух дополнительных терминалов, которые обеспечат доступ в систему секретарю и директору.
- ☐ Система должна работать с имеющимся числом штатных работников на факультете.
- ☐ В рамках бюджетных ограничений Экономический Колледж выделил \$9500 на неизбежные затраты при разработке новой системы.

7.1.5. Предназначение системы

После выявления проблем и ограничений проектировщик и конечные пользователи кооперируются для обсуждения предназначения новой системы, уделяя основное внимание проблемам, которые основные пользователи полагают наиболее значимыми. Для лаборатории UCL определены два основных направления решаемых задач. К первому направлению относятся *общие задачи*, определяющие общесистемные требования:

- ☐ повышение эффективности и, тем самым, увеличение пропускной способности UCL и расширение сферы деятельности лаборатории;
- ☐ обеспечение необходимой информацией для планирования, управления и безопасности.

Ко второму направлению относятся *специфические задачи*, определяющие требования к компонентам системы и описанные в следующих разделах.

Управление инвентаризацией/хранением/заказами

- ☐ Улучшение контроля над заявками на поставку, что позволит директору отслеживать открытые заявки и покупки.
- ☐ Инвентарный учет.
- ☐ Учет инвентаря по типу (группе), а также по отдельным позициям.
- ☐ Предоставление быстрой и эффективной информации о размещении и статусе каждого предмета.
- ☐ Обеспечение своевременной информацией об использовании материалов и возможность получения статистической информации, необходимой для согласования объема предстоящих закупок.

Управление поддержкой оборудования и организация ремонта

- ☐ Наблюдение за данными о техническом обслуживании каждого объекта.
- ☐ Учет оборудования, возвращенного поставщику для ремонта или замены.

Выдача и учет оборудования

- ☐ Учет материалов, которые берутся сотрудниками на время.
- ☐ Учет времени возврата взятых материалов.
- ☐ Статистический анализ использования материалов и оборудования для получения справочной информации.

Ведение платежных ведомостей ассистентов

- ☐ Разработка графика работ и занятости ассистентов.
- ☐ Сводный отчет о работе каждого ассистента.

Управление резервированием лаборатории для проведения занятий

- ☐ Уменьшение времени на процесс резервирования лаборатории.
- ☐ Составление графика резервирования.
- ☐ Создание сводных статистических отчетов по факультетам, преподавателям, сотрудникам и датам (в целях планирования загрузки лаборатории).

Управление доступом в лабораторию

- ☐ Обеспечение строгого контроля пользователей и ресурсов в лаборатории.
- ☐ Уменьшение времени регистрации пользователей.
- ☐ Предоставление информации о пиковой загрузке лаборатории (для планирования расписания).

7.1.6. Сфера действия и границы

С юридической и практической точек зрения проектировщик базы данных (и, естественно, вся команда разработчиков) не может работать с системой, границы которой не были точно определены, т. е. проектировщик не должен работать с системой, сфера применения которой не определена. Если ограничения системы не определены, проектировщик может на законном основании потребовать расширить систему до неопределенных пределов. Кроме того, структура неограниченной системы не содержит встроенных ограничений, которые и позволяют ее использовать в практических целях в реальной инфраструктуре.

Для определения сферы и границ деятельности лаборатории UCL проектировщик должен постараться ответить на следующие вопросы.

1. *Чем будет ограничена система?* Проект базы данных включает в себя только часть организационной диаграммы UCL, представленной на рис. 7.1. Он не будет зависеть от других систем баз данных, используемых в кампусе.
2. *Какая сфера деятельности охватывается системой?* Система Университетской Компьютерной Лаборатории охватывает шесть сфер деятельности (см. разд. 7.1.3 этой главы) и связана со специфическими задачами, перечисленными в разд. 7.1.5 данной главы. Другими словами, система будет ограничена таким образом, чтобы она могла обслуживать следующие сферы деятельности:
 - управление инвентаризацией/хранением/заказами;
 - управление поддержкой оборудования и организация ремонта;
 - управление выдачей и учетом оборудования;
 - ведение платежных ведомостей ассистентов;
 - управление резервированием лаборатории для проведения занятий;
 - управление доступом в лабораторию.
3. *Какую стратегию проектирования и реализации необходимо использовать для введения системы в действие в рамках времени, отпущенного на разработку?* Чтобы

максимально усилить эффективность проектирования, всю сферу деятельности необходимо подразделить на системные модули. *Модуль* это фрагмент проекта, который можно реализовать как автономную единицу. Модули можно связать в единую систему. Модули играют очень важную роль при проектировании системы, поскольку их использование позволяет осуществить поэтапную реализацию и тестирование системы.

4. *Какие модули должны быть включены в систему?* Сферы деятельности, упомянутые во втором вопросе данного списка, можно разделить на два основных направления: управление лабораторией и управление материально-техническим снабжением. Поэтому имеет смысл разрабатывать два модуля, представленные в табл. 7.4. Обратите внимание, что каждый модуль состоит из именованных процессов. Например, в состав модуля "Система Управления Лабораторией" входят процессы ACCESS (доступ), RESERVATIONS (резервирование) и PERSONNEL (персонал).

Таблица 7.4. Модули системы Университетской Компьютерной Лаборатории (UCL)

Модуль	Сфера деятельности	Имя процесса
Система Управления Лабораторией (Lab)	Доступ в компьютерную лабораторию	ACCESS (доступ)
	Резервирование	RESERVATIONS (резервирование)
	Ведение платежных ведомостей ассистентов	PERSONNEL (персонал)
Система Управления Материально-Техническим Снабжением (Inventory)	Инвентаризация	INVENTORY (инвентаризация)
	Заказ	ORDER (заказ)
	Хранение	STORAGE (хранение)
	Обслуживание и ремонт оборудования	MAINTENANCE (обслуживание)
	Выдача и учет оборудования	CHECK_OUT (контроль выдачи)

5. *Как взаимодействуют модули?* Процесс INVENTORY (инвентаризация) модуля "Система Управления Материально-Техническим Снабжением" (Inventory) является ключевым компонентом системы. Его наличие расширяет возможности директора по наблюдению за деятельностью лаборатории и осуществлению административных функций. На рис. 7.2 показано, как модуль "Система Управления Материально-Техническим Снабжением" (Inventory) взаимодействует с модулем "Система Управления Лабораторией" (Lab) с помощью процесса CHECK_OUT (контроль выдачи).

Хотя процесс INVENTORY (инвентаризация) не будет зависеть от других (специфических) систем инвентаризации, используемых в кампусе, в нем будет использоваться схема классификации инвентаря отдела снабжения всего Университета.

Использование этой классификации упрощает взаимодействие пользователя с отделом снабжения. Кроме того, она позволит в будущем упростить возможную интеграцию с общеуниверситетской системой управления материально-техническим снабжением.

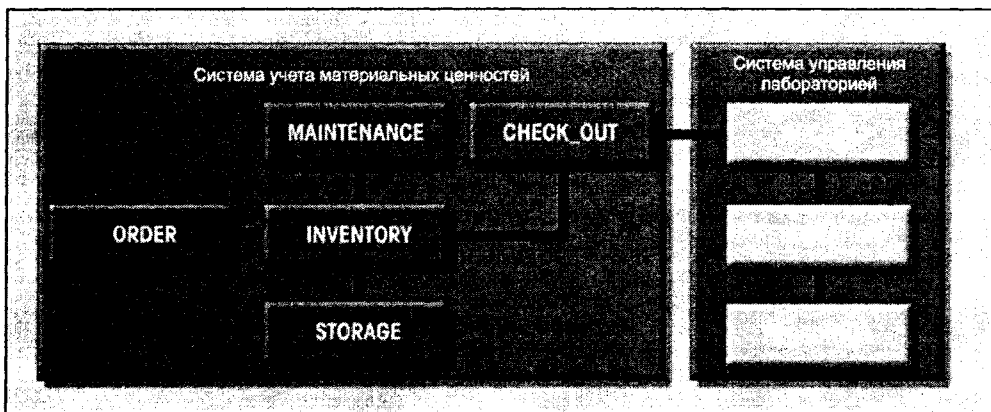


Рис. 7.2. Схема управления Университетской Компьютерной Лабораторией (UCL)

Процесс INVENTORY (инвентаризация) должен выполнять следующие функции:

- ☐ регистрация нового типа материальных средств и отдельных предметов;
- ☐ отслеживание местоположения, классификация и использование материально-технических средств.

Процесс INVENTORY (инвентаризация) будет взаимодействовать с процессами ORDER (заказ), STORAGE (хранение), MAINTENANCE (обслуживание) и CHECK_OUT (контроль выдачи).

Процесс ORDER (заказ) отслеживает типы материально-технических средств, которые заказываются у поставщиков. Система будет спроектирована для отслеживания заявок на покупку и требований, оформленных в UCL. Процесс ORDER (заказ) будет взаимодействовать с процессом INVENTORY (инвентаризация).

Процесс MAINTENANCE (обслуживание) будет отслеживать организацию ремонта объектов своими силами, а также оборудование, возвращаемое поставщику для ремонта. Процесс MAINTENANCE (обслуживание) также взаимодействует с процессом INVENTORY (инвентаризация), поскольку оборудование, имеющееся в инвентарной ведомости, может иметь предысторию ремонта.

Процесс CHECK_OUT (контроль выдачи) отслеживает материалы и оборудование, взятые на время пользователями лаборатории: преподавателями, сотрудниками или студентами.

С помощью процесса ACCESS (доступ) директор может отслеживать пользователей лаборатории. Этот процесс будет взаимодействовать с процессом CHECK_OUT (контроль выдачи), поскольку студенты, преподаватели и сотрудники берут на время материалы и оборудование.

Процесс RESERVATIONS (резервирование) следит за резервированием лаборатории, выполняемым преподавателями или сотрудниками. Этот процесс взаимодействует с процессами:

- ☐ ACCESS (доступ) (поскольку лабораторию резервируют преподаватели);
- ☐ PERSONNEL (персонал) (поскольку запись о резервировании выполняется ассистентом).

Процесс PERSONNEL (персонал) поможет директору наблюдать за расписанием работы ассистентов и количеством отработанных ими часов. Этот процесс взаимодействует с процессом RESERVATIONS (резервирование), поскольку записи о резервировании лаборатории выполняются ассистентом.

7.2. Этап проектирования базы данных: концептуальный проект

Чтобы разработать хороший концептуальный проект, мы должны собрать информацию, позволяющую точно определить сущности и описать их атрибуты и связи. Связи сущностей должны точно отражать реальные взаимосвязи.

7.2.1. Источники информации и пользователи

На стадии начальной разработки БД большую часть полезной информации получают при опросе основных пользователей системы. Мы должны начать этап концептуального проектирования с выявления надежных источников информации. При этом мы еще раз проверяем основных пользователей и составляем подробный список имеющихся и потенциальных пользователей системы. Кроме того, в процессе выяснения источников информации определяется документооборот имеющейся системы, включая данные и отчетные формы. На этом этапе нельзя пренебрегать ни одним документом. Если бумага существует, значит, кто-то полагает, что она необходима. Для UCL мы должны выявить следующие источники информации:

- ☐ помощника декана;
- ☐ директора лаборатории (CLD);
- ☐ секретаря лаборатории (CLS);
- ☐ ассистентов лаборатории (LA) и дипломированных ассистентов (GA);
- ☐ студентов, преподавателей и сотрудников, которые пользуются ресурсами лаборатории;
- ☐ используемые в настоящее время компьютерные формы документов, папки с файлами и отчетные формы.

Неудивительно, что список предполагаемых пользователей системы дублирует, по крайней мере, часть списка информационных ресурсов:

- ☐ CLD (который является и системным администратором) управляет системой, вводит данные в БД и определяет требования к отчетности;

- LA и GA являются основными пользователями UCL и также заносят данные в БД;
- CLS является системным пользователем, который не только выполняет запросы к базе данных, но и обновляет БД.

Мы полагаем, что необходимо создать сводную таблицу для определения всех источников информации системы и ее пользователей. Таковую таблицу можно использовать для перекрестной проверки, что упростит проверку источников информации и пользователей. Сводные данные по источникам информации UCL представлены в табл. 7.5. Обратите внимание, что в сводной таблице также определены предложенные системные модули, процессы и взаимодействия, упомянутые в предыдущих разделах данной главы.

Таблица 7.5. Источники информации и пользователи

Модуль	Процесс	Источник	Пользователи	Взаимодействие
Система Учета Материально-Технического Снабжения	INVENTORY (инвентаризация)			
	Данные по инвентарю	Инвентарные формы, CLD	CLD, CLS, декан*	ORDER
	Данные по объекту	Инвентарные формы	CLD, CLS, декан	MAINTENANCE
	Выдача под отчет	Инвентарные формы	CLD, CLS, декан	CHECK_OUT
	Ремонт	Журнал неисправности оборудования	CLD, CLS, декан	MAINTENANCE
	Выдача под отчет	Формы выдачи под отчет	CLD, CLS, декан	CHECK_OUT
	Местоположение	Инвентарные формы	CLD, CLS, декан	STORAGE
	ORDER (заказ)			
	Данные заказа	Формы заказа	CLD, CLS, декан	INVENTORY
	Заказанные позиции	Формы заказа	CLD, CLS, декан	INVENTORY
	Полученные материалы	Формы заказа, инвентарные описи	CLD, CLS, декан	INVENTORY
	Тип инвентаря	Инвентарные формы, CLD	CLD, CLS, декан	INVENTORY

Таблица 7.5 (продолжение)

Модуль	Процесс	Источник	Пользователи	Взаимодействие
Система Управления Лабораторией	Поставщик	Формы заказа	CLD, CLS, декан	INVENTORY
	STORAGE (хранение)			
	Местоположение	Инвентарные формы	CLD, CLS	INVENTORY
	Данные по материалам и оборудованию	Инвентарные формы	CLD, CLS	INVENTORY
	MAINTENANCE (обслуживание)			
	Ремонт	Журнал неисправности оборудования	CLD, CLS, GA	
	Данные по материалам и оборудованию	Инвентарные формы	CLD, CLS, GA	INVENTORY
	Данные по поставщику	Инвентарные формы	CLD, CLS, GA	
	CHECK_OUT (контроль выдачи)			
	Данные по материалам и оборудованию	Инвентарные формы	CLD, CLS, GA	INVENTORY
	Пользователи	Журналы выдачи под отчет	CLD, CLS, GA	ACCESS
	ACCESS (доступ)			
	Пользователь	Журнал посещения лаборатории	CLD, LA	RESERVATION, CHECK_OUT
	RESERVATIONS (резервирование)			
	Данные по резервированию	Формы заказа на резервирование	CLD, CLS, LA	ACCESS

Таблица 7.5 (окончание)

Модуль	Процесс	Источник	Пользователи	Взаимодействие
	PERSONNEL (персонал)			
	Ассистенты	Форма учета ассистентов	CLD, CLS	
	График работы	Форма графика работы	CLD, CLS, LA	
	Отработанные часы	Табель	CLD, CLS, LA	

Примечания. CLD — директор; CLS — секретарь; LA — ассистент; GA — дипломированный ассистент.

*Хотя декан и не является активным пользователем системы, он использует системные отчеты для принятия решений.

7.2.2. Необходимая информация: требования пользователя

Проект должен удовлетворять всем существенным требованиям пользователя. Существенные требования базируются на предлагаемом уровне эффективности получения информации. Сводка всех существенных требований пользователей UCL определяет список основных требований к системе.

- ☐ Система должна быть проста в использовании. Интерфейс основного меню должен быть очень удобным.
- ☐ Система должна обеспечивать безопасность. Это осуществляется с помощью паролей и прав доступа.
- ☐ Система должна быть полностью интегрирована, т. е. должны быть исключены избыточность данных и избыточные обновления. Система должна гарантировать целостность БД.
- ☐ Пользователи должны иметь возможность получать доступ к БД одновременно с нескольких рабочих станций. Местоположение рабочих станций должно соответствовать установкам, представленным в табл. 7.6. На рис. 7.3 представлена схема Системы Управления Университетской Компьютерной Лабораторией (University Computer Lab Management System, UCLMS).

Таблица 7.6. Распределение рабочих станций: предназначение и пользователи

Пользователь	Используемые процессы	Предназначение	Идентификатор станции (ID)
Директор UCL (CLD)	Все	Системное администрирование	WS3

Таблица 7.6 (окончание)

Пользователь	Используемые процессы	Предназначение	Идентификатор станции (ID)
Секретарь UCL (CLS)	INVENTORY (инвентаризация) ORDER (заказ) STORAGE (хранение) MAINTENANCE (обслуживание) CHECK_OUT (контроль выдачи) RESERVATIONS (резервирование) PERSONNEL (персонал)	Обновления и запросы	WS4
Ассистент (LA) и дипломированный ассистент (GA)	ACCESS (доступ) RESERVATIONS (резервирование) CHECK_OUT (контроль выдачи) MAINTENANCE (обслуживание) PERSONNEL (персонал)	Обновления и запросы	WS1, WS2

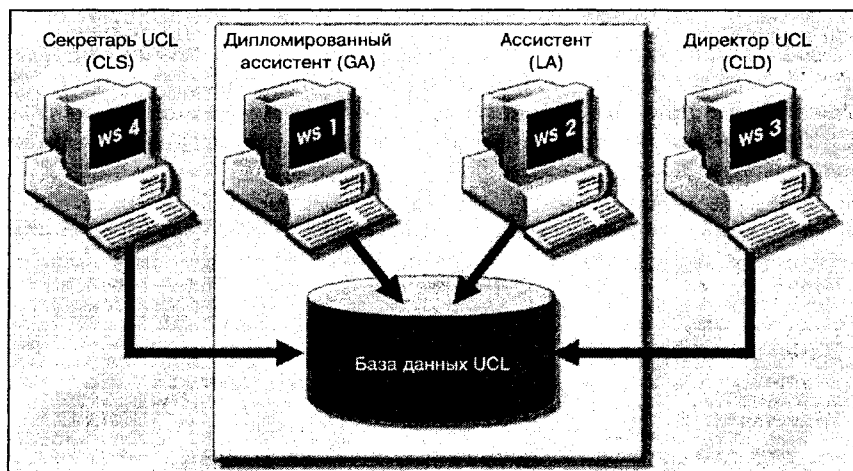


Рис. 7.3. Система управления Университетской Компьютерной Лабораторией: общая схема

□ Процессы системы должны выполнять следующие функции:

- процесс *PERSONNEL* (персонал) — работа с данными по всем ассистентам, их распорядку работы и отработанным часам;
- процессы *INVENTORY* (инвентаризация) и *STORAGE* (хранение) — контролируют запас материалов по местоположению и по типу инвентаря. Система также должна отслеживать расходные материалы, фиксируя их потребление (получение). Система должна отслеживать малоценный инвентарь и объекты строгой отчетности;
- процесс *ORDER* (заказ) — интегрируется с модулем *INVENTORY* (инвентаризация) для установления связи между заказами и типом оборудования. Система должна выдавать информацию о размещении заказов и общую стоимость заказов по поставщикам, по объектам и по типу оборудования. Система также должна выдавать суммарную стоимость заказов для формирования бюджета;
- процесс *MAINTENANCE* (обслуживание) — отслеживает предысторию обслуживания всего оборудования. Этот процесс должен вести учет оборудования, отосланного поставщикам для замены или обслуживания;
- процесс *RESERVATIONS* (резервирование) — позволяет директору составлять график резервирования лаборатории. Система должна автоматически показывать график загрузки на указанный день и производить резервирование в соответствии с политикой факультета и/или лаборатории;
- процесс *ACCESS* (доступ) — следит за уровнем использования UCL. Система позволяет ассистентам регистрировать студентов, пользующихся услугами лаборатории. Система получает идентификационный номер пользователя (ID) с помощью устройства считывания штрихкода, установленного на компьютере при входе в лабораторию. Система должна позволять студентам получать диски с учебными материалами и руководством по использованию программного обеспечения.

- Входные требования системы в некотором смысле определяются требованиями к ее выходным данным, т. е. требованиями к планируемым запросам и к возможностям создания отчетов. Отчеты, которые требуются UCL, представлены в табл. 7.7. Требования к отчетности помогают определить необходимые атрибуты данной сущности. Определение точной формы отчета является важной частью процесса концептуального проектирования.

Таблица 7.7. Отчеты UCLMS

№ п/п	Отчет	Описание	Пользователи
1	Движение инвентаря	Движение инвентаря по дате и по типу	CLD, CLS
2	Инвентарная опись	Инвентарная опись по типу	CLD, CLS
3	Местоположение инвентаря	Инвентарный список объектов по местоположению	CLD, CLS

Таблица 7.7 (окончание)

№ п/п	Отчет	Описание	Пользователи
4	Заказы	Заказы по дате, поставщику и статусу	Декан, CLD
5	Открытые заказы	Открытые заказы по дате и поставщику	Декан, CLD
6	Заказы к оплате	Заказы полученные, но не оплаченные	Декан, CLD
7	Оплаченные заказы	Оплаченные заказы по дате и поставщику	Декан, CLD
8	Обслуживание	Предыстория обслуживания по дате и оборудованию	CLD, GA, LA
9	Выдача	Выдача материалов и оборудования по дате и пользователю	CLD, CLS
10	График работы ассистентов	График работы ассистентов лаборатории	CLD, CLS, GA, LA
11	Отработанные часы ассистентов	Часы, отработанные ассистентом лаборатории	CLD, CLS, GA, LA
12	График резервирования	Резервирование лаборатории по дате и пользователям	CLD, CLS, LA
13	Статистика использования UCL	Статистика использования компьютерной лаборатории	Декан, CLD, зав. кафедрами

7.2.3. Разработка первоначальной ER-модели

Во время начальной стадии разработки БД и концептуального проектирования мы определяем первоначальный набор сущностей. Эти сущности представляют собой наиболее важную информацию об объектах системы с точки зрения представления конечных пользователей и проектировщика. Некоторые сущности являются объектами реального мира, например, пользователи, ассистенты, предметы, местоположение или поставщики. Другие представляют собой информацию о сущностях, например, график работы, отработанные часы, ремонт, журнал занятости лаборатории или резервирование. Лаборатория UCL будет использовать сущности, представленные в табл. 7.8.

Таблица 7.8. Сущности UCL, определенные на основе начального этапа разработки

Имя сущности	Описание сущности	Тип сущности
USER	Данные пользователя: включая администрацию, преподавателей и студентов	
LAB_ASSISTANT	Данные по ассистентам: включая дипломированных ассистентов	

Таблица 7.8 (окончание)

Имя сущности	Описание сущности	Тип сущности
WORK_SCHEDULE	График работы ассистентов: время работа каждого ассистента лаборатории	
HOURS_WORKED	Данные по отработанным часам ассистентов: фактическое количество отработанных часов за каждый оплачиваемый период для каждого ассистента	Слабая
LOG	Ежедневные пользователи лаборатории: одна запись для каждого посетителя	
RESERVATION	Подробности резервирования лаборатории	
INV_TYPE	Типы инвентаря	
ITEM	Сведения об инвентаре	
LOCATION	Место хранения	
REPAIR	Дата ремонта по элементам оборудования	
VENDOR	Сведения о поставщиках	
ORDER	Сведения о заказах	

Проектировщик и конечный пользователь при определении сущностей должны прийти к соглашению. Проектировщик затем определяет связи между сущностями, в основном опираясь на описание операций (см. разд. 7.1.3). Точнее, связи сущностей базируются на бизнес-правилах, построенных на основе подробного описания операций.

Бизнес-правила должны быть определены и проверены. Проектировщик базы данных UCL проводит серию опросов основных пользователей системы. В данном случае это директор лаборатории UCL, который отвечает за обеспечение деятельности лаборатории, и помощник декана факультета Компьютерных Информационных Систем (Computer Information Systems, CIS), который осуществляет общее администрирование системы. После определения необходимых бизнес-правил и включения их в ER-модель проектировщик представляет модель этим же персонам, чтобы убедиться, что она в точности соответствует фактическим или предполагаемым операциям. Проектировщик также представляет ER-модель конечным пользователям для того, чтобы убедиться, что они точно описали свои действия и функции. Процесс проверки может привести к появлению дополнительных сущностей и связей.

В процессе ER-моделирования UCL был получен следующий набор бизнес-правил, сущностей и связей.

Бизнес-правило 1

Каждый предмет имеет один и только один инвентарный тип, а каждому инвентарному типу может принадлежать от нуля до нескольких предметов.

Для пояснения этого бизнес-правила мы приведем один пример, представленный в табл. 7.9. Обратите внимание, что тип инвентаря представляет собой классификатор,

в который включены все предметы внутри данной категории. Например, и Dell Dimension, и Toshiba представляют собой персональные компьютеры.

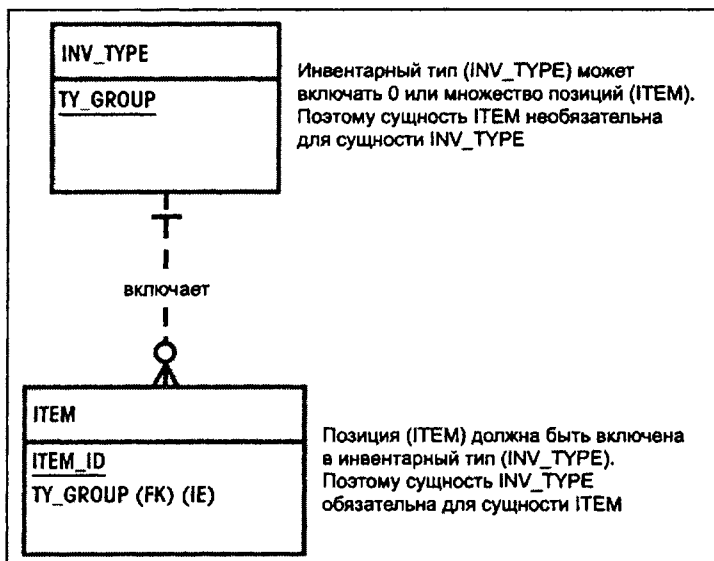


Рис. 7.4. Сегмент ER-модели для бизнес-правила 1

Таблица 7.9. Пример классификации инвентаря по типам

Тип инвентаря				Предмет	
Категория	Класс	Тип	Подтип	ID предмета (инв. №)	Описание
Оборудование	Персональный компьютер	Настольный	Pentium	3233452	Dell Dimension, память 128 Мбайт, жесткий диск 20 Гбайт, диск-вод Zip 100 Мбайт
Оборудование	Персональный компьютер	Ноутбук	Pentium	3312455	Toshiba, память 128 Мбайт, жесткий диск 16 Гбайт
Оборудование	Принтер	Лазерный		3122146	HP LaserJet IV
Материалы	Бумага	Для лазерного принтера	8,5 × 11		Бумага для лазерного принтера
Материалы	Диск	3,5"	HD		Диск 3,5" повышенной плотности

Таблица 7.9 (окончание)

Тип инвентаря				Предмет	
Категория	Класс	Тип	Подтип	ID предмета (инв. №)	Описание
Материалы	Картридж	Для струй- ного прин- тера	Цветной		Цветной картридж для струйного принтера

Необходимо подчеркнуть, что отдельный предмет принадлежит только одному инвентарному типу.

На основе первого бизнес-правила мы получаем сегмент ER-модели, представленный на рис. 7.4.

Бизнес-правило 2

Инвентарь можно сразу использовать после его получения или поместить на склад. Другими словами, инвентарь не обязательно должен храниться на складе. Некоторые предметы, такие как чернильные картриджи, входящие в группу, могут храниться более чем в одном месте. Следовательно, некоторые предметы могут не храниться ни в одном (0) или храниться в одном или более местах. В каждом месте хранения может храниться 0, один или более предметов (см. рис. 7.5).

Бизнес-правило 3

Заказ адресован только одному поставщику, и каждый поставщик может получить 0, один или несколько заказов (рис. 7.6).

Бизнес-правило 4

В каждом заказе содержится один или несколько заказываемых предметов, и каждая строка заказа предмета принадлежит только одному заказу (рис. 7.7).

Бизнес-правило 5

Каждая строка в заказе соответствует одному инвентарному типу, а каждый инвентарный тип может встречаться в одной или более строках заказа (рис. 7.8).

Пример:

Заказанный предмет:		Компьютер Pentium	
Тип инвентаря	Категория:	Оборудование	
	Класс:	Компьютер	
	Тип:	Настольный	
	Подтип:	Pentium	
Предмет	3233452	Серийный номер	
		Gateway 2000 Pentium PIII-933	

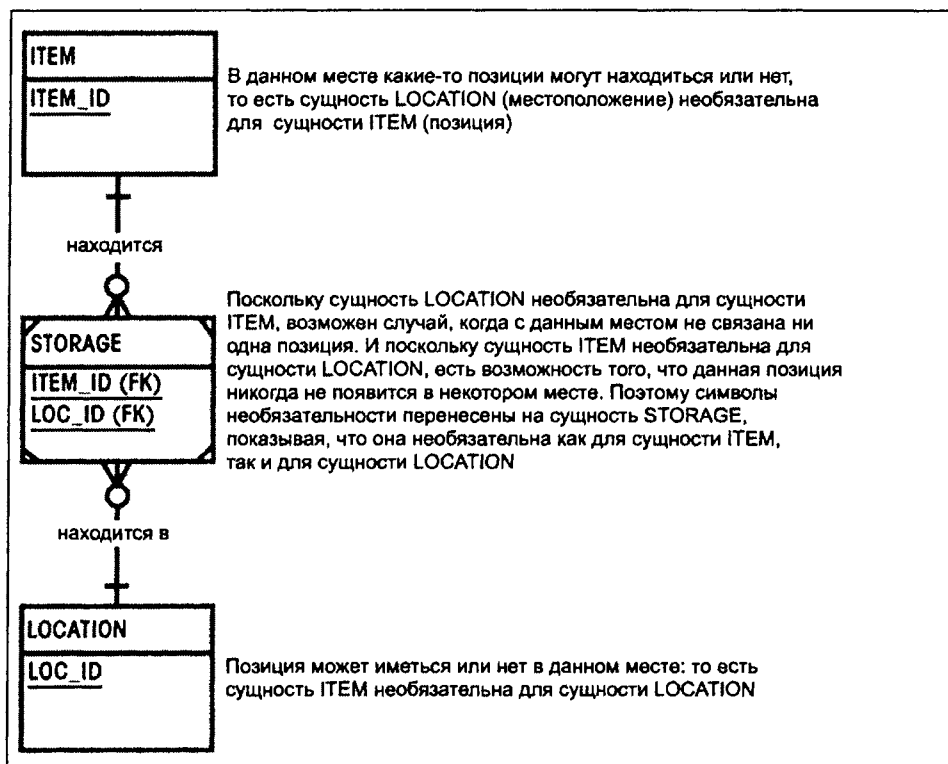


Рис. 7.5. Сегмент ER-модели для бизнес-правила 2

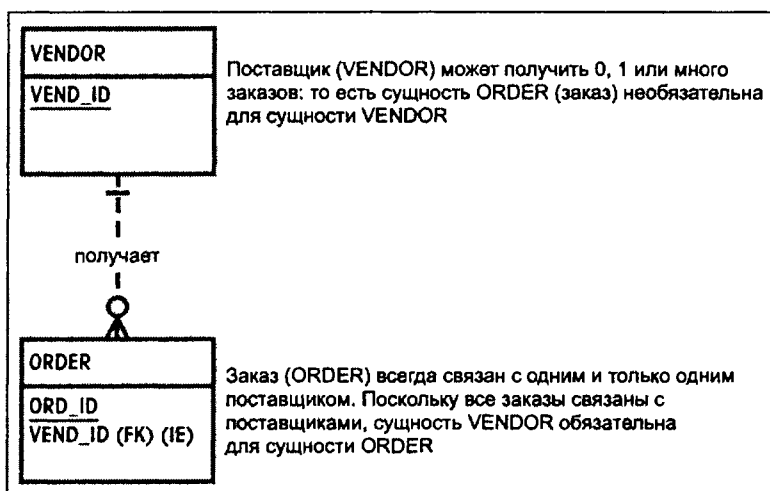


Рис. 7.6. Сегмент ER-модели для бизнес-правила 3

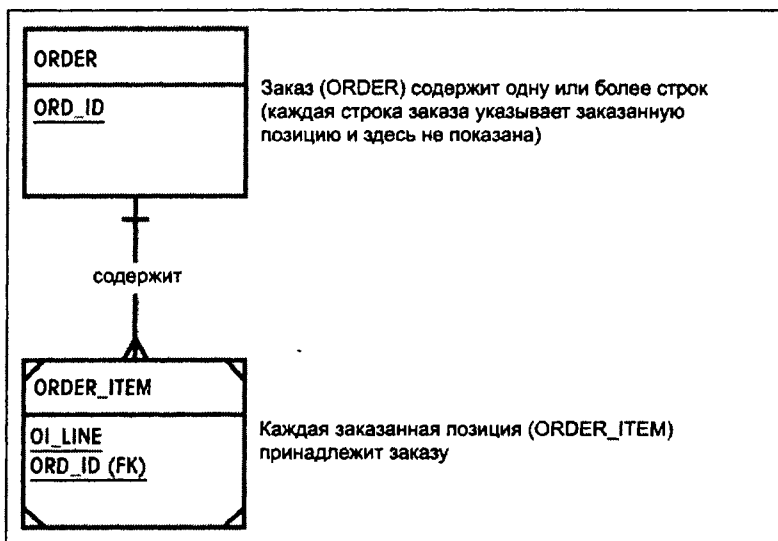


Рис. 7.7. Сегмент ER-модели
для бизнес-правила 4

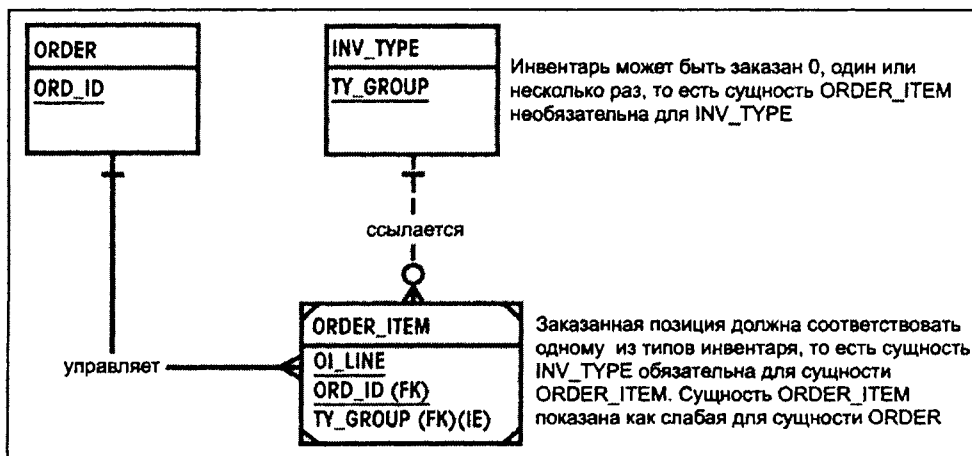


Рис. 7.8. Сегмент ER-модели
для бизнес-правила 5

Бизнес-правило 6

Каждый предмет может ремонтироваться 0, один или более раз, и каждая запись о ремонте соответствует только одному предмету (рис. 7.9).

Бизнес-правило 8

Каждый пользователь может взять на время 0, один или несколько предметов. Каждый предмет может быть выдан в течение семестра 0 (никому), одному или нескольким пользователям (рис. 7.11).

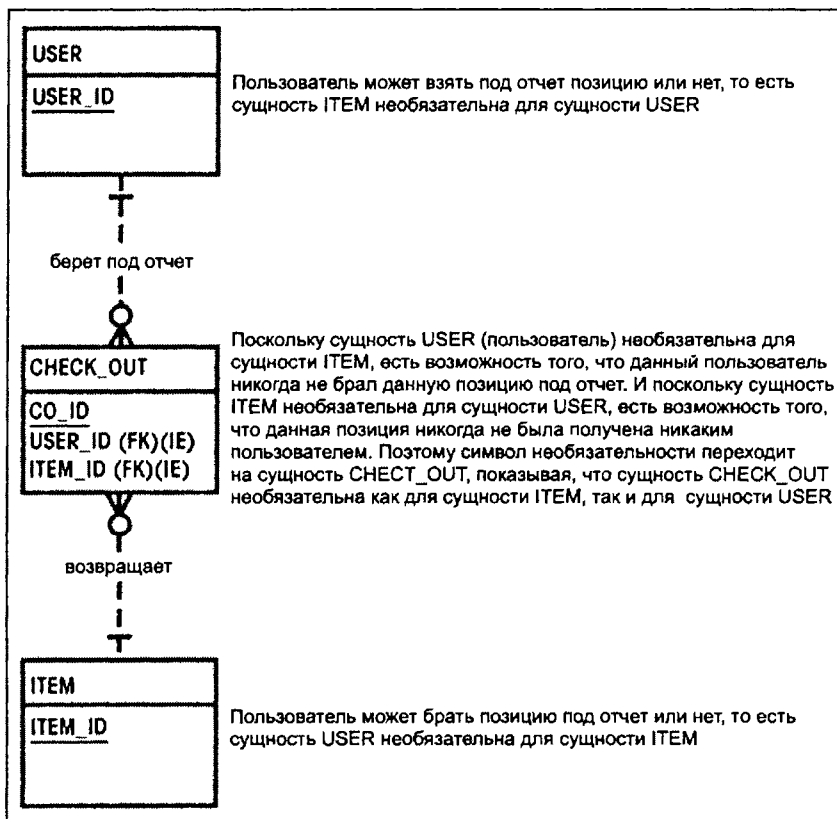


Рис. 7.11. Сегмент ER-модели для бизнес-правила 8

Бизнес-правило 9

Каждый пользователь (преподаватель или сотрудник) может взять в пользования один или более предметов, а каждый предмет может быть взят в течение семестра, одним или более пользователями (рис. 7.12).

Бизнес-правило 10

Каждый пользователь (студент) может зарегистрироваться в журнале пользования несколько раз в течение семестра, и каждая запись делается только для одного пользователя (рис. 7.13).

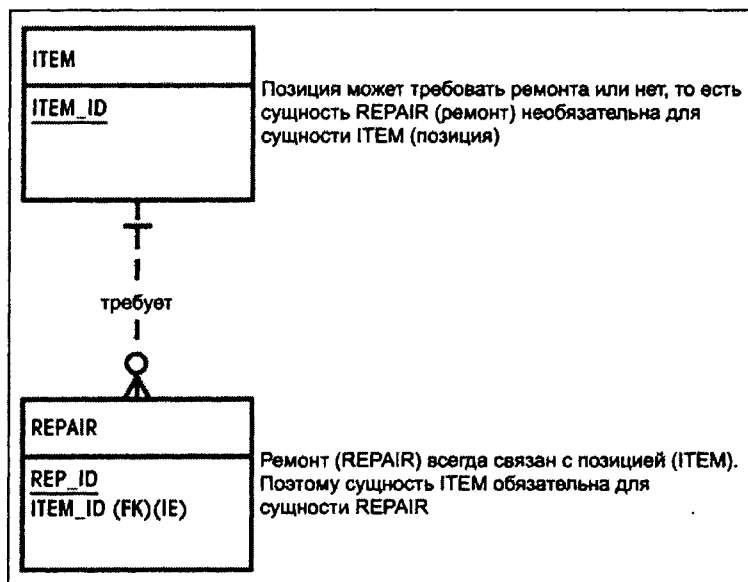


Рис. 7.9. Сегмент ER-модели для бизнес-правила 6

Бизнес-правило 7

Каждый предмет, который необходимо ремонтировать, может быть возвращен поставщику или нет (иногда оборудование ремонтируется в лаборатории), а у каждого поставщика может находиться 0 или несколько возвращенных на ремонт изделий (рис. 7.10).

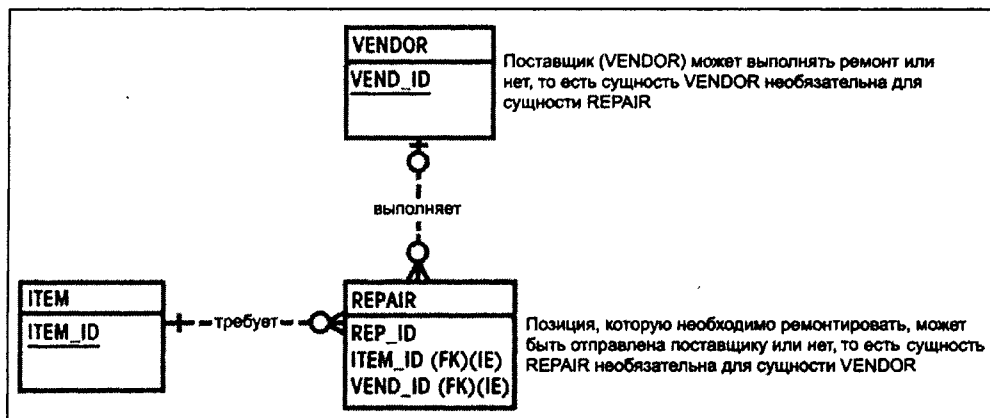


Рис. 7.10. Сегмент ER-модели для бизнес-правила 7

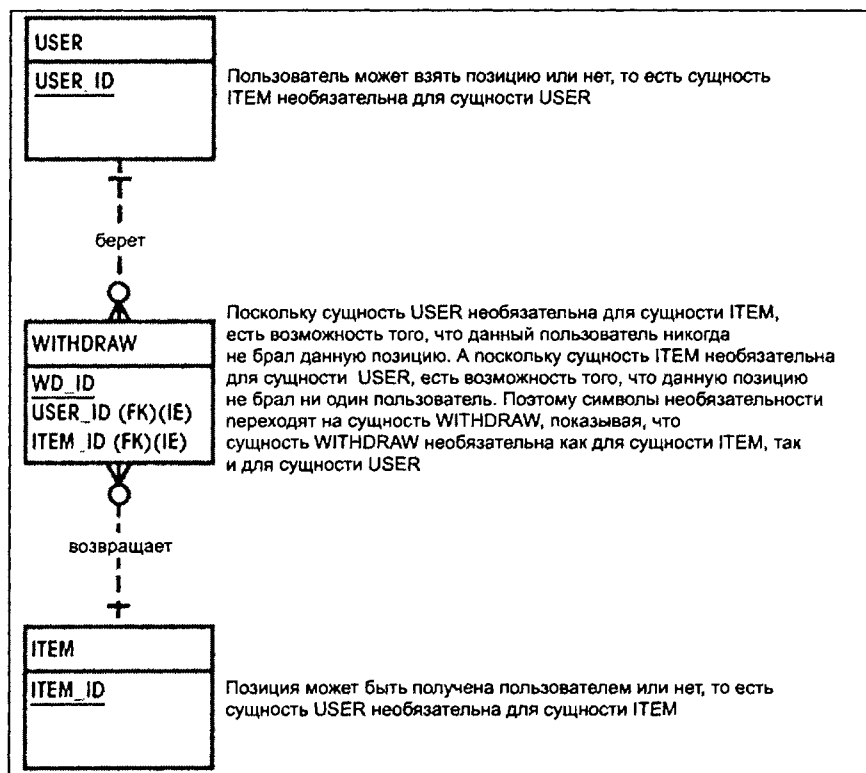


Рис. 7.12. Сегмент ER-модели для бизнес-правила 9

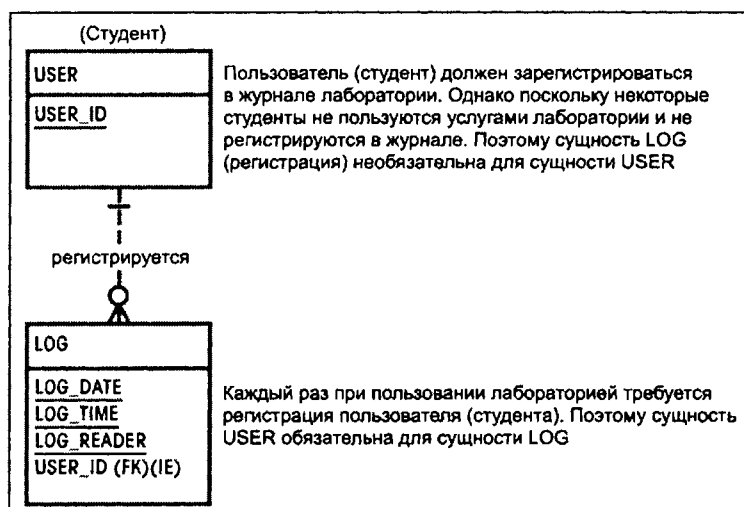


Рис. 7.13. Сегмент ER-модели для бизнес-правила 10

Бизнес-правило 11

Каждый пользователь (преподаватель) может сделать 0, одно или более резервирований лаборатории в течение семестра, и каждое резервирование делается одним преподавателем (рис. 7.14).

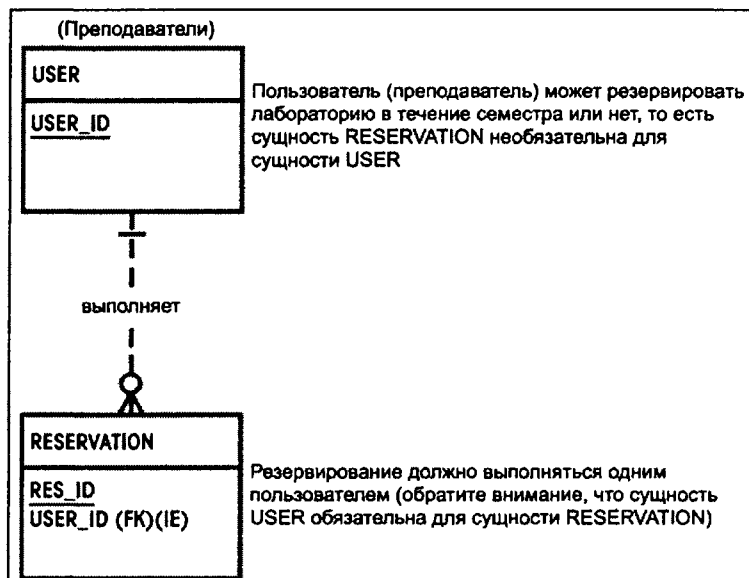


Рис. 7.14. Сегмент ER-модели для бизнес-правила 11

Бизнес-правило 12

Каждое резервирование лаборатории выполняется ассистентом, и каждый ассистент может сделать 0, одно или несколько резервирований в течение семестра (рис. 7.15).

Бизнес-правило 13

Каждый ассистент назначается на дежурство по лаборатории, по крайней мере, один раз в неделю в соответствии с графиком работ, и каждое назначение делается для одного ассистента (рис. 7.16).

Бизнес-правило 14

Каждый ассистент накапливает отработанные часы в течение каждого оплачиваемого двухнедельного периода, и каждая запись об отработанных часах связана только с одним ассистентом (рис. 7.17).

Бизнес-правило 15

Каждый предмет поставляется определенным поставщиком, и каждый поставщик может поставлять несколько различных предметов (рис. 7.18).

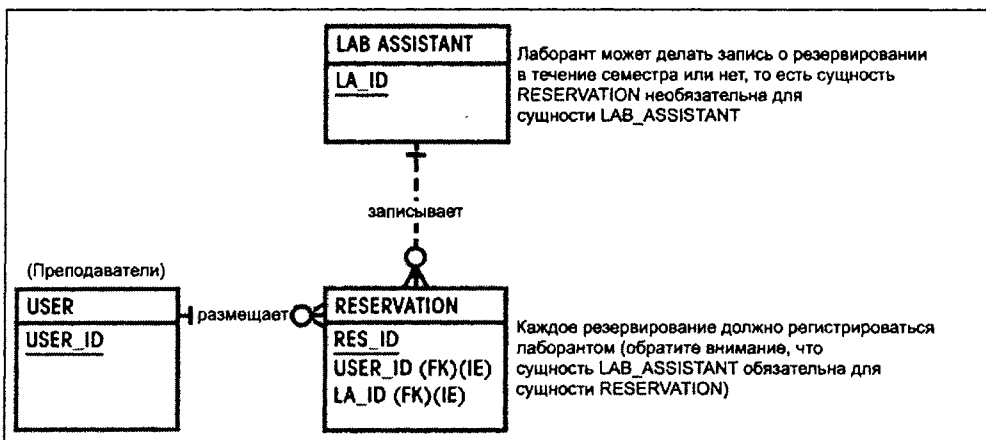


Рис. 7.15. Сегмент ER-модели для бизнес-правила 12

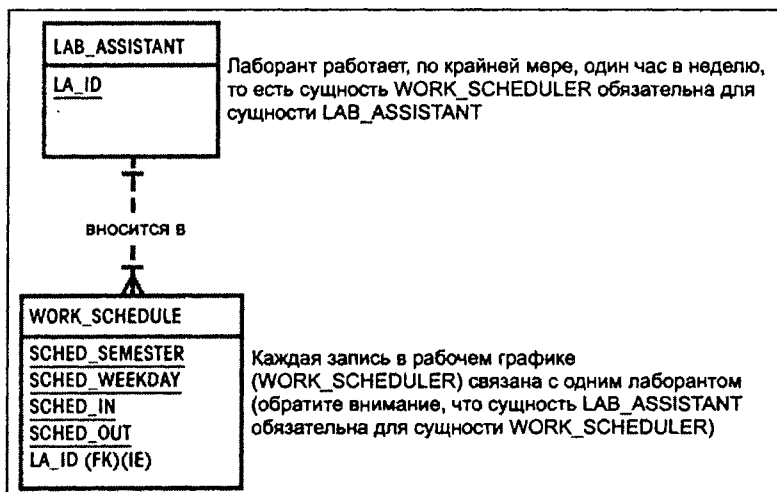


Рис. 7.16. Сегмент ER-модели для бизнес-правила 13

Примечание для проектировщика

Помните, что студент может взять материалы на время только во время его нахождения в лаборатории. Если такое ограничение записано в виде бизнес-правила, то его можно представить на ER-диаграмме; в противном случае оно должно быть отражено в программном коде, отражающем операции UCL.

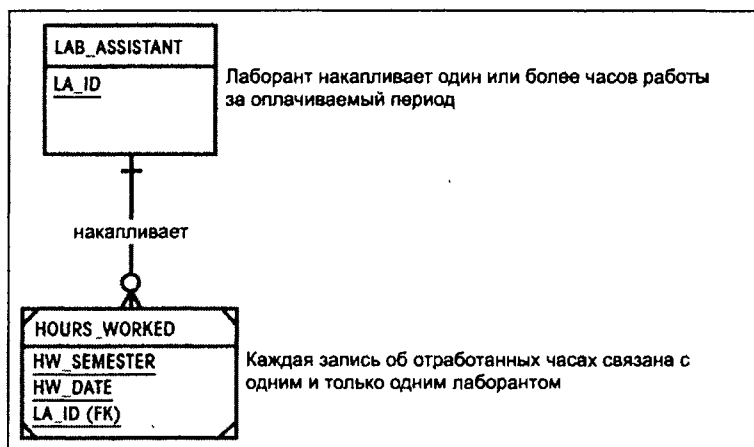


Рис. 7.17. Сегмент ER-модели для бизнес-правила 14

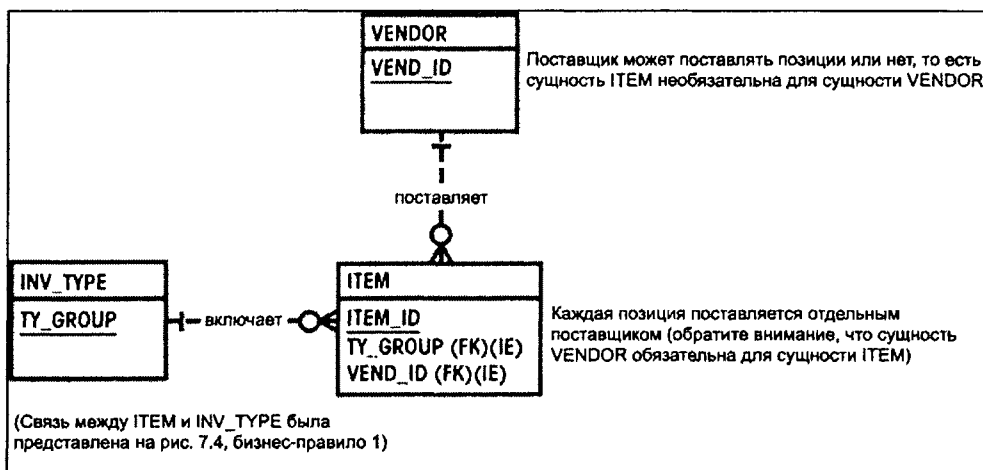


Рис. 7.18. Сегмент ER-модели для бизнес-правила 15

Хотя табл. 7.8 и 7.10 содержат схожую информацию, они отражают разные этапы процесса ER-моделирования. В табл. 7.8 представлена начальная информация о сущностях, полученная из описания операций UCL. Это описание — источник бизнес-правил лаборатории. Бизнес-правила часто порождают вопросы, которые могут стать причиной появления новых сущностей, связей и атрибутов. Кроме того, поскольку сущности и связи образуют сегменты ER-диаграммы, в процессе моделирования может возникнуть необходимость в дополнительных сущностях и/или отношениях. Информация о сущностях, представленная в табл. 7.10, отражает результаты такого динамического процесса моделирования.

В табл. 7.10 представлены предлагаемые сущности системы управления UCL. ER-компоненты, которые мы идентифицировали к настоящему времени, объединяются в ER-диаграмму. На рис. 7.19 представлена база данных, как она видится конечному пользователю и проектировщику к настоящему моменту времени.

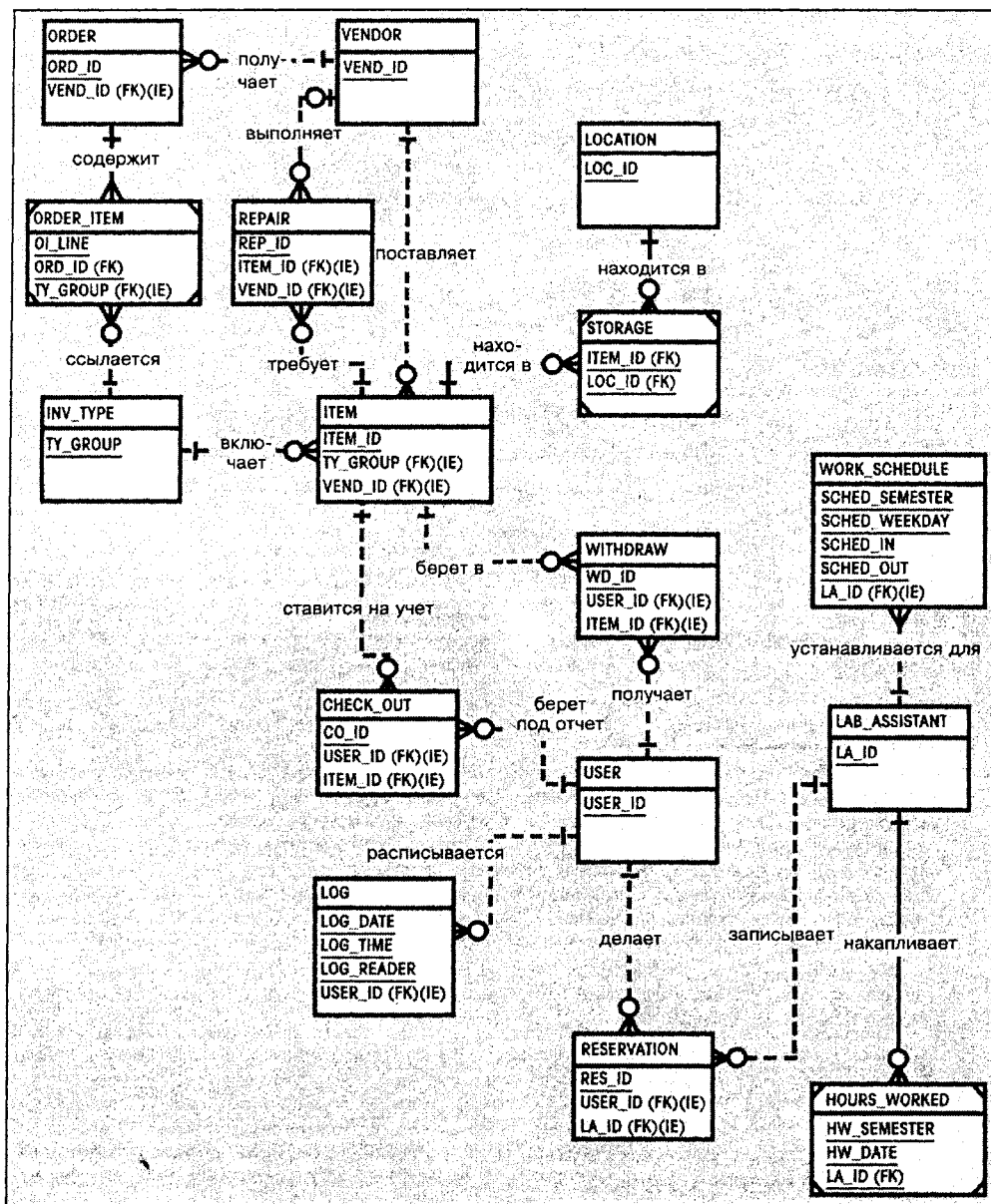


Рис. 7.19. Начальная ER-диаграмма системы управления лабораторией UCL

Таблица 7.10. Сущности UCL на основе бизнес-правил

Имя сущности	Описание сущности	Тип сущности
USER	Данные о пользователе	
LAB_ASSISTANT	Данные об ассистенте	
WORK_SCHEDULE	Рабочий график ассистента	
HOURS_WORKED	Отработанные ассистентом часы	Слабая
LOG	Регистрация ежедневных пользователей лаборатории	
RESERVATION	Данные о резервировании лаборатории	
INV_TYPE	Инвентарный тип	
ITEM	Данные об объекте	
CHECK_OUT	Выдача под отчет	
WITHDRAW	Выдача	
LOCATION	Место хранения объекта	
STORAGE	Объекты, хранящиеся в данном месте	Составная
REPAIR	Данные о ремонте	
VENDOR	Данные о поставщике	
ORDER	Данные о заказе	
ORDER_ITEM	Данные о заказанных предметах	Слабая

Примечание

Бизнес-правила создаются на основе многих источников, таких как конечные пользователи, формы, руководства и т. д. Поэтому их нельзя создавать в каком-то раз и навсегда установленном порядке. Например, мы начали обзор бизнес-правил этой главы, определив правило для учета инвентаря, затем мы перешли к конечным пользователям и их работе в лаборатории, а затем опять вернулись к бизнес-правилам учета инвентаря. Все эти бизнес-правила затем были преобразованы в ER-сегменты, которые были помещены в структуру, представленную на рис. 7.19. При необходимости вы можете в любом месте ER-диаграммы сформировать бизнес-правила, соответствующие отслеживаемой вами части проекта. Или можно сгруппировать бизнес-правила по некоторым процессам. Хотя такая перестройка бизнес-правил может соответствовать вашему пониманию организации, в ней нет необходимости.

Также помните, что различные конечные пользователи имеют представление о связях данных на разных уровнях. Например, обратите внимание, что связь M:N между сущностями ITEM и LOCATION представлена в виде составной сущности с названием STORAGE. (Первичный ключ сущности STORAGE состоит из первичных ключей связанных таблиц, что делает сущность STORAGE составной.) Сравните реализацию этих связей с одной из связей M:N между ORDER и INV_TYPE. В последнем случае первичный ключ сущности ORDER_ITEM состоит из OI_LINE и ORDER_ID, таким образом, представляя сущность ORDER_ITEM как слабую.

Обратите внимание, что исходное бизнес-правило, выражающее связь M:N между ITEM и LOCATION, можно записать так:

Каждый предмет (ITEM) может храниться во многих местах (LOCATION), и каждое место хранения (LOCATION) может быть использовано для хранения многих предметов (ITEM).

Однако эта связь M:N преобразуется в две связи 1:M, выражаемые такими двумя бизнес-правилами:

- Каждый предмет (ITEM) может встречаться один или более раз в хранилище (STORAGE), и каждое хранилище (STORAGE) может содержать несколько предметов (ITEM).
- Каждое место (LOCATION) может ссылаться один или несколько раз на хранилище (STORAGE), и каждая запись в хранилище (STORAGE) ссылается на одно и только одно местоположение (LOCATION).

Говоря кратко, проектировщик БД должен объединять компоненты проекта, в то же время, не забывая о том, что:

- проект базировался на нескольких источниках информации;
- порядок, в котором разрабатывались бизнес-правила и ER-диаграммы, не имеет существенного значения;
- различные конечные пользователи имеют разные представления о связях. Таким образом, проектировщик должен обладать определенной профессиональной интуицией, выбирая способ отражения в проекте базы данных разных точек зрения.

Резюме

Начальный концептуальный проект базы данных основан на результатах начальной стадии разработки, определяющей предназначение системы базы данных, организационную структуру и описание операций. На начальной стадии также выявляются проблемы и ограничения, предложения по предназначению системы, а также сфера действия и границы системы. Определение сферы действия и границ позволяет проектировщику сосредоточиться на важнейших системных процессах и их взаимодействии.

На начальной стадии разработки проектировщик подготавливает этап концептуального проектирования, тщательно устанавливая источники информации компании и пользователей системы. В хорошем концептуальном проекте необходимо принимать в расчет требования конечных пользователей, как к нерегламентированным запросам, так и к формальным отчетам. Содержимое отчета и его форма тщательно анализируются с целью определения сущностей, которые должны войти в концептуальный проект.

Подробное описание операций помогает определить бизнес-правила, на которых базируются компоненты ER-диаграмм. Бизнес-правила помогают подтвердить и/или установить как сами сущности, так и их связи (1:1, 1:M или M:N) с другими сущностями. ER-диаграмма содержит информацию о связности, характеристиках сущности (слабая или нормальная), мощности, является ли сущность составной и является ли связь необязательной. (Хотя на ER-диаграммах типа "птичья лапка" мощность связи не указывается явным образом, тип связи и связность неявно определяют мощность. Например, необязательная связь 1:M неявно определяет мощность (0,N) для стороны "M" связи. Фактически если вы используете такой инструментарий, как Visio, вам необходимо выразить мощность как функцию связностей.)

Основные термины

Модуль — module

Несериализованные объекты, малоценный инвентарь — nonserialized items

Сериализованные объекты, объекты строгого учета — serialized items

Вопросы

1. Какие факторы оказывают влияние на проект базы данных на начальной стадии разработки?
2. Почему организационная структура имеет существенное значение для проектировщика баз данных?
3. В чем различие между сферой действия проекта базы данных и его границами? Почему сфера действия и границы так важны для проектировщика БД?
4. Какие бизнес-правила и связи можно записать для ER-диаграммы, представленной на рис. 7.20?

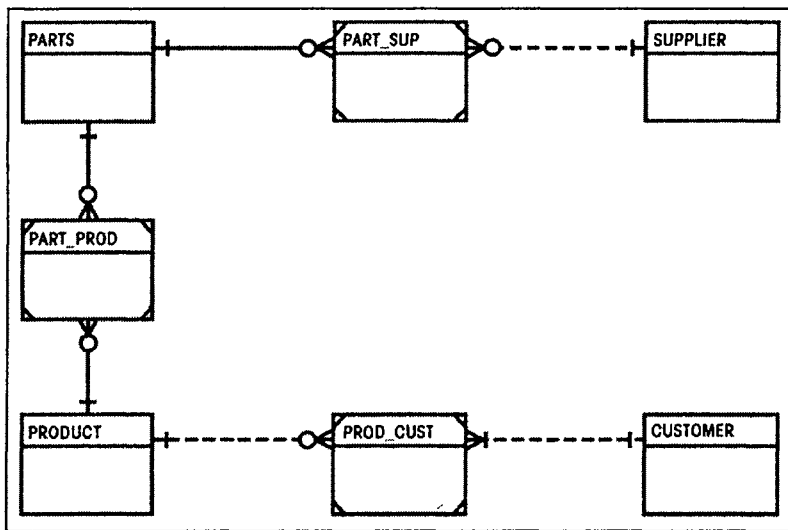


Рис. 7.20. ER-диаграмма для вопроса 4

5. Запишите связность и мощность для каждой сущности, представленной в вопросе 4.
6. Что такое модуль и какова его роль в системе базы данных?
7. Что такое взаимодействие модулей и как оно достигается?

Задачи

1. Модифицируйте изначальную ER-диаграмму, представленную на рис. 7.19, с целью включения следующих супертипов и подтипов сущности: "Пользователем (USER) UCL может быть студент или преподаватель".
2. Используя ER-диаграмму, проиллюстрируйте, каким образом изменения, сделанные вами в задаче 1, повлияют на связь сущности USER с сущностями LOG (журнал), RESERVATION (резервирование), CHECK_OUT (выдача под отчет) и WITHDRAW (выдача).
3. Создайте начальную ER-диаграмму для агентства по продаже автомобилей. Агентство продает как новые, так и подержанные автомобили, а также производит сервисное обслуживание. Ваш проект должен быть основан на следующих бизнес-правилах:
 - продавец может продать несколько автомобилей, но каждый автомобиль продается только одним продавцом;
 - клиент может приобрести несколько автомобилей, но каждый автомобиль продается только одному клиенту;
 - продавец выписывает только один счет на каждый проданный автомобиль;
 - клиент получает счет на каждый купленный автомобиль;
 - клиенту может потребоваться только обслуживание автомобиля, т. е. клиентом является не только тот, кто покупает автомобиль;
 - когда клиент ставит один или несколько автомобилей на обслуживание или ремонт, на каждый автомобиль выписывается одна квитанция;
 - агентство ведет историю обслуживания для всех автомобилей. В записи об обслуживании указывается серийный номер автомобиля;
 - с автомобилем, поставленным на обслуживание, могут работать несколько механиков, и каждый механик может обслуживать несколько автомобилей;
 - для обслуживания и ремонта автомобиля могут потребоваться (или нет) различные запчасти (например, настройка карбюратора или чистка инжектора не требует запчастей).
4. Создайте начальную ER-диаграмму для ателье проката видеокассет. Используйте (по крайней мере) следующее описание операций, на которых могут быть основаны бизнес-правила.

Ателье проката видеокассет классифицирует названия фильмов в соответствии с их типом: комедия, вестерн, классика, фантастика, мультфильмы, боевик, музыка или новинки. Каждый тип содержит несколько наименований фильмов и для большинства наименований данного типа доступны несколько копий. Например, посмотрите сводку, представленную в табл. 7.11.

Таблица 7.11. Связь типов видеокассет и наименований

Тип	Наименование	Копия
Музикл	Моя прекрасная леди	1
	Моя прекрасная леди	2
	Оклахома!	1
	Оклахома!	2
	Оклахома!	3
Мультфильм	Дилли Дэлли и Чит Чат Кэт	1
	Дилли Дэлли и Чит Чат Кэт	2
	Дилли Дэлли и Чит Чат Кэт	3
Боевик	Путешествие по Амазонке	1
	Путешествие по Амазонке	2

При разработке проекта БД ателье проката видеокассет помните о следующих условиях:

- классификация типов видеокассет — стандартная; совсем необязательно, что фильмы всех типов имеются в наличии;
- список фильмов обновляется по мере необходимости, однако фильм из этого списка может не выдаваться, если хозяин ателье решит, что этого нельзя делать по каким-либо причинам;
- ателье не обязательно заказывает фильмы у всех поставщиков из имеющегося списка; некоторые поставщики в списке являются потенциальными, у которых фильмы могут быть заказаны в будущем;
- фильмы, которые считаются новинками, переводятся в соответствующий тип через 30 дней. Директор ателье проката должен иметь возможность получения отчета о прокате по типам за некоторый период (неделя, месяц, год);
- если клиенту нужен фильм с определенным названием, у клерка должна быть возможность найти его быстро. Когда клиент выбирает одно или несколько наименований, выписывается счет. В каждом счете может быть указана стоимость одного или более фильмов. Все клиенты производят оплату наличными;
- при получении фильма клиентом в записи хранятся дата получения, дата предполагаемого возврата и время. Клерк должен иметь возможность при возвращении кассет быстро проверить дату возврата и рассчитать плату за возможную задержку;
- директору ателье требуются периодические отчеты о доходах по наименованиям и по типам фильмов. Хозяин также должен иметь возможность получения периодических отчетов об имеющихся запасах кассет и отслеживать заказы;
- директору ателье, нанявшему двух сотрудников на полный рабочий день (на окладе) и трех сотрудников на неполный день (почасовиков), необходимо от-

слеживать время работы всех сотрудников и вести ведомости на оплату. Почасовики должны составлять график работы, и все сотрудники должны расписываться в журнале учета рабочего времени.

5. Предположим, что предприниматель производит три дорогих, но небольших по размеру изделия P1, P2 и P3. Изделие P1 собирается из компонентов C1 и C2; изделие P2 собирается из компонентов C1, C3 и C4, а изделие P3 собирается из компонентов C2 и C3. Компоненты могут закупаться у нескольких поставщиков, как это представлено в табл. 7.12.

Таблица 7.12. Сводка по компонентам-поставщикам

Поставщик	Поставляемые компоненты
V1	C1, C2
V2	C1, C2, C3, C4
V3	C1, C2, C4

Каждое изделие имеет уникальный серийный номер, как и каждый компонент. Для контроля характеристик изделия тщательно хранятся записи, позволяющие проследить, от какого поставщика были получены компоненты для данного изделия.

Изделия продаются непосредственно конечному клиенту, т. е. никакая оптовая торговля не допускается. Записи о продаже включают в себя идентификационную информацию о клиенте и серийный номер изделия. Используя эту информацию, проделайте следующее:

- напишите бизнес-правила, управляющие производством и продажей изделий;
- создайте ER-диаграмму, позволяющую производителю отслеживать требования к изделию/компоненту.

6. Создайте ER-диаграмму для склада оборудования. Убедитесь, что вы учли в диаграмме (по крайней мере) операции по хранению, инвентаризацию и учет сотрудников. Постройте ER-диаграмму на основе набора разработанных вами бизнес-правил. (*Замечание:* полезно посетить такой склад и провести опрос, чтобы определить тип и масштаб складских операций.)
7. Используйте следующее краткое описание операций в качестве источника для проекта БД.

Все самолеты, принадлежащие авиакомпании ROBCOR, требуют периодического обслуживания. При проведении обслуживания используется форма регистрации технического обслуживания, куда заносятся идентификационный номер самолета, основная причина технического обслуживания и дата начала обслуживания. Пример формы приведен на рис. 7.21.

Обратите внимание, что в форме есть строки для записи даты окончания обслуживания и для подписи ответственного механика, отвечающего за ввод самолета в строй. Каждая форма технического обслуживания нумеруется последо-

вательно. (Внимание: ответственный механик является специалистом FAA IA (авторизованным специалистом федерального авиационного агентства). Из 10 механиков авиакомпания ROBCOR такие полномочия имеют только 3.)


ROBCOR Aircraft		
Service		
Log#: 2155	Aircraft: 2155W	Date in: 18-Jan-2002
Checked in by: George D. Ramsey (115)		page 1 of 1
Squawk summary		
1. Left mag rough on run-up _____		
2. Nose gear shimmies at taxi speeds _____		
3. Left main gear door does not close flush with wing _____		
4. Gear struts do not maintain proper pressure _____		
5. Left engine vibrates when power is pulled back to 20 in. manifold pressure _____		
6. _____		
7. _____		
8. _____		
9. _____		
10. _____		
Aircraft release date: 19-Jan-2002		Released by: Bea L. Patterson (109)

Рис. 7.21. Форма регистрации проведения технического обслуживания

После создания формы учета технического обслуживания ее номер записывается в специальный лист спецификаций технического обслуживания, также называемый *формой очередного обслуживания*. По завершении лист спецификаций содержит подробные сведения о каждом проведенном обслуживании, времени, затраченном на него, использованных запасных частях (при необходимости) и идентификационных данных механика, выполнявшего обслуживание. Лист спецификаций служит основанием для выставления счета (с учетом времени и запасных частей по каждой операции технического обслуживания), а также является одним из документов, позволяющих контролировать расходование запасных частей. Пример листа спецификаций приведен на рис. 7.22.


Расход запасных частей, использованных при любых операциях, должен быть подписан механиком, который их получал, что позволяет отслеживать наличие запасных частей на складе. Каждая подписанная форма содержит список всех запасных частей, связанных с данной записью в регистрационной форме обслуживания. Поэтому в форме выдачи запасных частей содержится номер регист-

рационной формы, на основании которой выдавались запасные части. Кроме того, процедура выдачи запасных частей используется для обновления склада запасных частей авиакомпании ROBCOR. Пример формы выдачи запасных частей представлен на рис. 7.22.

ROBCOR Aircraft Service

Log#: 2155

Item	Action description	Time	Part	Units	Mechanic
1	Performed run-up. Rough mag reset.	0.8	None	0	112
2	Cleaned #2 bottom plug, left engine	0.9	None	0	112
3	Replaced nose gear shimmy dampener	1.3	P-213342A	1	103
4	Replaced left main gear door oleo strut seal	1.7	GR/311109S	1	112
5	Cleaned and checked gear strut seals	1.6	None	0	116




page 1 of 1

Рис. 7.22. Форма очередного технического обслуживания

ROBCOR Aircraft Service

Form sequence #: 24226
Log #: 2155

Part	Description	Units	Unit Price	Mechanic
P-213342A	Nose gear shimmy dampener, PA31-350/1973	1	\$189.45	112
GR/311109S	Left main gear door oleo strut Seal, PA31-350/173	1	\$59.76	103



page 1 of 1

Рис. 7.23. Форма выдачи запасных частей

Механики являются высококлассными специалистами авиакомпании ROBCOR, и их специализация достаточно отличается, например, от бухгалтера или секретаря.

Используя это краткое описание операций, нарисуйте ER-диаграмму с полным набором обозначений по методу Чена. Убедитесь, что вы включили все необходимые связи, связности и мощности.

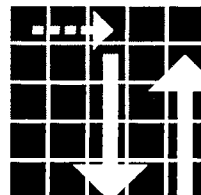
8. Вы только что приняты на работу в компанию грузоперевозок ROBCOR для разработки базы данных. Чтобы получить представление о предназначении БД, вы должны затратить какое-то время на опрос служащих ROBCOR и изучение некоторых отчетных форм, используемых для контроля назначения водителей трейлеров и обслуживания грузовиков. Вы выяснили следующее:
 - некоторые водители — специалисты по вождению трейлеров более чем одного типа, имеющихся в компании ROBCOR. Водитель может, следовательно, быть назначен более чем на один тип трейлера в течение некоторого периода времени. ROBCOR оперирует несколькими трейлерами данного типа. Например, в компании имеются два панелевоза, четыре пятитонных грузовика, два одноосных самосвала, один двухосный самосвал и один шестнадцатиколесный трейлер. Водитель с простой лицензией специализируется на вождении только панелевоза и пятитонного грузовика и может, следовательно, быть назначен на один из шести трейлеров. Водителя с коммерческой лицензией, в которой указана допустимая грузоподъемность, можно назначить на любые девять трейлеров компании ROBCOR. Каждый раз, когда водитель назначается на трейлер, делается запись в журнале, где указываются номер сотрудника, идентификационный номер трейлера и дата выезда. По возвращении водителя из рейса журнал обновляется, и в него заносятся дата прибытия и число затраченных на рейс часов;
 - если трейлеру необходимо техническое обслуживание, то заполняется журнал технического обслуживания. В этот журнал заносится дата постановки трейлера на обслуживание. Трейлер не может покинуть зону технического обслуживания, пока в журнал не будет занесена дата окончания, и журнал не будет подписан инспектором;
 - все инспекторы являются квалифицированными механиками, но не все механики являются инспекторами;
 - после того как в журнале технического обслуживания сделана запись, ее номер передается в журнал учета работ, в котором регистрируются все производимые при обслуживании работы. Единственная запись в журнале технического обслуживания может инициировать несколько записей в журнале учета работ. Например, трейлеру могут потребоваться замена масла и инжектора, регулировка тормозов и ремонт крыла;
 - каждую запись в журнале учета работ подписывает механик, выполняющий эту работу. Для отслеживания стоимости обслуживания каждого трейлера в запись включаются использованные запасные части и время, затраченное на установку запасных частей или выполнение обслуживания. (Не всегда при

обслуживании требуются запасные части. Например, регулировка сцепления не требует использования каких-либо запасных частей.);

- всем сотрудникам автоматически предоставляется страхование жизни. Однако в льготы компании ROBCOR могут включаться дополнительные варианты по страхованию жизни, а также страхование от несчастных случаев и по инвалидности. Сотрудники могут выбрать какой-либо из возможных вариантов или не выбирать ни одного из них.

Используя это краткое описание, создайте ER-диаграмму. Убедитесь, что вы включили все необходимые сущности и связи и определили все связности и мощности.

Глава 8



Университетская лаборатория: тестирование концептуального проекта, логическое проектирование и реализация

В этой главе мы будем изучать:

- ☐ как создаются и уточняются модули системы управления лабораторией;
- ☐ как определяются атрибуты и домены для каждой сущности в начальной ER-диаграмме;
- ☐ как определяются транзакции базы данных внутри модуля системы;
- ☐ процесс тестирования системы, в котором используются технологии моделирования и нормализации совместно с обнаружением и устранением избыточности данных;
- ☐ действия при реализации проекта базы данных;
- ☐ действия при тестировании базы данных и ее развитие;
- ☐ действия при вводе базы данных в эксплуатацию.

Обзор

В этой главе мы будем проверять ER-модель, разработанную нами в *гл. 7*. Проверка (verification) заключается в установлении связи между моделированием базы данных и процессом проектирования, а также приложениями БД. Следовательно, процесс проверки требует распознавания и определения всех транзакций БД (вставка, обновление, удаление и вывод данных) и, кроме того, этот процесс должен быть достаточно гибким для того, чтобы обеспечить возможные расширения и модификации системы.

Процесс проверки также позволит проектировщику найти и устранить ненужную избыточность данных, обеспечить целостность данных, выявить необходимость модификации и удостовериться, что выполнены все требования конечных пользователей. Процесс проверки включает в себя интеграцию представлений множества конечных пользователей о базе данных, каждый из которых предъявляет свои требования к системе и предполагает выполнять определенный набор операций. В этой главе (как это происходит и на самом деле) в процессе проверки придется модифицировать началь-

ную ER-модель. Здесь мы будем использовать процедуры нормализации, которые зачастую рассматриваются как часть этапа *логического* проектирования. Однако в действительности процедуры моделирования и нормализации в основном используются совместно (см. гл. 6). Модификация может включать в себя создание и/или удаление новых сущностей, атрибутов в имеющихся сущностях и связях.

Перед тем как приступить к процессу проверки, необходимо, прежде всего, определить все атрибуты и домены для каждой сущности в начальной ER-диаграмме, а также нормализовать имеющиеся сущности. Мы должны выбрать надлежащий первичный ключ и определить для связи между сущностями внешний ключ. После окончания процесса проверки мы завершим проект, выполнив логическое и физическое моделирование.

8.1. Завершение концептуального и логического проектирования базы данных

Концептуальный проект базы данных, разработанный в гл. 7, все еще является, в некоторой степени, черновым. Хотя он и помогает определить основные характеристики среды базы данных, все же в нем не хватает сведений, которые позволили бы эффективно реализовать БД. Вновь прибегая к строительной аналогии, можно сказать, что важно знать, из чего будет сделана стена, изображенная на проекте, — из досок, кирпича, блоков или бетона; будет ли эта стена несущей или просто играть роль перегородки. Короче говоря, необходима подробная информация.

Перед тем как двигаться дальше, будет полезно вновь просмотреть *разд. 6.4* и *рис. 6.3*. Это поможет вам оценить то, что было сделано в гл. 7, и выяснить, что осталось еще сделать. В гл. 7 мы выполнили следующее.

- Этап 1 (начальная стадия проектирования БД) жизненного цикла БД (DBLC). Обратите особое внимание на перечень действий на начальной стадии проекта, представленный на *рис. 6.4*.
- Затем мы приступили к Этапу 2 DBLC (проектирование БД). Мы завершили шаги 1—3, представленные в табл. 6.2. То есть в гл. 7 мы определили и проанализировали бизнес-правила, выявили основные сущности, а также связи между этими сущностями.

В этой главе мы завершим концептуальный и логический проекты базы данных Университетской Компьютерной Лаборатории. Мы обсудим элементы физического дизайна и исследуем проблемы, с которыми придется столкнуться на этапе реализации. В табл. 8.1 представлен перечень задач, которые нам предстоит рассмотреть.

Таблица 8.1. Задачи, рассматриваемые в этой главе

Задача	Раздел главы
ER-моделирование и нормализация	8.2
Проверка моделирования данных	8.3
Логический проект	8.4
Физический проект	8.5

Таблица 8.1 (окончание)

Задача	Раздел главы
Реализация	8.6
Тестирование и оценка	8.7
Работа с базой данных	8.8

Начальная ER-диаграмма, созданная в гл. 7 (рис. 7.19), будет для нас отправной точкой. Другими словами, мы будем использовать начальную разработку в качестве основы для определения атрибутов, нормализации таблиц и проверки модели, с тем чтобы убедиться, что проект отвечает требованиям, предъявляемым к обработке данных. Нужно помнить, что действия, которые мы описали, зачастую выполняются совместно и носят итеративный характер. То есть они нередко выполняются одновременно и многократно повторяются. Например, после определения сущностей и их атрибутов необходимо выполнить нормализацию, при этом могут возникнуть новые сущности и атрибуты, которые вновь нужно нормализовать. После успешного выполнения всех этих мероприятий мы получим ER-модель, в которой сущности, атрибуты и их связи отвечают требованиям конечных пользователей к обработке информации.

Для упрощения разработки концептуальной модели Университетской Компьютерной Лаборатории (UCL) мы будем использовать два модуля, каждый из которых действует в своей функциональной сфере. Напомним, что этими двумя модулями, впервые представленным в гл. 7, табл. 7.4, являются:

- *модуль "Система Управления Лабораторией" (Lab)*, который отражает повседневную деятельность лаборатории и предназначен для учета пользователей лаборатории, персонала UCL и распределения ресурсов. С помощью этого модуля директор компьютерной лаборатории (CLD) может отслеживать ресурсы UCL по типам пользователей, факультетам и т. д. Такой учет очень важен при формировании бюджета лаборатории;
- *модуль "Система Управления Материально-Техническим Снабжением" (Inventory)*, в котором учитываются оборудование, расходные материалы, заказы и обслуживание. (Например, оборудование, отправленное на ремонт, временно снимается с учета, в то время, как отремонтированное оборудование вновь ставится на учет.) Этот модуль также позволяет директору (CLD) отслеживать оборудование, которое берут под отчет преподаватели и сотрудники.

Список сущностей, определенных во время этого процесса, а также используемые префиксы атрибутов представлены в табл. 8.2.

Таблица 8.2. Организация проекта Университетской Лаборатории

Модуль	Сущность	Префикс атрибутов
Система Управления Лабораторией (Lab)	USER	USER_
	LOG	LOG_

Таблица 8.2 (окончание)

Модуль	Сущность	Префикс атрибутов
Система Управления Материально-Техническим Снабжением (Inventory)	LAB_ASSISTANT	LA_
	WORK_SCHEDULE	SCHED_
	HOURS_WORKED	HW_
	RESERVATION	RES_
	RES_SLOT	RSLOT
	INV_TYPE	TY_
	ITEM	ITEM_
	STORAGE	STOR_
	LOCATION	LOC_
	REPAIR	REP_
	VENDOR	VEND_
	ORDER	ORD_
	ORDER_ITEM	OI_
	WITHDRAW	WD_
	WD_ITEM	WI_
	CHECK_OUT	CO_
	CHECK_OUT_ITEM	CI_
	INV_TRANS	TRANS_
	TR_ITEM	TI_

Если вы сравните сущности, перечисленные в табл. 8.2, с начальным проектом БД, представленным в гл. 7, то обнаружите, что здесь появились новые компоненты. Например, появился атрибут RES_SLOT (время резервирования), поскольку отдельное резервирование (RESERVATION) может быть выполнено на несколько дат. Например, 01/11/02 преподаватель может зарезервировать лабораторию с 8:00 до 8:50 и с 13:00 до 13:50 на 01/23/02, а также с 6:00 до 8:40 на 01/25/02. Следовательно, между сущностями RESERVATION и RES_SLOT есть связь типа 1:M. Новые сущности будут обсуждаться при разработке каждого модуля системы. Мы также увидим, что в процессе тестирования некоторые сущности, представленные в табл. 8.2, будут заменены другими.

8.2. Завершение концептуального проекта: сущности, атрибуты и нормализация

В предыдущем разделе данной главы мы описали два системных модуля. Теперь определим сущности и атрибуты каждого из них. Даже после определения сущно-

стей и атрибутов в процессе нормализации их приходится все время пересматривать. Другими словами, нормализация рассматривается как неотъемлемая часть процесса ER-моделирования. Поэтому необходимо тщательно отслеживать функциональные зависимости. Для выявления новых сущностей и определения практических методов оценки их функциональности используются технические приемы нормализации, продемонстрированные в гл. 4. Сущности и их атрибуты также необходимо пересматривать в процессе их оценки с точки зрения требований конечного пользователя.

В этой главе техника нормализации рассматриваться не будет. Структуры, представленные здесь, тем не менее, получены при помощи процедур нормализации соответствующего уровня. Мы полагаемся на ваши знания принципов и технических приемов нормализации, необходимых в процессе создания и проверки сущностей и атрибутов. Корректируя модель, не забывайте, что погоня за элегантностью дизайна зачастую приводит к снижению скорости и качества обработки информации.

8.2.1. Модуль "Система Управления Лабораторией"

Перед тем как приступить к исследованию структуры каждого компонента модуля "Система Управления Лабораторией" (Lab), взглянем на фрагмент ER-диаграммы, представленный на рис. 8.1.

Примечание

Чтобы сосредоточиться на связях сущностей, на всех ER-диаграммах, приведенных в этой главе, показаны только атрибуты первичного и внешнего ключей каждой сущности. Вы можете, конечно, добавить атрибуты, которые будут определены во всех итоговых таблицах.

Если вы при проектировании БД используете Visio или подобный CASE-инструментарий, то помните, что на уровне сущности вы создаете только атрибут первичного ключа, если ее связь с родительской сущностью не определена. Вы никогда не определите внешний ключ на уровне сущности вне ее связи с другими сущностями. Вместо этого сначала создайте сущности, их первичные ключи (пока нет первичных атрибутов, наследованных от связанных сущностей) и их связи, затем щелкните правой кнопкой (в среде CASE-инструментария) на сущности и выберите в контекстном меню команду **Update foreign keys** (Обновить внешние ключи). Это будет гарантией того, что:

- все внешние ключи будут включены в состав сущностей так, чтобы они корректно отражали связи, определенные для этих сущностей;
- все компоненты первичных ключей, унаследованные от связанных с ними сущностей, будут должным образом включены в каждую сущность, которая требует использования таких унаследованных первичных ключей;
- внешние ключи всегда будут наследовать характеристики атрибутов от первичных ключей, на которые они указывают. А это означает, что вы никогда не увидите сообщения об ошибке типа "incompatible data type" (несовместимый тип данных) при попытке реализации проекта;
- программное обеспечение будет автоматически контролировать связи на непротиворечивость, замкнутость и некорректность, что, по сути, обеспечит нам встроенный контроль качества проекта.

Большинство CASE-приложений обеспечивает такой сервис по созданию первичных и внешних ключей, и если он доступен, то его всегда необходимо использовать. И все-таки конечной целью является разработка высококачественного проекта, который может быть успешно реализован. Если вы уверены, что проект не содержит ошибок, сделанных на уровне логики и реализации, вы можете двигаться вперед и добавлять остальные атрибуты.

Естественно, если вы выполняете первоначальное проектирование с помощью карандаша и бумаги, т. е. не можете воспользоваться только что описанным сервисом, вы должны записать атрибуты первичного и внешнего ключа на уровне сущности для того, чтобы посмотреть, к чему приведет реализация. То есть вы сами будете выполнять функцию "update foreign keys", которую можно найти практически в любом программном обеспечении автоматизации проектирования.

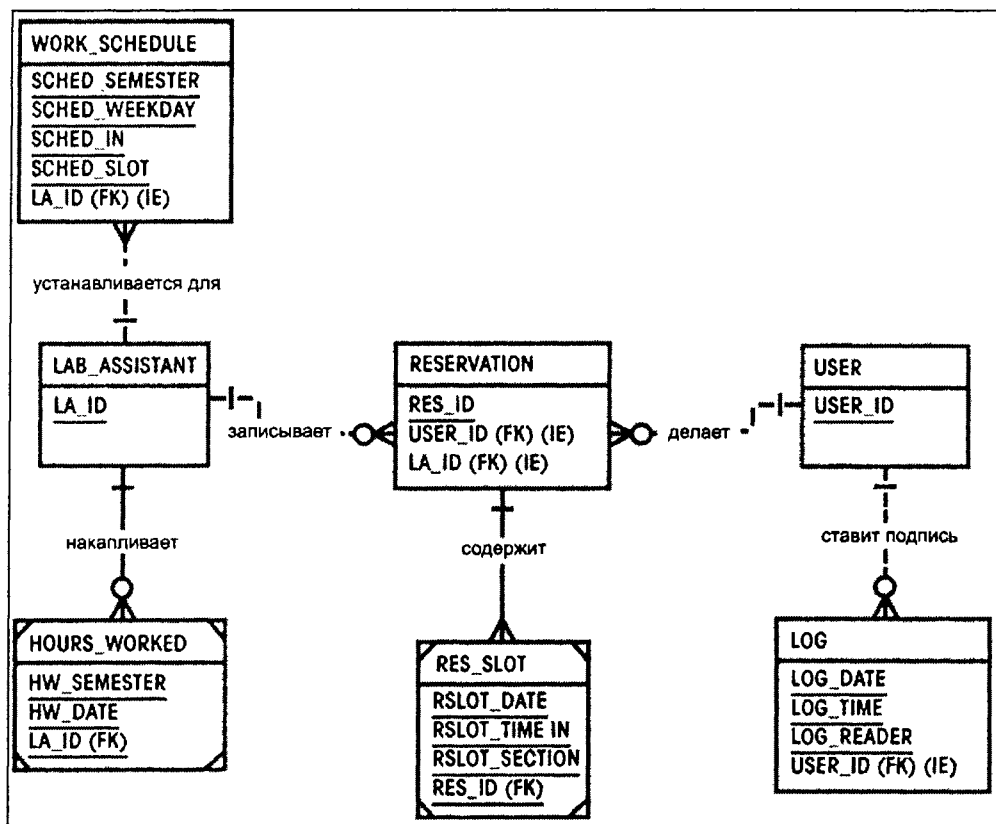


Рис. 8.1. Фрагмент ER-диаграммы модуля "Система Управления Лабораторией" (Lab)

Фрагмент ER-диаграммы, представленный на рис. 8.1, мы будем использовать в качестве ориентира для определения нашего положения в процессе проектирования и дальнейшего направления действий. С помощью рис. 8.1 и табл. 8.2 проведем исследование характеристик сущности USER, представленных в табл. 8.3

Таблица 8.3. Сущность USER

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
USER_ID	Идентификационный код пользователя		PK	
DEPT_CODE	Код факультета			
USER_TYPE	Тип пользователя: Fac = преподаватель Staff = сотрудник Stu = студент			
USER_CLASS	Класс пользователя: UG = студент GR = выпускник Fac = преподаватель Staff = сотрудник			
USER_SEX	Пол: M = мужской F = женский			

Обратите внимание в табл. 8.3 на атрибут DEPT_CODE, который добавлен для того, чтобы директор мог отслеживать факультеты, пользующиеся услугами лаборатории. Эта информация имеет существенное значение, поскольку факультеты участвуют в формировании бюджета лаборатории, и факультеты, использующие лабораторию чаще, должны вносить больший вклад, чем факультеты, которые редко пользуются лабораторией. Следовательно, возможность контролировать использование лаборатории факультетами (по атрибуту DEPT_CODE) очень важна. Хотя атрибут DEPT_CODE, очевидно, является внешним ключом сущности DEPARTMENT, нам не требуется в настоящий момент какая-либо дополнительная информация по факультетам. Поэтому сущность DEPARTMENT не включена в данный проект, а DEPT_CODE, включенный в сущность USER, в настоящий момент времени не идентифицируется как внешний ключ.

Примечание

Мы прежде всего создали сущность USER, чтобы смоделировать систему и убедиться, что мы можем обеспечить работоспособность системы в данной главе. В таблице USER атрибуты USER_ID, DEPT_CODE и USER_TYPE могут использоваться для подведения итогов использования лаборатории при планировании бюджета.

Как и в большинстве реальных проектов, фактические данные пользователя Системы Управления Университетской Компьютерной Лабораторией (UCLMS) являются частью существующей внешней базы данных. Эта внешняя БД управляется и обслуживается Университетом. Процедуры ввода данных и структура этой БД отличаются от тех, что мы предлагаем для UCLMS. Например, пользовательские данные лаборатории вводятся с помощью считывателя магнитных карт, на которых имеется гораздо больше информации, чем необходимо в данном проекте. Добавление этих данных к проекту, представленному в этой главе, не улучшит понимание основных процессов проверки, которые и являются основной нашей целью. По тем же причинам мы не включили в проект сущности DEPARTMENT (факультет) и COLLEGE (колледж). (Включение атрибута DEPT_CODE в сущность USER сделано только для того, чтобы обеспечить возможность отслеживания использования ресурсов лаборатории без необходимости получения дополнительных сведений о факультете или колледже.) Однако мы будем обращаться к некоторым аспектам реальной информационной системы в задачах гл. 13.

Данные, используемые в этой главе, представляют собой лишь небольшое подмножество реальных сведений. Например, в реальной системе содержится более 30 000 записей об использовании лаборатории в течение семестра, и число записей быстро растет. Наконец, для того, чтобы обеспечить конфиденциальность, все реальные значения изменены.

Обратите внимание, что структура таблицы USER обладает некоторой избыточностью. Например, атрибут USER_CLASS, очевидно, определяет атрибут USER_TYPE. Если вы знаете, что USER_CLASS = UG, то вы также знаете, что USER_TYPE есть Stu. С другой стороны, USER_TYPE не определяет USER_CLASS, поскольку Stu может означать и UG (старшекурсник), и GR (выпускник). В любом случае мы теперь знаем, что таблица приведена к 2НФ. Чтобы полностью устранить функциональные зависимости в 2НФ, мы могли бы объединить USER_TYPE и USER_CLASS в единый строковый атрибут, в котором содержались бы значения Stu/UG, Stu/GR, Fac и Staff. Однако поскольку Университет устанавливает форму отчета, в которой должны быть представлены суммарные сведения об использовании лаборатории по преподавателям, сотрудникам и студентам, предпочтительнее использовать предложенную структуру таблицы. Кроме того, необходимо, чтобы отчет подразделялся по различным категориям студентов (выпускник/старшекурсник, мужчина/женщина). В реальном проекте базы данных часто требуется идти на компромисс между обеспечением эффективности обработки информации и безукоризненностью проекта. Пример данных таблицы USER представлен на рис. 8.2.

Если внимательно посмотреть на данные, представленные на рис. 8.2, то можно заметить, что когда USER_TYPE принимает значение Fac или Staff, то USER_CLASS тоже равен Fac или Staff. Это дублирование необходимо для отчетности, поскольку позволяет легко создавать сводку по атрибуту USER_CLASS. Наконец, обратите внимание, что в значениях атрибута USER_ID мы использовали дефисы. Мы считаем, что с дефисами номер социального страхования легче читается, а это отнимает всего лишь два лишних байта на одну запись. Хранение информации обходится недорого и постоянно дешевеет, поэтому 2 дополнительных байта на запись USER_ID не слишком обременительны. С другой стороны, если при поиске данных в качестве ключа будет выбран номер социального страхования, скорость поиска увеличится, если в символьный атрибут дефисы не включать. Современные системы баз данных предоставляют проектировщику возможность использования маски ввода в целях улучшения внешнего представления (999-99-9999, вместо 9999999999) без фактического хранения этих дефисов в базе данных. Как обычно, для сбалансирования противоречащих требований профессиональные проектировщики должны полагаться на свою интуицию.

	USER_ID	DEPT_CODE	USER_TYPE	USER_CLASS	USER_SEX
▶ +	55-67-4867	CIS	Fac	Fac	F
+	264-77-0032	CIS	Fac	Fac	M
+	278-12-3332	ACCT	Staff	Staff	F
+	286-39-0551	ACCT	Fac	Fac	F
+	295-07-2314	BIOL	Staff	Staff	M
+	298-56-7891	MKTAMGT	Fac	Fac	M
+	299-87-9234	CIS	Fac	Fac	M
+	301-12-2889	SOC	Stu	GR	F
+	301-12-3123	ACCT	Fac	Fac	F
+	301-23-4245	ACCT	Fac	Fac	F
+	301-25-7228	CIS	Fac	Fac	F
+	318-59-2995	CIS	Stu	UG	F
+	324-52-8551	CIS	Stu	UG	F
+	331-56-4678	SOC	Stu	GR	M
+	331-62-0032	CIS	Stu	GR	M
+	341-55-0585	BIOL	Stu	UG	F
+	345-76-6899	CIS	Stu	UG	F
+	352-14-5875	CIS	Staff	Staff	M
+	352-14-9219	ACCT	Stu	UG	M
+	367-11-2512	MKTAMGT	Stu	GR	F
+	367-11-8438	SOC	Stu	UG	M
+	367-85-6784	HIST	Stu	UG	M
+	386-12-3456	CIS	Staff	Staff	M

Рис. 8.2. Пример данных таблицы USER

Теперь, в соответствии с описанием модуля, представленным в табл. 8.2, мы исследуем сущность LOG, представленную таблицей LOG. Каждый раз, когда пользователи (USER) получают доступ в лабораторию, их идентификатор считывается в журнал с помощью одного или нескольких считывателей магнитных карт.

Примечание

Для макетирования системы и обеспечения самостоятельности БД мы должны модифицировать процедуру ввода в сущность USER. Например, если USER_ID не соответствует ни одной записи в таблице USER, то таблица USER обновляется путем добавления нового пользователя. Помните, что в реальной системе необходимо обеспечить безопасность, поэтому система должна отказывать в регистрации пользователям, чья идентификация не соответствует имеющимся записям во внешней базе данных студентов.

Сведения о сущности LOG представлены в табл. 8.4.

Таблица 8.4. Сущность LOG

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
LOG_DATE	Дата регистрации (системная)		PK	

Таблица 8.4 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
LOG_TIME	Время регистрации (системное)		PK	
LOG_READER	Номер устройства считывания магнитных карт		PK	
USER_ID	Идентификационный номер пользователя		FK	USER

В табл. 8.4 показан составной первичный ключ, полученный из атрибутов LOG_DATE, LOG_TIME и LOG_READER. Предполагается, что невозможна ситуация, при которой считывающее устройство в данное время (с точностью до секунды) делает более одной записи, поскольку для обработки одной карты требуется несколько секунд. Если атрибут LOG_READER не включить в первичный ключ, то возможна ситуация, при которой два различных устройства считывания, принявшие магнитные карты в одно и то же время, сделают одновременную запись и тем самым нарушат требование целостности.

Пример данных таблицы LOG приведен на рис. 8.3.

	LOG_DATE	LOG_TIME	LOG_READER	USER_ID
►	17-Jan-2002	2:24:32 PM	1	401-85-2112
	17-Jan-2002	2:24:39 PM	2	331-62-0032
	17-Jan-2002	2:25:41 PM	2	324-52-8551
	17-Jan-2002	2:25:44 PM	1	352-14-9219
	17-Jan-2002	2:25:58 PM	2	443-56-5678
	17-Jan-2002	2:26:03 PM	1	295-07-2314
	17-Jan-2002	2:26:28 PM	2	264-77-0032
	17-Jan-2002	2:29:19 PM	1	401-67-5878
	17-Jan-2002	2:30:35 PM	1	286-39-0551
	17-Jan-2002	2:32:09 PM	1	367-11-2512
	17-Jan-2002	2:32:09 PM	2	301-12-2889
	17-Jan-2002	2:35:42 PM	2	386-12-3456
	17-Jan-2002	2:36:33 PM	1	414-38-4217
	17-Jan-2002	2:38:21 PM	1	331-62-0032
	17-Jan-2002	2:39:57 PM	1	401-85-2112
	17-Jan-2002	2:42:13 PM	1	401-67-2318
	17-Jan-2002	2:43:01 PM	2	345-76-6899
	17-Jan-2002	2:43:03 PM	1	403-11-2323
	17-Jan-2002	2:43:42 PM	1	352-14-5875
	17-Jan-2002	2:43:58 PM	2	299-87-9234
	17-Jan-2002	2:44:47 PM	1	264-77-0032
	17-Jan-2002	2:45:53 PM	1	544-56-3234
	17-Jan-2002	2:48:37 PM	2	414-75-8901

Рис. 8.3. Пример данных таблицы LOG

В таблице на рис. 8.3 обратите внимание на новый атрибут LOG_ID, наличие которого устраняет необходимость в составном ключе. Можно возразить, что этот атрибут избыточен, поскольку комбинация LOG_DATE, LOG_TIME и LOG_READER уже выполняет роль первичного ключа. Это действительно так, но наличие потенциального ключа не нарушает структуру таблицы, а первичный ключ, состоящий из одного атрибута, уменьшает накладные расходы, упрощая индексирование по первичному ключу. Это еще один пример решения, которое приходится принимать проектировщику (вы, например, можете принять решение использовать составной первичный ключ). Попробуйте ответить на такой вопрос: нужен ли данный атрибут и какая от него польза? Если он необходим, то какую цену придется заплатить за его создание и использование? Какую функцию он будет выполнять и нельзя ли выполнить ее с помощью имеющихся атрибутов? Как видите, проектирование базы данных всегда требует определенной профессиональной интуиции.

Директор руководит группой ассистентов (LA). Политика Университета состоит в ограничении штата лаборатории, участвующего в повседневных операциях, дипломированными ассистентами (GA), работающими студентами (SW, Student Workers) и студентами-практикантами (Work Study Students, WS). Ассистенты (GA) могут работать до 20 часов в неделю, SW — 10 часов в неделю, а WS — 4 часа в неделю. Атрибуты ассистентов приведены в табл. 8.5.

Таблица 8.5. Сущность LAB_ASSISTANT

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многознач- ный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
LA_ID	Идентификационный номер ассистента		PK	
LA_NAME	Имя ассистента	C		
LA_PHONE	Телефон ассистента в кампусе	C		
LA_SEMESTER	Последний отработанный семестр	C		
LA_TYPE	Квалификация ассистента: GA = дипломированный ассистент SW = работающий студент WS = студент-практикант			
LA_HIRE_DATE	Дата приема на работу			

При изучении табл. 8.5 нужно иметь в виду, что информационные требования часто определяют степень разбиения составной сущности. Например, вероятно, директору потребуется список телефонов для связи с ассистентами. Поэтому неплохо было бы

разбить атрибут LA_NAME на имя, фамилию и инициал. С другой стороны, нет смысла разделять номер телефона, например, 4142345 на его базовый номер и расширение 2345. Хотя верно, что для большей информативности необходима большая атомарность, все же излишек атрибутов увеличивает сложность системы, не давая при этом никаких очевидных преимуществ. Точно так же в атрибут LA_SEMESTER включаются записи типа SPRING02 (весна 2002) для обозначения последнего семестра, в течение которого работал ассистент (LA). Отчеты конечных пользователей показывают, что мы немного выиграем от того, что разобьем эту запись на обозначение семестра SPRING и года 02.

Несколько записей таблицы LAB_ASSISTANT представлены на рис. 8.4.

	LA_ID	LA_LNAME	LA_FNAME	LA_INITIAL	LA_PHONE	LA_SEMESTER	LA_TYPE	LA_HIRE_DATE
▶	355-77-8456	Jones	James	C	4123234	SPRING02	GA	07-Jan-2002
+	367-88-2341	Smith	Anne	G	4123245	SPRING02	SW	10-Jun-2001
+	367-99-9585	Hernandez	Maria	M	4123245	SPRING02	GA	07-Jan-2002
+	392-12-5456	Inum	Klong	J	4123234	SPRING02	WS	07-Jan-2002
+	402-12-6543	Jamerson	George	D	4126789	SPRING02	SW	13-Jul-2001
+	411-23-4568	Patterson	Herman	W	4127890	SPRING02	GA	07-Jan-2002
+	412-31-2255	Troyana	Emily	H	4121121	FALL01	SW	04-Jan-2001
+	451-25-6877	Evans	Peter	G	4123234	SPRING02	GA	07-Jan-2002
+	452-11-0094	Vann	Evangelina	D	4121121	SPRING02	SW	07-Jan-2002
+	452-23-1323	Casey	Robert	D	4145345	SPRING02	GA	19-Jun-2001
+	455-12-3456	Kellen	Nancy	B	3439006	SPRING02	GA	07-Jan-2002
+	455-23-4567	Dawson	James	D	3431234	SPRING02	SW	07-Jan-2002
+	455-34-5678	Smith	Jeffrey	A	4122229	SPRING02	GA	22-Mar-2001
+	455-45-6789	Peron	Kamey	W	4123456	SPRING02	WS	07-Jan-2002
+	456-12-3456	Korbin	Gerald	L	4145345	SPRING02	WS	07-Jan-2002
+	456-23-4567	Porter	Suzanna	D	3439006	SPRING02	WS	07-Jan-2002
+	456-34-5678	Morris	Wilson	M	5123312	SPRING02	SW	07-Jan-2002
+	456-45-6789	Wills	Kathryn	A	4146567	FALL01	GA	22-May-2001
+	457-12-3456	Thompson	Bruce	G	4126789	SPRING02	SW	07-Jan-2002
+	457-23-4567	Womack	Jeanine	J	3768993	SPRING02	SW	07-Jan-2002
+	457-34-5678	Gabril	Jason	T	3234536	SPRING02	GA	09-Aug-2001
+	457-45-6789	Tabrin	Mary	B	3765546	SPRING02	GA	07-Jan-2002
+	458-12-3456	Sorels	Joseph	P	4145569	SPRING02	GA	07-Jan-2002

Рис. 8.4. Избранные записи таблицы LAB_ASSISTANT

Примечание

Атрибут LA_SEMESTER позволяет директору проверять, можно ли назначать ассистента на работу в данном семестре. Некоторые ассистенты работают в лаборатории один семестр, в другом семестре выполняют общеуниверситетские обязанности, а в следующем семестре вновь возвращаются на работу в лабораторию. Таким образом, запись FALL98 (осень 1998) может говорить о том, что последнее назначение ассистента на работу в лаборатории было осенью 1998 года. Поскольку записи LAB_ASSISTANT после окончания ассистентом колледжа, увольнения или ухода по собственному желанию удаляются (и архивируются), обозначение LA_SEMESTER (семестр) указывает на последний, а не текущий семестр, в течение которого работал ассистент.

Для контроля графика работы ассистентов директор хранит его в том виде, как это представлено в табл. 8.6. В табл. 8.6 определены атрибуты сущности WORK_SCHEDULE (рабочий график), представленные в табл. 8.7.

Таблица 8.6. График работы ассистентов лаборатории

Период времени		Понедельник	Вторник	Среда	Четверг	Пятница	Суббота	Воскресенье
0600—0800	1	Jones (GA)	Thomas (GA)	Gabril (GA)	Evans (GA)	Hernando (GA)		
	2	Jamerson (WS)	Chung (SW)	Chung (SW)	Tabrin (GA)	Mustava (GA)		
	3	Hernando (GA)	Womack (SW)	Thomas (GA)	Jones (GA)	Tabrin (GA)		
	4	Vann (SW)	Dalton (SW)	Smith, C (SW)	Smith, C (SW)	Rommel (SW)		
0800—1000	1	Jones (GA)	Thomas (GA)	Gabril (GA)	Evans (GA)	Hernando (GA)		
	2	Hernandez (GA)	Porter (WS)	Chung (SW)	Tabrin (GA)	Mustava (GA)		
	3	Jamerson (WS)	Womack (SW)	Thomas (GA)	Dalton (SW)	Tabrin (GA)		
	4	Vann (SW)	Chung (SW)	Smith, C (SW)	Smith, C (SW)	Rommel (SW)		
1000—1200	1	Hernandez (GA)	Jones (GA)	Gabril (GA)	Jones (GA)	Hernando (GA)		
	2	Troyana (SW)	Porter (WS)	Troyana (SW)	Tabrin (GA)	Antony (SW)		
	3	Jamerson (WS)	Willis (GA)	Willis (GA)	Antony (SW)	Tabrin (GA)		
	4	Morris (SW)	Womack (SW)	Antony (SW)	Dalton (SW)	Rommel (SW)		
1200—1400	1	Evans (GA)	Jones (GA)	Gabril (GA)	Jones (GA)	Kallen (GA)	Jones (GA)	Tabrin (GA)
	2	Troyana (SW)	Vann (SW)	Troyana (SW)	Tabrin (GA)	Evans (GA)	Dalton (SW)	Mustava (GA)
	3	Willis (GA)	Willis (GA)	Willis (GA)	Antony (SW)	Mustava (GA)		
	4	Highton (SW)	Womack (SW)	Antony (SW)	Smith, C (SW)	Rostav (SW)		
1400—1600	1	Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Tabrin (GA)
	2	Inum (WS)	Vann (SW)	Troyana (SW)	Tabrin (GA)	Willis (GA)	Dalton (SW)	Mustava (GA)

Таблица 8.6 (окончание)

Период времени	Понедельник	Вторник	Среда	Четверг	Пятница	Суббота	Воскре- сенье
1600—1800	3 Jones (GA)	Morris (SW)	Morris (SW)	Mustava (GA)	Mustava (GA)	Batey (SW)	Kadin (SW)
	4 Highlon(SW)	Womack (SW)	Chung (SW)	Jones (WS)	Batey (WS)	Avery (SW)	
	1 Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2 Kadin (SW)	Vann (SW)	Sorals (GA)	Thomas (GA)	Willis (GA)	Dalton (SW)	Mustava (GA)
1800—2000	3 Winston (SW)	Morris (SW)	Morris (SW)	Jones, A (WS)	Jones, A (WS)	Batey (SW)	
	4 Rostav (SW)	Avery (SW)	Avery (SW)	Highlon (SW)	Jones (GA)	Rommel (SW)	
	1 Evans (GA)	Hernando (GA)	Kallen (GA)	Kallen (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2 Kadin (SW)	Thomas (GA)	Sorals (GA)	Sorals (GA)	Willis (GA)	Aaaron (SW)	Mustava (GA)
2000—2200	3 Winston (SW)	Avery (SW)	Thomas (GA)	Jones, A (WS)	Aaaron (SW)	Rommel (SW)	
	4 Rostav (SW)	Winston (SW)	Avery (SW)	Kadin (SW)	Batey (SW)		
	1 Evans (GA)	Hernando (GA)	Kallen (GA)	Sorals (GA)	Kallen (GA)	Gabril (GA)	Sorals (GA)
	2 Kadin (SW)	Thomas (GA)	Sorals (GA)	Thomas (GA)	Willis (GA)	Aaaron (SW)	Witte (SW)
2200—2400	3 Highlon (SW)	Avery (SW)	Thomas (GA)	Jones, A (WS)	Aaron (SW)	Winston (SW)	
	4 Rostav (SW)	Winston (SW)	Winston (SW)	Rostav (SW)	Rommel (SW)		
	1 Casey (GA)	Casey (GA)	Casey (GA)	Casey (GA)	Casey (GA)	Gabril (GA)	Sorals (GA)
	2 Thompson (SW)	Thompson (SW)	Thompson (SW)	Karpov (SW)	Karpov (SW)	Witte (SW)	Witte (SW)
3							
4							

Таблица 8.7. Сущность WORK_SCHEDULE

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многознач- ный (M)	Первич- ный ключ (PK) и/или внешний ключ (FK)	Ссылки
SCHED_SEMESTER	Идентификатор семестра	C	PK	
SCHED_WEEKDAY	Рабочие дни		PK	
SCHED_IN	Время начала		PK	
SCHED_OUT	Время конца			
SCHED_SLOT	Номер отрезка времени (1—4)		PK	
LA_ID	Идентификатор ассистента		FK	LAB_ASSISTANT

Первичный ключ таблицы WORK_SCHEDULE состоит из атрибутов SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN и SCHED_SLOT. Требуется, чтобы любой ассистент не имел два одинаковых значения для времени начала работы в один и тот же рабочий день. Например, ассистент не может иметь два одинаковых значения "время начала работы" = 10:00 в понедельник. Записи в атрибутах SCHED_IN и SCHED_OUT основаны на 24-часовом формате времени и могут принимать значения от 0600 до 2400.

Второе требование состоит в том, что на заданный отрезок времени не может быть назначено более четырех ассистентов. К сожалению, первоначально сделанный выбор первичного ключа не соответствует этим требованиям. Этот недостаток проекта можно исправить тремя способами.

1. Создать первичный ключ, состоящий из атрибутов SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN и SCHED_SLOT.
2. Создать уникальный индекс на основе атрибутов SCHED_SEMESTER, SCHED_WEEKDAY, SCHED_IN и LA_ID.
3. Создать правило проверки данных для гарантии того, что атрибут SCHED_SLOT будет принимать только значения 1, 2, 3 или 4.

Первый способ можно реализовать декларированием компонентов первичного ключа при создании таблицы. Второй способ можно реализовать с помощью команды CREATE INDEX. Третий способ требует написания кода для реализации проверки значения атрибута SCHED_SLOT. Если вы используете PCYБД Oracle, то третий способ можно реализовать с помощью триггера. (Если же вы используете MS Access, то для реализации третьего способа можно установить правило проверки данных.) Реализация первого и третьего способов гарантирует, что в данный отрезок времени будут работать максимум четыре ассистента. Второй способ гарантирует, что ни

один ассистент не появится более одного раза в данной комбинации "рабочий день/время".

Пример данных таблицы WORK_SCHEDULE представлен на рис. 8.5.

	SCHED_SEMESTER	SCHED_WEEKDAY	SCHED_IN	SCHED_OUT	SCHED_SLOT	LA_ID
▶	SPRING02	Friday	08:00	08:00	1	387-99-9565
	SPRING02	Friday	08:00	10:00	1	387-99-9565
	SPRING02	Friday	10:00	12:00	1	387-99-9565
	SPRING02	Friday	14:00	16:00	2	456-45-6789
	SPRING02	Friday	16:00	18:00	2	456-45-6789
	SPRING02	Friday	16:00	18:00	4	365-77-8458
	SPRING02	Friday	18:00	20:00	2	456-45-6789
	SPRING02	Friday	20:00	22:00	2	456-45-6789
	SPRING02	Monday	06:00	08:00	1	365-77-8458
	SPRING02	Monday	06:00	08:00	2	402-12-6543
	SPRING02	Monday	06:00	08:00	3	387-99-9565
	SPRING02	Monday	06:00	08:00	4	452-11-0094
	SPRING02	Monday	08:00	10:00	1	365-77-8458
	SPRING02	Monday	08:00	10:00	2	387-99-9565
	SPRING02	Monday	08:00	10:00	3	402-12-6543
	SPRING02	Monday	08:00	10:00	4	452-11-0094
	SPRING02	Monday	10:00	12:00	1	387-99-9565
	SPRING02	Monday	10:00	12:00	2	412-31-2255
	SPRING02	Monday	10:00	12:00	3	456-45-6789
	SPRING02	Monday	10:00	12:00	4	456-34-5678
	SPRING02	Monday	12:00	14:00	2	412-31-2255
	SPRING02	Monday	12:00	14:00	3	456-45-6789
	SPRING02	Monday	14:00	18:00	3	365-77-8458

Рис. 8.5. Пример данных таблицы WORK_SCHEDULE

Заметим, что порядок, в котором атрибуты WORK_SCHEDULE хранятся в таблице, не зависит от реляционной модели. Однако ваша СУБД, вероятнее всего, индексирует таблицу по первичному ключу. В данном случае порядок индексирования начинается с атрибута SCHED_SEMESTER и затем, соответственно, переходит на атрибуты SCHED_WEEKDAY, SCHED_IN и SCHED_SLOT. Данные, представленные на рис. 8.5, отображают именно такой порядок индексирования. В этом случае первичный ключ гарантирует порядок, независимый от LA_ID (идентификатор ассистента) или от фамилии ассистента. Здесь еще раз продемонстрировано условие, которое является несущественным для реляционной модели, но, в конечном счете, оказывает влияние на то, как проектировщик оценивает проект для реализации и разработки приложений.

С помощью структуры HOURS_WORKED, представленной в табл. 8.8, можно отслеживать отработанные каждым ассистентом часы в течение двухнедельного оплачиваемого периода.

Таблица 8.8. Сущность *HOURS_WORKED*

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
LA_ID	Идентификацион- ный номер ассис- тента		PK, FK	LAB_ ASSISTANT
HW_SEMESTER	Обозначение семе- стра	C	PK	
HW_DATE	Конечная дата ра- бочего периода		PK	
HW_HOURS_ WORKED	Общее количество отработанных часов			

Первичный ключ сущности *HOURS_WORKED* составлен из атрибутов *LA_ID*, *HW_SEMESTER* и *HW_DATE*. Включение в первичный ключ атрибута *HW_DATE* необходимо для обеспечения целостности сущности, поскольку комбинация *LA_ID* и *HW_SEMESTER* может встретиться несколько раз (каждый ассистент работает несколько недель в семестре). Также обратите внимание, что атрибут *HW_HOURS_WORKED*, представляющий собой общее количество часов, отработанных ассистентом в течение оплачиваемого периода, вводится конечным пользователем вручную. (В данном случае атрибут *HW_HOURS_WORKED* не является производным атрибутом. Заметим, что в имеющейся таблице нет атрибутов, на основе которых можно вычислить атрибут *HW_HOURS_WORKED*. Если в сущность *HOURS_WORKED* включить атрибуты *HW_TIME_IN* и *HW_TIME_OUT*, то значение атрибута *HW_HOURS_WORKED* можно будет рассчитать на основе этих двух атрибутов, и в таком случае он станет производным атрибутом.) Данные сущности *HOURS_WORKED* образуют основу для приложения, создающего платежную ведомость. Пример данных, входящих в эту таблицу, приведен на рис. 8.6.

	LA_ID	HW_SEMESTER	HW_DATE	HW_HOURS_WORKED
▶	365-77-8458	SPRING02	15-Jan-2002	40
	387-99-9585	SPRING02	15-Jan-2002	40
	402-12-6543	SPRING02	15-Jan-2002	8
	412-31-2255	SPRING02	15-Jan-2002	20
	452-11-0094	SPRING02	15-Jan-2002	20
	456-34-5678	SPRING02	15-Jan-2002	20
	456-45-6789	SPRING02	15-Jan-2002	40

Рис. 8.6. Пример данных таблицы *HOURS_WORKED*

Хотя в основном лаборатория предоставляется в распоряжение студентов, часть времени ее могут занимать преподаватели для проведения занятий или сотрудники для

обслуживания и модернизации оборудования. Чтобы в системе можно было выполнять резервирование лаборатории, создана первоначальная структура RESERVATION, представленная в табл. 8.9.

Таблица 8.9. Сущность RESERVATION

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
RES_DATE	Дата выполнения резервирования		PK	
USER_ID	Идентификационный номер пользователя (только преподаватель или сотрудник)		PK, FK	USER
RES_DATES_ RESVD	Зарезервированные даты	M	PK	
RES_TIME_IN	Зарезервированное время (начало)	M		
RES_TIME_ OUT	Зарезервированное время (конец)	M		
RES_USER	Число пользователей в зарезервированное время	M		
LA_ID	Ассистент, который выполнял резервирование		FK	LAB_ ASSISTANT

В этой таблице имеется несколько многозначных атрибутов. Например, преподаватель может резервировать лабораторию на несколько занятий и внутри каждой даты несколько раз в день, причем каждый раз для различного числа пользователей. Эти многозначные атрибуты непременно станут источником проблем на стадии реализации. Например, сколько производных атрибутов (RES_DATES_RESVD1, RES_DATES_RESVD2, RES_DATES_RESVD3) атрибута RES_DATES_RESVD необходимо хранить для дат резервирования? Если вы предусмотрите какое-то заведомо избыточное число, то получите много пустых значений. Если вы предусмотрите слишком мало места, то на резервирование будет накладываться дополнительное ограничение. И если вы впоследствии захотите разрешить дополнительное резервирование, то необходимо будет менять структуру таблицы. Проблема усугубляется еще и тем, что для каждой резервируемой даты возможно еще несколько дублирующихся часов. Таких производных атрибутов может быть очень много. (Однажды мы проверяли базу данных, в которой одна из таблиц содержала 114 атрибутов — и это число возросло. Неудивительно, что эта база данных работала очень неэффективно!)

Применяя правила нормализации, которые мы изучали в гл. 4, можно разбить структуру RESERVATION на две таблицы, имеющие связь 1:M. Первая из этих двух таблиц, которая по-прежнему называется RESERVATION и относится к стороне "1", представлена в табл. 8.10.

Таблица 8.10. Исправленная сущность RESERVATION

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многознач- ный (M)	Первич- ный ключ (PK) и/или внешний ключ (FK)	Ссылки
RES_ID	Идентификатор резервирования		PK	
RES_DATE	Дата выполнения резервирования			
USER_ID	Идентификационный номер пользователя (только преподаватель или сотрудник)		FK	USER
LA_ID	Ассистент, который выполнял резервирование		FK	LAB ASSISTANT

Пример данных для исправленной таблицы RESERVATION приведен на рис. 8.7.

		RES_ID	RES_DATE	USER_ID	LA_ID
▶	+	523	25-Jan-2002	255-67-4567	387-99-9565
	+	524	25-Jan-2002	264-77-0032	387-99-9565
	+	525	26-Jan-2002	301-23-4245	451-25-6877

Рис. 8.7. Пример данных таблицы RESERVATION

Эта новая структура RESERVATION работает намного лучше. Каждый раз, когда ассистент выполняет резервирование, дата резервирования записывается в атрибут RES_DATE. Мы можем также проследивать, для кого (USER_ID) выполнялось резервирование и кто (LA_ID) его выполнял. Многозначные резервирования затем обрабатываются на стороне "M" в таблице RES_SLOT, структура которой представлена в табл. 8.11.

Таблица 8.11. Сущность (слабая) RES_SLOT

Имя атрибута	Содержимое	Тип атрибута: составной (C), произ- водный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
RES_ID	Идентификатор резервирования		PK, FK	RESERVATION

Таблица 8.11 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
RSLOT_DATE	Зарезервированная дата		PK	
RSLOT_TIME_IN	Зарезервированное время (начало)		PK	
RSLOT_TIME_OUT	Зарезервированное время (конец)			
RSLOT_USERS	Число пользователей в зарезервированное время			
RSLOT_LAB_SECTION	Зарезервированная секция лаборатории		PK	

Обратите внимание, что ввод атрибута RSLOT_LAB_SECTION позволяет выполнять два резервирования на один день и в одно время, при этом разные группы будут работать в разных секциях лаборатории. Также обратите внимание, что RES_SLOT является слабой сущностью, поскольку ее существование зависит от сущности RESERVATION, и один из компонентов ее первичного ключа (RES_ID) наследуется от сущности RESERVATION.

На рис. 8.8 представлен пример данных таблицы резервирования.

	RES_ID	RSLOT_DATE	RSLOT_TIME_IN	RSLOT_LAB_SECTION	RSLOT_TIME_OUT	RSLOT_USERS
▶	523	03-Feb-2002	8:00 AM	A	9:50 AM	23
	523	10-Feb-2002	8:00 AM	A	9:50 AM	23
	524	04-Feb-2002	2:00 PM	C	3:15 PM	35
	525	03-Feb-2002	6:00 PM	A	8:40 PM	18
	525	07-Feb-2002	10:00 AM	A	10:50 AM	24
	525	10-Feb-2002	8:00 PM	B	8:40 PM	18

Рис. 8.8. Пример данных таблицы RES_SLOT

Исследуя данные, представленные на рис. 8.8, можно легко проследить весь процесс резервирования, если помнить о связи 1:M между сущностями RESERVATION и RES_SLOT. Обратите внимание, например, что 25 января 2002 года (см. данные таблицы RESERVATION на рис. 8.7) для пользователя 255-67-4567 было выполнено резервирование (см. данные таблицы RES_SLOT на рис. 8.8) для 23 пользователей на 3 февраля 2002 года с 8:00 до 9:50 в секции А лаборатории, а также на 23 пользователя на 10 февраля 2002 года с 8:00 до 9:50 в секции А.

8.2.2. Модуль

"Система Управления

Материально-Техническим Снабжением"

Чтобы детально проследить процесс разработки модуля "Система Управления Материально-Техническим Снабжением" (Inventory), полезно взглянуть на его ER-компоненты, представленные на рис. 8.9.

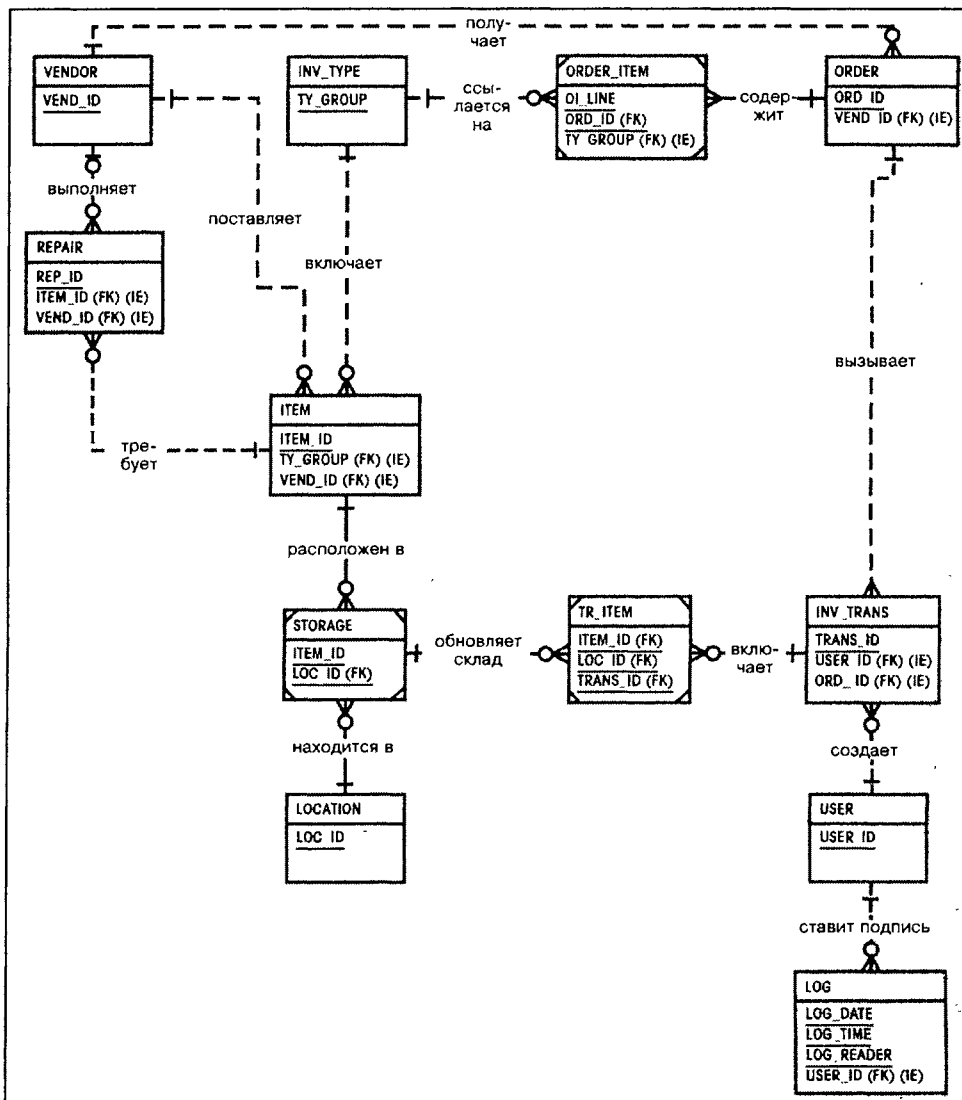


Рис. 8.9. Сегмент ER-диаграммы для модуля "Система Управления Материально-Техническим Снабжением" (Inventory)

Эта иллюстрация поможет нам понять весь процесс разработки. При создании сущностей и атрибутов модуля все время обращайтесь к этому рисунку.

В процессе изучения рис. 8.9 может возникнуть вопрос, почему здесь не представлены сущности **WITHDRAW** и **CHECK_OUT**, имеющиеся в начальной ER-диаграмме (см. гл. 7, а также табл. 8.2). Вы также можете заметить, что здесь появились новые сущности, например, **INV_TRANS**. Эти изменения являются следствием процесса проверки ER-модели, мы детально обсудим их позже в этом разделе. Кроме того, отметим, что сущность **USER**, не входящая явно в модуль Inventory, которую мы уже обсудили в разд. 8.2.1, является связующим звеном между сущностями фрагментов. Поэтому хотя мы и не будем обсуждать эту сущность, включить ее в данный фрагмент необходимо.

Сущность **INV_TYPE** играет важную роль в данном модуле. Ее существование помогает директору (CLD) получать подробные инвентарные сводки. (Например, сколько имеется в наличии пачек бумаги 8,5" × 11"? Сколько доступно лазерных принтеров? Сколько имеется коробок с дискетами 3,5"?). Для того чтобы понять предназначение сущности **INV_TYPE**, вы должны, прежде всего, познакомиться с классификацией инвентаря, представленной в табл. 8.12.

Таблица 8.12. Иерархия классификаций инвентаря

Группа	Категория	Класс	Тип	Подтип
HWPCDTP3	Оборудование (HW)	Персональный компьютер (PC)	Настольный (DT)	Pentium III (P3)
HWPCLT3	Оборудование (HW)	Персональный компьютер (PC)	Ноутбук (LT)	Intel III (P3)
HWPCLT2	Оборудование (HW)	Персональный компьютер (PC)	Ноутбук (LT)	Pentium II (P2)
HWPRLSBL	Оборудование (HW)	Принтер (PR)	Лазерный (LS)	Черно-белый (BL)
HWPRLSBL	Оборудование (HW)	Принтер (PR)	Струйный (IJ)	Цветной (CO)
SUPSS11	Материалы (SU)	Бумага (PP)	Простая (SS)	11 дюймов (11)
HWEXVIXX	Оборудование (HW)	Плата расширения (EX)	Видео (VI)	XX
SWDBXXXX	Программное обеспечение (SW)	База данных (DB)	XX	XX

Обратите внимание, что в классификации, представленной в табл. 8.12, имеются три категории: оборудование, программное обеспечение и материалы. Отметим также, что код каждой группы точно описывает тип инвентаря, о котором мы говорим. Например, код первой группы (HWPCDTP3) описывает категорию "оборудование" (HW), класс "персональный компьютер" (PC), тип "настольный" (DT) и подтип Pentium III (P3). Некоторые коды групп (например, последний в табл. 8.12) не оп-

ределяют тип и подтип, вместо этого в них используются символы XX для указания, что эти параметры не определены. Позже мы покажем, что коды групп могут использоваться в качестве первичных ключей для определения строк INV_TYPE, и будем хранить компоненты категории, класса, типа и подтипа как отдельные атрибуты для расширения возможностей при выполнении отчетов.

Иерархию классификаций можно также представить в виде дерева (рис. 8.10).

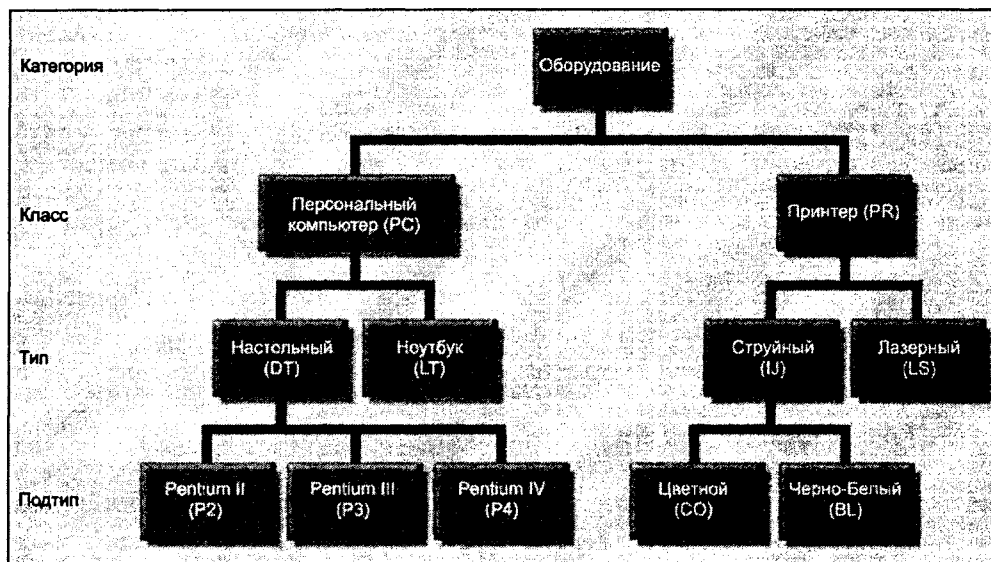


Рис. 8.10. Иерархия классификаций сущности INV_TYPE в виде дерева

Иерархия классификаций, представленная в табл. 8.12 и на рис. 8.10, отражена в структуре INV_TYPE (табл. 8.13). На рис. 8.11 приведен пример заполнения данными структуры INV_TYPE.

Таблица 8.13. Сущность INV_TYPE

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
TY_GROUP	Код группы	C	PK	
TY_CATEGORY	Категория			
TY_CLASS	Класс			
TY_TYPE	Тип			
TY_SUBTYPE	Подтип			

Таблица 8.13 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
TY_DESCRIPTION	Описание группы			
TY_UNIT	Единица измерения (коробка, пачка и т. д.)			

	TY_GROUP	TY_CATEGORY	TY_CLASS	TY_TYPE	TY_SUBTYPE	TY_DESCRIPTION	TY_UNIT
+	HABAPR1X	HW	BA	PN	XX	Bar Code Reader, Pen Type	UN
+	HVACAMQ12	HW	CA	MO	12	External Modem VMod, 12'	UN
+	HVACASP08	HW	CA	SP	08	Serial Printer Cable, 8-pin	UN
+	HVEXHD10	HW	EX	HD	10	Expansion Board-IDE HD ctrl.	UN
+	HVEXHDMF	HW	EX	HD	MF	Expansion Board-MFM HD ctrl.	UN
+	HVEXMEXX	HW	EX	ME	XX	Expansion Board, Memory	UN
+	HVEXVIXX	HW	EX	VI	XX	Expansion Board, Video	UN
+	HVMSXXXX	HW	MS	XX	XX	Hardware, Miscellaneous	UN
+	HVANCETCX	HW	NC	ET	CX	Ethernet NIC, Coax	UN
+	HVANCETTP	HW	NC	ET	TP	Ethernet NIC, Twisted Pair	UN
+	HVANCTR4M	HW	NC	TR	4M	Token Ring, NIC 4M	UN
+	HVAPCDT48	HW	PC	DT	48	Intel 486 desktop	UN
+	HVAPCDTCE	HW	PC	DT	CE	Intel Celeron desktop computer	UN
+	HVAPCDTM2	HW	PC	DT	M2	Apple Mac II	UN
+	HVAPCDTP2	HW	PC	DT	P2	Desktop PC, Intel Pentium P2	UN
+	HVAPCDTP3	HW	PC	DT	P3	Desktop PC, Intel Pentium P3	UN
+	HVAPCDTP4	HW	PC	DT	P4	Desktop PC, Intel Pentium P4	UN
+	HVAPCLT48	HW	PC	LT	48	Intel 486 laptop	UN
+	HVAPCLTCE	HW	PC	LT	CE	Intel Celeron laptop computer	UN
+	HVAPCLTP2	HW	PC	DT	P2	Laptop PC, Intel Pentium P2	UN
+	HVAPCLTP3	HW	PC	LT	P3	Laptop PC, Intel Pentium P3	UN
+	HVAPCLTP4	HW	PC	LT	P4	Laptop PC, Intel Pentium P4	UN
+	HVAPCTWP4	HW	PC	TWP	P4	Tower PC, Intel Pentium P4	UN
+	HVAPJAXX	HW	PJ	PA	XX	LCD Flat Panel Projector	UN
+	HVAPRDM13	HW	PR	DM	13	Dot Matrix printer, 132 col.	UN
+	HVAPRDM80	HW	PR	DM	80	Dot Matrix printer, 80 col.	UN
+	HVAPRIJBL	HW	PR	IJ	BL	Ink-jet Printer, black	UN
+	HVAPRIJCO	HW	PR	IJ	CO	Ink-jet Printer, color	UN
+	HVAPRLPMF	HW	PR	LP	MF	Line printer, Mainframe	UN

Рис. 8.11. Примеры данных таблицы INV_TYPE

При рассмотрении структуры INV_TYPE в табл. 8.13 обратите внимание, что в ней используется первичный ключ, состоящий из одного атрибута (TY_GROUP), что позволяет быстрее и эффективнее обрабатывать запросы к системе. Хотя мы могли бы создать составной ключ, комбинируя TY_CATEGORY, TY_CLASS, TY_TYPE и TY_SUBTYPE, такой первичный ключ привел бы к использованию сложной системы указателей СУБД, что снизило бы эффективность системы. Кроме того, разбиение

атрибута TY_GROUP на TY_CATEGORY, TY_CLASS, TY_TYPE и TY_SUBTYPE упростит создание различных сводных отчетов, в то же время сохраняя все преимущества использования первичного ключа, состоящего из одного атрибута! (В гл. 3 мы отмечали, что требования к информации могут влиять на процесс проектирования. Табл. 8.13 и пример заполнения данных на рис. 8.11 хорошо иллюстрируют это утверждение.)

Несмотря на то, что сущность INV_TYPE обеспечивает достаточную гибкость в получении сводок по инвентарю, мы все же должны иметь возможность обращаться и к отдельной позиции. Например, важно знать, что в наличии имеется 217 компьютеров, но нам также нужно уметь отслеживать каждый компьютер, установленный в офисе преподавателя или в лаборатории. Связь между INV_TYPE и ITEM обеспечивает такую возможность (еще раз обратитесь к фрагменту ER-диаграммы, представленному на рис. 8.9). Структура сущности ITEM представлена в табл. 8.14.

Таблица 8.14. Сущность ITEM

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первич- ный ключ (PK) и/или внешний ключ (FK)	Ссылки
ITEM_ID	Идентификационный код объекта		PK	
TY_GROUP	Код группы		FK	INV_ TYPE
ITEM_UNOV_ID	Университетский иден- тификационный номер			
ITEM_ SERIAL_NUM	Серийный номер объек- та (от производителя)			
ITEM_DESCRIPTOR	Описание объекта			
ITEM_QTY	Общее количество во всех местах			
VEND_ID	Оригинальный код по- ставщика		FK	VENDOR
ITEM_STATUS	Статус объекта: 1 = доступен 2 = в ремонте 3 = неисправен 4 = на руках			
ITEM_BUY_DATE	Дата приобретения			

Рассмотрим состояние атрибутов сущности ITEM для трех основных типов инвентаря:

- *Оборудование.* После покупки 20 компьютеров каждому из них будет присвоен ITEM_ID и ITEM_UNIV_ID, а также ITEM_SERIAL_NUM, при этом будут сделаны 20 записей.
- *Материалы.* После приобретения 20 пачек бумаги для лазерных принтеров будет сделана только одна запись, поскольку нет необходимости учитывать каждую коробку. Поэтому в этом случае не требуется индивидуальный номер (ID) и атрибут ITEM_UNIV_ID будет иметь пустое значение. Кроме того, коробка с бумагой не имеет серийного номера, поэтому ITEM_SERIAL_NUM также имеет пустое значение. Однако поскольку директор должен иметь возможность учитывать пачки бумаги, необходимо в качестве первичного ключа использовать ITEM_ID. Естественно, чтобы избежать пустых значений, для ITEM_UNIV_ID и ITEM_SERIAL_NUM можно использовать специальные коды, например, 00000.
- *Программное обеспечение.* Даже если лицензия закуплена на 180 копий программного обеспечения, в таблице ITEM будет сделана только одна запись. Отдельные инсталляции можно отслеживать с помощью сущности STORAGE, представленной в табл. 8.15.

Таблица 8.15. Сущность STORAGE

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
LOC_ID	Идентификационный код места нахождения		PK, FK	LOCATIONS
ITEM_ID	Идентификатор объекта		PK, FK	ITEM
STOR_QTY	Количество в данном месте			

Обратите внимание, что атрибут ITEM_QTY в табл. 8.14 является производным, поскольку он получается суммированием количества имеющихся в наличии объектов по разным местам. Помните, что для повышения качества проекта желательно избегать производных атрибутов. Наличие такого атрибута в данном случае отражает желание конечных пользователей получать быстрый ответ на запрос типа "Сколько пачек бумаги размера 8,5" × 11" имеется во всех местах хранения?" Поскольку производный атрибут ITEM_QTY вычисляется и записывается в таблицу ITEM прикладным программным обеспечением всякий раз при выполнении операций над инвентарем, он не вызовет аномалий данных.

Некоторые примеры данных таблицы ITEM приведены на рис. 8.12.

Необходимо иметь возможность найти любой объект в данный момент времени. Следовательно, место хранения объекта должно быть известно. Здесь важную роль играет сущность STORAGE, представленная в табл. 8.15.

	ITEM_ID	TY_GROUP	ITEM_UNIV_ID	ITEM_SERIAL_NUM	ITEM_QTY	VEND_ID	ITEM_STATUS	ITEM_BUY_DATE
▶	1212004	HVEXMEXX	00000	00000	2	PCJUN	2	10-May-2001
+	2088723	HMPCLTP3	3009765	CX-5437688	1	PCJUN	4	12-May-2001
+	2879954	SUPPCO11	00000	00000	84	PCPAL	1	12-May-2001
+	2988950	SVDBXXXX	00000	00000	20	COMPU	1	15-May-2001
+	3045887	HVEXVIXX	3422012	BBF-985643	1	COMPU	1	15-May-2001
+	3154567	SUPPSS11	00000	00000	121	PCPAL	1	15-May-2001
+	3210946	HVEXHDD	3215457	12Q3122309	1	PCJUN	1	16-May-2001
+	3212345	HVANCETCX	3245367	TR/3255675	1	PCJUN	1	16-May-2001
+	3244536	HVPCAPMC	3001323	213-FRD-21	1	PCPAL	1	17-May-2000
+	3444658	HVNCSTR4M	2156332	WR12DR3658	1	PCJUN	1	17-May-2001
+	3856456	HVANCETTP	4322456	931234-Q12	1	COMPU	3	17-May-2001
+	4009212	HVEXHDD	00000	00000	4	PCJUN	1	19-May-2001
+	4112151	HMPCLTP3	3018565	CX-5389912	1	PCJUN	1	19-May-2001
+	4112152	HMPCLTP3	3018712	CX-5354227	1	PCJUN	4	19-May-2001
+	4112153	HMPCLTP3	3018714	CX-5625138	1	PCJUN	1	19-May-2001
+	4228753	HMPCLTP3	3105622	CX-5512912	1	PCJUN	4	19-May-2001
+	4228754	HMPCLTP3	3105628	CX-5525199	1	PCJUN	1	19-May-2001
+	4238129	SUCANCO	00000	00000	18	COMPU	1	23-May-2001
+	4238130	SUCANJBL	00000	00000	29	COMPU	1	23-May-2001
+	4238131	SUCALPXX	00000	00000	35	COMPU	1	23-May-2001
+	4238132	SUDK35HD	00000	00000	52	COMPU	1	23-May-2001
+	4325456	HVANCETTP	3119875	983425-Q12	1	COMPU	1	30-May-2001
+	4358255	HMPCLTP3	3055710	CK-1105191	1	PCPAL	1	30-May-2001
+	4358256	HMPCLTP3	3057113	CK-1089295	1	PCPAL	4	30-May-2001
+	4358257	HMPJPAXX	3057115	B559121Q	1	PCPAL	4	30-May-2001
+	4358258	HMPJPAXX	3057116	B503552W	1	PCPAL	1	30-May-2001
+	4415256	HVBPANXX	3080121	XKD-125431	1	COMPU	1	07-Jun-2001
+	4451234	HVPCDTP3	3125571	CW-5122777	1	PCJUN	2	07-Jun-2001
+	4451235	HVPCDTP3	3125579	CW-5029878	1	PCJUN	4	07-Jun-2001

Рис. 8.12. Примеры данных таблицы ITEM

Пример данных таблицы STORAGE представлен на рис. 8.13.

Проследив атрибут ITEM_ID в таблице STORAGE (см. рис. 8.13) и в таблице ITEM (см. рис. 8.12), а затем TY_GROUP = SUPPSS11 в таблице INV_TYPE (см. рис. 8.11), можно определить, что первая запись в STORAGE указывает на 19 пачек простой бумаги (ITEM_ID = 3154567), которые находятся в КОМ106-1. Еще 25 пачек такой же бумаги находятся в КОМ205В-1, 52 пачки находятся в КОМ245А-2 и 5 пачек в КОМ245В-1.

Сведения о месте хранения находятся в сущности LOCATION, представленной в табл. 8.16. Вновь обратившись к сегменту ER-диаграммы на рис. 8.9, можно убедиться, что между LOCATION и STORAGE есть связь 1:М.

Таблица 8.16. Сущность LOCATION

Имя атрибута	Содержимое	Тип атрибута: составной (С), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
LOC_ID	Идентификационный код места нахождения		PK	

Таблица 8.16 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
LOC_ DESCRIPTION	Описание местонахождения (например: офис преподавателя, аудитория, кабинет и т. д.)			

Пример данных для сущности LOCATION приведен на рис. 8.14.

	LOC_ID	ITEM_ID	STOR_QTY
▶	KOM106-1	3154567	19
	KOM106-1	4238130	15
	KOM106-1	4238132	10
	KOM203E-1	2088723	1
	KOM203E-1	3212345	1
	KOM203E-1	4456789	1
	KOM203E-1	4456791	1
	KOM203E-2	4562397	1
	KOM205B-1	2879954	10
	KOM205B-1	3154567	25
	KOM245A-1	2879954	35
	KOM245A-1	4238131	18
	KOM245A-1	4238132	10
	KOM245A-1	4451238	1
	KOM245A-2	3154567	52
	KOM245A-2	4009212	1
	KOM245A-2	4112151	1
	KOM245A-2	4238132	12
	KOM245B-1	3154567	5
	KOM245B-1	4228753	1
	KOM245B-1	4358255	1

Рис. 8.13. Пример данных таблицы STORAGE

	LOC_ID	LOC_DESCRIPTION
▶	+ KOM106-1	CIS Department office
	+ KOM106-2	CIS Department office
	+ KOM200-1	Classroom
	+ KOM200-2	Classroom
	+ KOM200-3	Classroom
	+ KOM200-4	Classroom
	+ KOM203E-1	Faculty office
	+ KOM203E-2	Faculty office
	+ KOM205A-1	Hall storage closet
	+ KOM205A-2	Hall storage closet
	+ KOM205A-3	Hall storage closet
	+ KOM205A-4	Hall storage closet
	+ KOM205B-1	Hall storage closet
	+ KOM205B-2	Hall storage closet
	+ KOM245A-1	Computer lab, section A
	+ KOM245A-2	Computer lab, section A
	+ KOM245A-3	Computer lab, section A
	+ KOM245A-4	Computer lab, section A
	+ KOM245A-5	Computer lab, section A
	+ KOM245B-1	Computer lab, section B
	+ KOM245B-2	Computer lab, section B

Рис. 8.14. Пример данных таблицы LOCATION

В атрибуте LOC_DESCRIPTION сущности LOCATION вы можете записать столько сведений, сколько необходимо. Например, можно указать ящик, полки и другие сведения о месте хранения. На самом деле, при применении этой методики к системе складского учета на предприятии можно создать ряд новых атрибутов, определяющих секцию, проход, полку и ящик и т. п.

К настоящему моменту мы можем:

- ☐ обеспечить подробное описание инвентаря лаборатории UCL;
- ☐ без труда создать инвентарную ведомость по категориям;

- ☐ определить статус объекта;
- ☐ проследить объекты по месту их хранения.

Поскольку учет инвентаря очень важен, особенно во время аудиторских проверок, наша система уже демонстрирует большие возможности для конечных пользователей. Оставшуюся часть Системы Управления Материально-Техническим Снабжением мы будем строить на этой прочной основе.

Объекты инвентаря являются динамичными. Фактически некоторые объекты, такие как расходные материалы, имеют весьма ограниченный срок существования. Бумага, например, не залеживается в компьютерной лаборатории долго. Программное обеспечение (как и оборудование) устаревает. Оборудование может ломаться и требует ремонта или обслуживания. Фактически необходимость ремонта, обуславливает некоторые специальные приемы работы с базой данных. Объект, отправляемый на ремонт, остается на балансе лаборатории, но не может быть использован. Не должно быть ни одного объекта, который был бы сломан, но не отправлен на ремонт. Некоторые объекты могут ремонтироваться непосредственно в лаборатории, а некоторые возвращаются поставщику для замены или ремонта. Короче говоря, ремонт это событие, которое необходимо контролировать. Поэтому сущности REPAIR, представленной в табл. 8.17, отводится в нашем проекте важная роль.

Таблица 8.17. Сущность REPAIR

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
REP_ID	Идентификационный код ремонта		PK	
ITEM_ID	Идентификационный код объекта		FK	ITEM
REP_DATE	Дата поломки			
REP_DESCRIPT	Описание неисправности			
REP_STATUS	Состояние ремонта: 1 = в ремонте 2 = отремонтирован 3 = возвращен по- ставщику 4 = вышел из строя			
VEND_ID	Код поставщика		FK	VENDOR
REP_REF	Номер регистрации у поставщика			
REP_DATE_OUT	Дата отправки объекта			

Таблица 8.17 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
REP_DATE_IN	Дата возврата объекта			
REP_COST	Стоимость ремонта			

Некоторые атрибуты сущности REPAIR в табл. 8.17 требуют пояснения.

- VEND_ID это необязательный внешний ключ, поскольку сущность VENDOR не предоставляет никаких сведений о ремонте, если значение атрибута REP_STATUS не равно 3. Если REP_STATUS = 3, то VEND_ID содержит допустимое значение VEND_ID, т. е. совпадающее со значением в таблице VENDOR. Если вы хотите избежать пустых значений, например, используя код "no vendor", тогда для обеспечения целостности таблица VENDOR должна содержать запись "no vendor".
- Поскольку стоимость до завершения ремонта неизвестна, атрибут REP_COST имеет значение \$0,00 до тех пор, пока атрибут REP_STATUS не примет значение 2. В некоторых случаях ремонт выполняется за счет поставщика, поэтому возможен случай, когда значение атрибута REP_COST остается равным \$0,00 и при значении REP_STATUS = 2. Также иногда ремонт может выполняться силами лаборатории и не требует затрат, за исключением стоимости запасных частей.
- Атрибут REP_DESCRIPT не может принимать пустого значения, обязательно необходимо дать какое-то описание возникшей проблемы.

Несколько примеров записей таблицы REPAIR представлены на рис. 8.15.

REP_ID	ITEM_ID	REP_DATE	REP_DESCRIPT	REP_STATUS	VEND_ID	REP_REF	REP_DATE_OUT	REP_DATE_IN	REP_COST
121	3045887	15-Jan-2002	Memory board failure	2	COMPU	RT-54566-674	17-Jan-2002	20-Jan-2002	\$0.00
122	5453234	17-Jan-2002	Token ring board failure	2	PCJUN	231156	17-Jan-2002	22-Jan-2002	\$54.82
123	3045887	05-Jan-2002	Video board failure	1	COMPU	RT-57455-121	07-Jan-2002		\$0.00
124	3244536	01-Jan-2002	Power supply shortburn	3	PCPAL	FRG-324458	09-Jan-2002		\$0.00

Рис. 8.15. Пример данных таблицы REPAIR

Для размещения заказов, возврата оборудования на ремонт и т. д. в системе должны иметься сведения о поставщиках. В табл. 8.18 представлена упрощенная структура сущности VENDOR.

Таблица 8.18. Сущность VENDOR

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
VEND_ID	Идентификационный код поставщика		PK	

Таблица 8.18 (окончание)


Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
VEND_NAME	Название поставщика	C		
VEND_ADDRESS	Адрес поставщика			
VEND_CITY	Город поставщика			
VEND_STATE	Штат поставщика			
VEND_ZIP	ZIP-код			
VEND_PHONE	Телефон	C		
VEND_CONTACT	Контактное лицо	C		
VEND_CON_PHONE	Контактный телефон	C		
VEND_TECH_PHONE	Телефон службы технической поддержки	C		

При разработке структуры сущности VENDOR необходимо помнить о следующем:

- ☐ поскольку в системе должна быть возможность создания маркировки объектов, отправляемых поставщику на ремонт, необходимо подразделить адрес поставщика на улицу, город, штат и ZIP-код;
- ☐ от конечных пользователей не поступало ни требования о необходимости разделения телефонного номера поставщика на код города и собственно номер, ни требования об алфавитном упорядочивании списка контактных лиц и службы технической поддержки. Поэтому последние четыре атрибута сущности VENDOR остаются составными.

Примеры записи данных в таблицу VENDOR приведены на рис. 8.16.

VEND_ID	VEND_NAME	VEND_ADDRESS	VEND_CITY	VEND_STATE	VEND_ZIP
COMPL	ComputerLand	1012 Hard Drive	San Francisco	CA	12345
PCJUN	PC Junction	1234 Video Circle	Nashville	TN	23456
PCPAL	PC Palace	4567 Ram Lane	New York	NY	34567

Таблица VENDOR
(продолжение)


VEND_PHONE	VEND_CONTACT	VEND_CON_PHONE	VEND_TECH_PHONE
8004567789	John E. Miesler	8004567785	8004567762
6153454567	Anna D. Williamson	6153454574	6153454573
8002341234	Juan H. Cordova	8002341232	8002341235

Рис. 8.16. Пример данных таблицы VENDOR

При частой замене оборудования, программного обеспечения и расходовании материалов сущность ORDER также играет важную роль в системе учета. Ее атрибуты, необходимые для составления бюджета, контроля и удовлетворения других требований конечных пользователей, отражены в структуре сущности ORDER, представленной в табл. 8.19.

Таблица 8.19. Сущность ORDER

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
ORD_ID	Идентификационный код заказа		PK	
ORD_DATE	Дата заказа	C		
VEND_ID	Идентификационный код поставщика		FK	VENDOR
ORD_VEND_REF	Регистрационный но- мер от поставщика (необязателен)			
ORD_PO_NUM	Номер заказа			
ORD_TOT_COST	Общая стоимость за- каза, включая доставку	D		
ORD_STATUS	Статус заказа: OPEN = открыт REC = получен CANCEL = отменен PAID = оплачен			
ORD_FUND_TYPE	Источник финансиро- вания заказа: BUS = бюджет кол- леджа			
USER_ID	Лицо, инициировав- шее заказ		FK	USER

В гл. 7 мы говорили, что в каждом заказе может быть указана одна или несколько позиций. В соответствии с правилами нормализации мы разделим заказ на две сущности: ORDER и ORDER_ITEM. Сущность ORDER связана с сущностью ORDER_ITEM связью 1:M. (Еще раз взгляните на рис. 8.9, чтобы уточнить связь между ORDER и ORDER_ITEM.) В таблице ORDER (сторона "1") содержатся основные данные о заказе; в таблице ORDER_ITEM (сторона "M") содержится перечень заказанных предметов (позиций).

В атрибуте ORD_STATUS сущности ORDER отражены данные, необходимые для отчетности. Очевидно, возможно существование заказа, который получен, но еще не оплачен, поэтому мы должны различать состояния REC (received, получен) и PAID (оплачен). Хотя отмененные заказы не влияют на движение материальных средств, их тоже необходимо учитывать. Например, если вы получили счет, то хорошо было бы узнать, не отменен ли данный заказ, прежде, чем оплачивать его. Атрибут ORD_FUND_TYPE позволяет узнать, на какой бюджет записываются оплата заказанных товаров. И поскольку отчетность еще никому не мешала, мы обязаны уметь определить лицо (с помощью USER_ID), инициировавшее заказ.

Требования к информации могут также определять, на какой стороне связи 1:M мы будем хранить данные. Например, вполне вероятно, что в данный момент времени поставлена лишь часть заказа. Скажем, заказ состоит из 12 компьютеров и 25 пачек бумаги. Доставлены все 25 пачек бумаги, но только 8 компьютеров. Поскольку требуется проследить, какая часть заказа выполнена, мы должны хранить данные о квитанции доставки на стороне "M" связи 1:M. Мы должны решить также, где хранить атрибут USER_ID. Например, если требуется получать точный список лиц, заказавших конкретные позиции в любом заказе, то атрибут USER_ID должен храниться на стороне "M". С другой стороны, если нам нужно знать только, кто инициировал весь заказ, то атрибут USER_ID лучше хранить на стороне "I". Используя такой план действий, мы можем затем составить список лиц, заказавших отдельные позиции, как часть описания каждой строки заказа.

Размещение атрибута VEND_ID зависит от простого бизнес-правила. В данном случае разумно предположить, что каждый заказ адресуется одному поставщику. (Только представьте, сколько усилий придется приложить, чтобы записать строку данного заказа, если он связан с различными поставщиками!) Поэтому атрибут VEND_ID размещается на стороне "I" связи между заказом и строками заказа. В исправленной структуре ORDER (см. табл. 8.19) учтены все решения, принятые нами при выполнении данного проекта.

Чтобы проиллюстрировать размещение данных, на рис. 8.17 представлен пример заполнения таблицы ORDER.

	ORD_ID	ORD_DATE	VEND_ID	ORD_VEND_REF	ORD_PO_NUM	ORD_TOT_COST	ORD_STATUS	ORD_FUND_TYPE	USER_ID
*	121	08-Feb-2002	PCJUN	320021	21234	\$17,401.23	OPEN	CIS	352-14-5875
*	122	08-Feb-2002	PCJUN	320213	21234	\$1,479.15	PAID	CIS	352-14-5875
*	123	10-Feb-2002	COMPU	320322	21234	\$1,788.45	PAID	CIS	352-14-5875
*	124	10-Feb-2002	PCPAL	000000	21234	\$2,430.45	REC	BUS	264-77-4567
*	125	14-Feb-2002	COMPU	320345	21234	\$650.98	REC	CIS	352-14-5875

Рис. 8.17. Пример данных таблицы ORDER

Сторона "M" связи между заказами и их компонентами будет храниться в таблице ORDER_ITEM. Поэтому каждый заказ (ORDER) ссылается на одну или более записей ORDER_ITEM, но каждая запись ORDER_ITEM ссылается на единственную запись в таблице ORDER. (Мы рассматривали эту проблему в *разд. 2.8* при обсуждении выписки счетов.) Структура сущности ORDER_ITEM представлена в табл. 8.20.

Таблица 8.20. Сущность ORDER_ITEM

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внешний ключ (FK)	Ссылки
ORD_ID	Идентификационный код заказа		PK, FK	ORDER
OI_LINE	Номер строки заказа		PK	
VEND_ID	Идентификационный код поставщика		FK	VENDOR
TY_GROUP	Группа инвентаря			
OI_DESCRIPTION	Описание позиции			
OI_UNIT_COST	Стоимость единицы			
OI_QTY_ORD	Количество			
OI_COST	Общая стоимость в строке	D		
OI_QTY_RECVD	Полученное количество			
OI_DATE_IN	Дата последнего по- лучения			


Использование двух сущностей — ORDER и ORDER_ITEM — дает некоторые дополнительные преимущества.

- Сущность ORDER_ITEM содержит атрибут OI_LINE, представляющий заказ, в котором INV_TYPE (позиции) действительно введены. Если вместо этого использовать атрибуты ORD_ID и TY_GROUP, любые изменения приведут к появлению разных распечаток каждый раз при добавлении в заказ новой позиции. В этом случае если бы конечный пользователь просматривал предыдущий заказ, то расположение позиций в нем отличалось бы от первоначального, что могло бы вызвать затруднения. При сохранении атрибута OI_LINE в ORDER_ITEM эта проблема снимается.
- Атрибут OI_DESCRIPTION имеет особое предназначение. Хотя для чтения описания из INV_TYPE можно использовать прикладное программное обеспечение, когда конечный пользователь вводит код TY_GROUP, мы все еще можем изменить это описание, чтобы пояснить некоторые детали заказа (см. пример данных на рис. 8.18). Поскольку данное описание не будет выполнять роль внешнего ключа, избыточность не вызовет структурных проблем. Кроме того, подробная информация может впоследствии оказаться очень полезной при возникновении вопросов по данному заказу.

Пример заполнения таблицы ORDER_ITEM представлен на рис. 8.18.

ORD_ID	OI_LINE	TY_GROUP	OI_DESCRIPTION	OI_UNIT_COST
121	1	HMNCTRAM	Token ring card, NIC 4M (BAS112B edition)	\$184.23
121	2	HMPCDTP3	DELL Dimension 4100, 80GB HD, 1.0GB RAM, 26 dpi SVGA	\$2,395.00
121	3	HMPCLTP3	DELL Inspiron 4000, 20GB HD, 128MB RAM, 14" XGA display	\$2,587.95
122	1	HMPRLSBL	HP Laserjet 4550, 64MB RAM	\$1,999.99
122	2	SUPPSS11	Laser printer paper, 5,000 sheet box	\$19.95
123	1	SUDK35HD	3.5-in floppy disks, DD/DS, 3M, 25/box, preformatted	\$22.50
123	2	SUPPCO11	Continuous feed paper, green band, perforated 8.5x11	\$38.25
123	3	SUCAJCO	Ink-jet color cartridge, high capacity	\$29.18
123	4	SUCAJBL	Ink-jet black cartridge, high capacity	\$26.00
124	1	SWNMUTXX	Network utilities software, Unicorn version 4.01	\$149.98
124	2	SWSTXXXX	Minitab 13.0, student version	\$49.85
125	1	SUCALPXX	Laser printer cartridge, XIT type	\$71.22

Таблица ORDER_ITEM
(продолжение)



OI_QTY_ORD	OI_COST	OI_QTY_RECVD	OI_DATE_IN
1	\$184.23	1	10-Feb-2002
5	\$11,975.00	2	
2	\$5,175.90	2	10-Feb-2002
5	\$750.80	5	11-Feb-2002
35	\$698.25	35	11-Feb-2002
20	\$450.00	20	17-Feb-2002
9	\$344.25	9	17-Feb-2002
15	\$437.70	0	17-Feb-2002
20	\$520.00	0	17-Feb-2002
1	\$149.99	1	18-Feb-2002
45	\$2,247.75	45	18-Feb-2002
9	\$640.98	9	19-Feb-2002

Рис. 8.18. Пример данных таблицы ORDER_ITEM

С помощью данных ORDER и ORDER_ITEM, представленных на рис. 8.17 и 8.18 соответственно, вы можете легко проследить все заказы и их компоненты. Например, если посмотреть в таблице ORDER на атрибут ORD_ID = 121 и найти его в таблице ORDER_ITEM, то можно прийти к следующим выводам:

- ❑ заказ состоит из трех позиций: одна сетевая карта типа Token Ring, пять настольных компьютеров Dell Pentium III и два ноутбука Dell Pentium III. (Обратите внимание, что номера OI_LINE в ORDER_ITEM находятся в диапазоне от 1 до 3 для ORD_ID = 121.);
- ❑ заказ был инициирован пользователем 352-14-5875 в субботу 6 февраля 2002 года у поставщика PCJUN;
- ❑ заказ все еще открыт, поскольку вторая строка в ORDER_ITEM (OI_LINE = 2) показывает, что получены только два из пяти компьютеров. (Обратите внимание, что OI_DATE_IN в ORDER_ITEM имеет пустое значение.)

Учет инвентаря в лаборатории UCL является непростой задачей, поскольку здесь имеют место два различных типа транзакций. Некоторые объекты, такие как бумага,

чернильные картриджи и другие расходные материалы, выдаются со склада по мере необходимости. Например, если преподавателю необходима пачка бумаги для занятий в аудитории или для офиса, запас "пачки бумаги" просто уменьшается на единицу. Однако преподаватель и сотрудник могут также взять объект на время (под отчет), например, индикаторную панель для занятий или ноутбук для демонстрационных целей. В этом случае взятый под отчет объект остается на учете, но его статус и местоположение изменяются. Нам необходимо иметь возможность отслеживания всех пользователей объектов, а также их местоположение. При возврате объекта другая транзакция (регистрация) вновь меняет его статус и местоположение.

Начнем изучение транзакций учета на примере простой выдачи материальных средств. Следующий сценарий описывает четыре транзакции, записанные под номерами 325, 326, 327 и 328. (Можно определить компоненты транзакции, изучая представленный выше пример данных. Например, исследуя таблицу USER, мы узнаем, что пользователь 299-87-9234 является преподавателем факультета CIS. В таблице ITEM можно найти позицию 4238131, а из таблицы INV_TYPE следует, что эта позиция представляет собой картридж для лазерного принтера и т. д.).

- Транзакция 325: преподаватель факультета CIS 299-87-9234 получил картридж для лазерного принтера (ITEM_ID = 4238131) из компьютерной лаборатории (KOM245A-1) в четверг 4 февраля 2002 года.
- Транзакция 326: сотрудник факультета CIS 352-14-5875 получил три картриджа для лазерного принтера из KOM245A-1, пять пачек простой бумаги 8,5" × 11" (ITEM_ID = 3154567) из KOM245B-1 и две коробки дискет 3,5" (ITEM_ID = 4238132) из KOM245A-1 в четверг 4 февраля 2002 года.
- Транзакция 327: сотрудник факультета CIS 352-14-5875 получил одну коробку дискет 3,5" из KOM106-1 в воскресенье 7 февраля 2002 года.
- Транзакция 328: преподаватель факультета CIS 255-67-4567 получил одну пачку простой бумаги 8,5" × 11" из KOM106-1 и один картридж с черными чернилами для струйного принтера (ITEM_ID = 4238130) из KOM106-1 в понедельник 8 февраля 2002 года.

Перед тем как проследить выполнение этих транзакций, необходимо установить, какие таблицы базы данных в них участвуют. В настоящее время мы не можем проследить эти транзакции, поскольку наша база данных отражает процесс получения инвентаря как связь M:N между сущностями USER и ITEM, представленную на рис. 8.19, панель А. Например, идентифицированный пользователь может получить несколько пачек бумаги, и пачка бумаги может быть взята со склада любым количеством идентифицированных пользователей.

Поскольку связь M:N нельзя реализовать должным образом в проекте реляционной базы, прежде всего, нужно преобразовать связь "получение" в составную сущность WITHDRAW (получение), представленную на рис. 8.19, панель В. Структура этой сущности представлена в табл. 8.21.

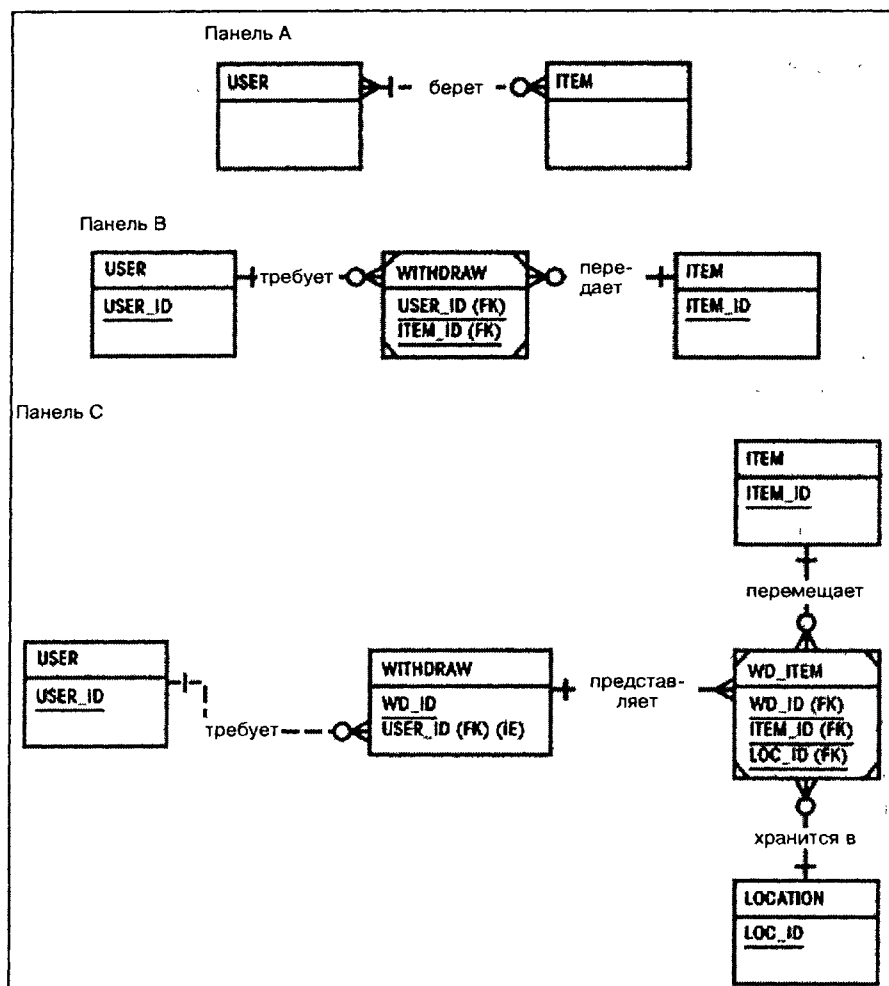


Рис. 8.19. Исправление сущности WITHDRAW

Таблица 8.21. Сущность WITHDRAW

Имя атрибута	Содержимое	Тип атрибута: составной (С), производный (D) или много- значный (М)	Первичный ключ (РК) и/или внешний ключ (FK)	Ссылки
WD_DATE	Дата получения		РК	
USER_ID	Идентификатор поль- зователя (преподава- тель или сотрудник)		РК, FK	USER

Таблица 8.21 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
ITEM_ID	Идентификатор полу- ченного объекта	M	PK, FK	ITEM
LOC_ID	Идентификатор ме- стонахождения	M	PK, FK	LOCATION
WD_QTY	Количество получен- ных предметов	M		

Сущность WITHDRAW в табл. 8.21 как будто бы имеет все необходимые атрибуты. Кроме того, на панели В рис. 8.19 видно, что добавление сущности WITHDRAW несомненно преобразует связь M:N между сущностями USER и ITEM в два набора связей 1:M. Но все же, несмотря на очевидное улучшение, сущность WITHDRAW не достаточно хорошо исполняет свои функции. Хотя ее компоненты помогают связать сущности USER, ITEM и LOCATION, она содержит три многозначных атрибута. Чтобы их устранить, мы можем разбить сущность WITHDRAW в табл. 8.21 на две сущности, представленные на панели С рис. 8.19.

Руководствуясь панелью С рис. 8.19, мы можем преобразовать сущность WITHDRAW и устранить многозначные атрибуты, поместив их в сущность WD_ITWEM. Эти две сущности представлены в табл. 8.22 и 8.23, а примеры данных для этих таблиц представлены на рис. 8.20 и 8.21 соответственно (эти данные как раз и описывают сценарий, представленный в начале наших рассуждений).

WD_ID	WD_DATE	USER_ID
325	04-Feb-2002	299-87-9234
326	04-Feb-2002	352-14-5875
327	07-Feb-2002	352-14-5875
328	08-Feb-2002	255-67-4567

Рис. 8.20. Пример данных таблицы WITHDRAW

Таблица 8.22. Исправленная сущность WITHDRAW

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
WD_ID	Идентификатор полу- чения		PK	

получения. Различие между процессами получения (WITHDRAW) и выдачи под отчет (CHECK_OUT) состоит в том, что в последнем случае выполняются две транзакции для каждого объекта: одна при выдаче объекта под отчет и другая при его возврате.

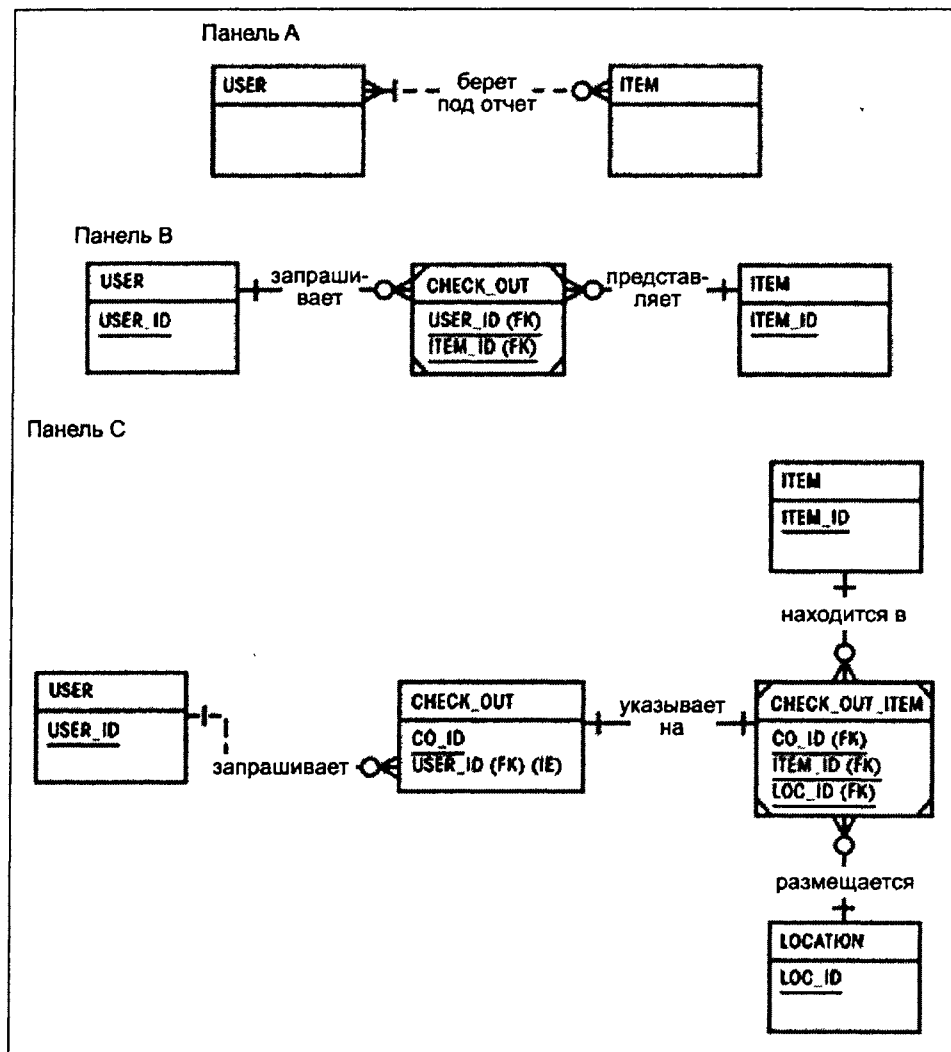


Рис. 8.22. Исправление сущности CHECK_OUT

Поскольку исправление процесса выдачи под отчет в основном схоже с исправлением процесса получения, мы просто сошлемся на панели А и В рис. 8.22 без обсуждения деталей процесса корректировки.

Таблица 8.22 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
WD_DATE	Дата получения			
USER_ID	Идентификатор поль- зователя (препода- ватель или сотрудник)		FK	USER

Таблица 8.23. Сущность (слабая) WD_ITEM

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
WD_ID	Идентификатор полу- чения		PK, FK	WITHDRA W
ITEM_ID	Идентификатор полу- чаемого предмета		PK, FK	ITEM
LOC_ID	Идентификатор ме- стоположения		PK, FK	LOCATION
WD_QTY	Количество получен- ных предметов			

	WD_ID	ITEM_ID	LOC_ID	WI_QTY
▶	325	4238131	KOM245A-1	1
	326	4238131	KOM245A-1	3
	326	3154567	KOM245B-1	5
	326	4238132	KOM245A-1	2
	327	4238132	KOM106-1	1
	328	3154567	KOM106-1	1
	328	4238130	KOM106-1	1

Рис. 8.21. Пример данных таблицы WD_ITEM

Сущности WITHDRAW и ITEM могут обеспечить транзакции получения, поэтому мы можем включить эту редакцию в окончательный проект. Однако далее мы разрабатываем другую версию для стандартизации всех транзакций инвентарного учета.

Транзакции выдачи под отчет обрабатываются теми же процессами, что и транзакции получения, поэтому на рис. 8.22 представлен исправленный вариант проекта этих транзакций, который является зеркальным отражением проекта транзакций

Табл. 8.24, определяющая структуру сущности CHECK_OUT, построена на основе панели С рис. 8.22.

Таблица 8.24. Сущность CHECK_OUT

Имя атрибута	Содержимое	Тип атрибута: составной (С), производный (D) или мно- гозначный (M)	Первичный ключ (РК) и/или внеш- ний ключ (FK)	Ссылки
CO_ID	Идентификатор операции выдачи под отчет		РК	
CO_DATE	Дата выдачи под отчет			
USER_ID	Идентификатор пользо- вателя		FK	USER

Чтобы проверить процесс выдачи под отчет в действии, проследим следующие транзакции (вы также увидите различие между процессами получения и выдачи под отчет при рассмотрении структуры сущности CHECK_OUT_ITEM и примеров ее заполнения).

- ❑ CO_ID = 101: преподавателю факультета бухгалтерского учета (301-23-4245) был выдан под отчет ноутбук (4228753) из КОМ245А-1 2 февраля 2002 года, и он вернул его 3 февраля 2002 года.
- ❑ CO_ID = 102: преподавателю факультета CIS (255-67-4567) были выданы под отчет ноутбук (4358255) из КОМ245В-1 и проекционная панель (4358258) из КОМ245В-1 3 февраля 2002 года; 4 февраля 2002 года им возвращен только ноутбук.
- ❑ CO_ID = 103: преподавателю факультета CIS (264-77-032) был выдан под отчет ноутбук (4228753), возвращенный преподавателем факультета бухгалтерского учета в транзакции 101, из КОМ245А-1 3 февраля 2002. Ноутбук пока не возвращен.
- ❑ CO_ID = 104: сотруднику факультета CIS (386-12-3456) выдан под отчет ноутбук (4112151) из КОМ245А-2 4 февраля 2002 года, возвращенный им 5 февраля 2002 года.

Эти транзакции отражены на рис. 8.23 и 8.24.

	CO_ID	CO_DATE	USER_ID
▶	101	02-Feb-2002	301-23-4245
	102	03-Feb-2002	255-67-4567
	103	03-Feb-2002	264-77-0032
	104	04-Feb-2002	386-12-3456

Рис. 8.23. Пример данных таблицы CHECK_OUT

Каждая из записей таблицы CHECK_OUT указывает на сведения о транзакции, т. е. сторону "М", выраженную сущностью CHECK_OUT_ITEM, которая представлена в табл. 8.25.

Таблица 8.25. Сущность (слабая) CHECK_OUT_ITEM

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или многознач- ный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
CO_ID	Идентификатор опе- рации выдачи под отчет		PK	
ITEM_ID	Идентификатор объ- екта		PK, FK	ITEM
LOC_ID	Идентификатор ме- стоположения		PK, FK	LOCATI ON
COI_QTY	Количество выданных под отчет предметов			
COI_DATE_IN	Дата возврата выдан- ных под отчет пред- метов			

После завершения транзакций сущность CHECK_OUT_ITEM выглядит так, как это представлено на рис. 8.24.

	CO_ID	ITEM_ID	LOC_ID	COI_QTY	COI_DATE_IN
▶	101	4228753	KOM245A-1	1	03-Feb-2002
	102	4358255	KOM245B-1	1	04-Feb-2002
	102	4358258	KOM245B-1	1	
	103	4228753	KOM245A-1	1	
	104	4112151	KOM245A-2	1	05-Feb-2002

Рис. 8.24. Пример данных таблицы CHECK_OUT_ITEM

Рис. 8.24 в точности отображает все сведения о выполненных транзакциях. Поскольку предмет 4358258 в транзакции 102 и предмет 4228753 в транзакции 103 еще не возвращены, их атрибуты COI_DATE_IN имеют пустые значения. Однако в следующем разделе вы узнаете, что обработку транзакций учета инвентаря можно улучшить.

8.3. Проверка ER-модели

Итак, к настоящему моменту мы определили следующее:

- ❑ *наборы сущностей, атрибутов и доменов;*
- ❑ *составные атрибуты.* Такие атрибуты могут быть (как правило, так и делается) разбиты на несколько независимых атрибутов;

- *многозначные атрибуты*, реализованные в виде новой сущности, связанной связью 1:M с оригинальной сущностью;
- *первичные ключи*. Обеспечена целостность первичных ключей;
- *внешние ключи*. Обеспечена целостность на уровне ссылок с помощью внешних ключей;
- *производные атрибуты*. Обеспечена возможность вычисления этих атрибутов;
- *составные сущности*. Реализованы с помощью связей 1:M.

Несмотря на то что мы значительно продвинулись вперед, все же до реализации модели нам еще далеко.

Для завершения концептуального проекта UCL мы должны проверить модель. *Проверка* представляет собой увязывание моделирования БД с проектированием, реализацией базы данных и разработкой приложений БД. Следовательно, процесс проверки используется для того, чтобы установить следующее:

- проект надлежащим образом отражает представление конечного пользователя или приложения о базе данных;
- все транзакции БД (вставка, обновление, удаление) определены и смоделированы так, что реализация проекта обеспечит выполнение всех требований к обработке транзакций;
- проект базы данных отвечает всем требованиям к выводу информации (экранные запросы, формы и форматы отчетов). Помните, что информационные требования могут влиять на процесс проектирования;
- обеспечены все необходимые экраны и формы ввода информации;
- проект обладает достаточной гибкостью для выполнения возможных расширений и модификаций.

Несмотря на то что первоначальная ER-диаграмма, представленная на рис. 7.19, была сильно изменена, она до сих пор имеет ряд недостатков:

- некоторые связи не совсем четко определены, или даже вовсе не определены;
- в модели имеется избыточность. Вспомните почти полное сходство между транзакциями получения (WITHDRAW) и выдачи под отчет (CHECK_OUT);
- модель можно расширить с тем, чтобы улучшить семантическую точность и обеспечить лучшее представление операций в действительности;
- модель нужно модифицировать для лучшего соответствия требованиям пользователей (производительность и безопасность).

В последующих разделах этой главы мы продемонстрируем процесс проверки на некоторых приложениях модуля Inventory (процесс проверки должен быть обязательно выполнен для всех модулей системы).

- *Выявление основной сущности (central entity)*. Хотя все транзакции конечных пользователей лаборатории UCL очень важны, управление материально-техническим снабжением имеет высший приоритет с точки зрения администрации. Причина этого проста: государственные аудиторы считают лабораторию и ее дорогостоящее оборудование гарантией уплаты государственных налогов. Неправильный учет материально-технических средств может иметь серьезные

последствия, следовательно, сущность ITEM следует считать основной сущностью данного модуля.

- *Идентификация каждого модуля и его компонентов.* В табл. 8.2 определены модули и их компоненты. Очень важно связать компоненты друг с другом при помощи общих сущностей. Например, несмотря на то что сущность USER принадлежит модулю Lab, а сущность ITEM принадлежит модулю Inventory, они взаимодействуют друг с другом. Например, пользователи (USER) записываются в журнал (LOG) модуля Lab. Пользователь (USER) также явно участвует в операциях получения материалов в модуле Inventory, а также в процессе получения оборудования под отчет.
- *Идентификация требований к транзакциям в каждом модуле.* Мы уделим внимание требованиям лишь к одному отчету в модуле INVENTORY. Предполагается, что вы самостоятельно проверите остальные требования к транзакциям с помощью информации, собранной в *разд. 7.1*. Затем проверьте достоверность выполнения этих требований по отношению к базе данных UCL для всех системных модулей.

При исследовании требований к отчетности модуля выявляются следующие проблемы:

- в модуле Inventory создаются три отчета, один из которых — отчет о движении материальных средств. Но движение материальных средств затрагивает несколько различных сущностей (CHECK_OUT, WITHDRAW и ORDER). Это затрудняет создание отчетов и ухудшает производительность системы в целом.
- имеющееся в наличии количество предметов данного наименования обновляется при покупке, получении со склада, выдаче под отчет, возврате или ревизии инвентаря. В данной модели, однако, представлены только процессы получения (withdraw) и выдачи под отчет (check-out).

Вот какое решение этих проблем предложил проектировщик БД:

Все, что необходимо модулю Inventory, это общая точка входа для всех передвижений инвентаря. Другими словами, система должна отслеживать весь приход инвентаря на склад и уход со склада. Чтобы решить эту задачу, мы должны создать новую сущность для записи движения материальных средств, т. е. нам необходима сущность для описания транзакций, связанных с инвентарем. Мы назовем ее INV_TRANS.

Создание общей точки входа преследует две цели.

- Стандартизация интерфейса модуля Inventory с другими (внешними) модулями. Любое движение материальных средств (поступают они на склад или убывают со склада) инициирует создание записи в этой сущности.
- Управление учетом и оперативное получение необходимой информации, например, отчет о движении инвентаря.

На рис. 8.25 иллюстрируется решение описанных выше проблем.

Сущность INV_TRANS на рис. 8.25 представляет собой простейший журнал регистрации транзакций с инвентарем, в котором будут храниться любые движения материальных средств. Каждая запись в INV_TRANS представляет собой один из двух типов движения: приход (+) или уход (–). Каждая запись INV_TRANS должна содержать строку в таблице TR_ITEM для каждого прихода, ухода, для каждой выдачи под отчет или возврата.

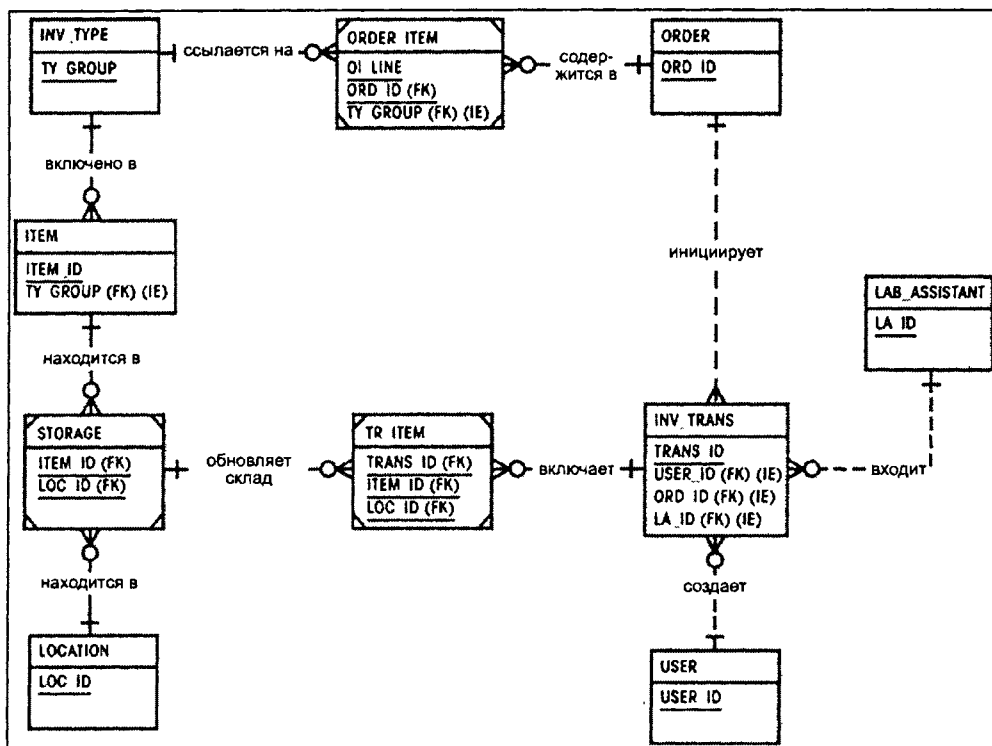


Рис. 8.25. Транзакции в модуле Inventory

Сущность INV_TRANS также дает нам возможность более разумно организовать передвижение инвентаря. Например, при получении заказанного оборудования создается транзакция инвентаря (+). Эта запись в INV_TRANS обновит значение атрибута OI_QTY_RECVD сущности ORDER_ITEM в модуле Inventory. Затем этот модуль создает запись транзакции (-) для регистрации выдачи, а затем еще одну запись транзакции (+) для регистрации возврата оборудования пользователем. При получении материалов со склада для регистрации выданной позиции также создается запись транзакции (-). Эти связи отображены на рис. 8.25.

Атрибуты новой сущности INV_TRANS представлены в табл. 8.26.

Таблица 8.26. Сущность INV_TRANS

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
TRANS_ID	Идентификатор тран- закции инвентаря		PK	

Таблица 8.26 (окончание)

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или много- значный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
TRANS_TYPE	Тип транзакции: I = приход O = уход			
TRANS_PURP OSE	Причина транзакции: PO = заказ на поставку CC = выдача под отчет WD = выдача AD = коррекция со- стояния склада			
TRANS_DATE	Дата транзакции			
LA_ID	Ассистент, выполни- вший запись транзакции		FK	LAB_AS SISTANT
USER_ID	Лицо, инициировавшее транзакцию		FK	USER
ORDER_ID	Идентификационный номер заказа		FK	ORDER
TRANS_ COMMENT	Комментарий			

Чтобы посмотреть, как работает сущность INV_TRANS (табл. 8.26), обратитесь к фрагменту ER-диаграммы на рис. 8.25 и убедитесь, что ее строки содержатся в слабой сущности TR_ITEM. Рис. 8.25 также показывает, что все передвижения материальных средств теперь можно легко проследить. Например, любая позиция должна где-то храниться, поэтому ее местоположение можно получить из сущности STORAGE. Поскольку INV_TRANS связана и с LABASSISTANT, и с USER, мы знаем, кто выполнял запись транзакций и кто ее инициировал. На рис. 8.26 представлен пример данных, которые вы можете отслеживать:

- ☐ транзакции получения, впервые представленные на рис. 8.20 и 8.21;
- ☐ транзакции выдачи под отчет и возврата, впервые представленные на рис. 8.23 и 8.24;
- ☐ покупка двух лазерных принтеров HP и 35 пачек бумаги.

Например, первая строка таблицы INV_TRANS позволяет нам узнать, что 4 февраля 2002 года пользователь 299-87-9234 взял со склада картридж для лазерного принтера.

Транзакция была записана ассистентом 387-99-9565 и эта транзакция уменьшила содержимое склада (TRANS TYPE = O).

	TRANS_ID	TRANS_TYPE	TRANS_PURPOSE	TRANS_DATE	LA_ID	USER_ID	ORDER_ID	TRANS_COMMENT
+	325	O	WO	04-Feb-2002	387-99-9565	239-87-9234		Laser printer cartridge
+	326	O	WO	04-Feb-2002	451-25-6817	352-14-5875		Laser printer cartridge
+	327	O	WO	07-Feb-2002	387-99-9565	352-14-5875		Box of 20 3.5" floppy disks, HD/DS
+	328	O	WO	08-Feb-2002	451-25-6817	255-67-4567		Two reams of 8.5x11" paper & ink-jet cartridge
+	401	O	CC	02-Feb-2002	451-25-6817	301-23-4245		Laptop check-out
+	402	I	CC	03-Feb-2002	452-11-0094	301-23-4245		Laptop returned
+	403	O	CC	03-Feb-2002	387-99-9565	255-67-4567		Laptop & projector check-out
+	404	O	CC	03-Feb-2002	452-11-0094	264-77-0032		Laptop check-out
+	405	I	CC	04-Feb-2002	451-25-6817	255-67-4567		Laptop returned
+	406	O	CC	04-Feb-2002	457-34-5678	386-12-3456		Laptop check-out
+	407	I	CC	05-Feb-2002	387-99-9565	386-12-3456		Laptop returned
+	408	I	PO	11-Feb-2002	451-25-6817	352-14-5875	122	Two HP printers, 35 boxes of paper order

Рис. 8.26. Пример данных таблицы INV_TRANS

Сведения о транзакциях, представленных на рис. 8.26, хранятся в таблице TR_ITEM, поэтому перед тем как исследовать эти сведения, мы должны рассмотреть структуру сущности TR_ITEM, представленную в табл. 8.27.

Таблица 8.27. Сущность (слабая) TR_ITEM

Имя атрибута	Содержимое	Тип атрибута: составной (C), производный (D) или мно- гозначный (M)	Первичный ключ (PK) и/или внеш- ний ключ (FK)	Ссылки
TRANS_ID	Идентификатор транзакции инвентаря (генерируется системой в сущности INV_TRANS)		PK, FK	INV_TRANS
ITEM_ID	Идентификатор объекта		PK, FK	ITEM
LOC_ID	Идентификатор местоположения		PK, FK	LOCATION
TRANS_QTY	Количество полученных предметов			

Исследуя пример данных, представленный на рис. 8.27, мы можем отследить сведения о транзакции рис. 8.26.

Например, мы видим, что в первой строке таблицы INV_TRANS атрибут TRANS_ID = 325 (см. рис. 8.26) указывает на атрибут TRANS_ID = 325 сущности TR_ITEM, представленной на рис. 8.27, и делаем вывод, что эта транзакция повлекла за собой получение одного предмета 4238131 (картридж для лазерного принтера). То, что предмет 4238131 является именно картриджем для лазерного принтера, мы узнаем из таблиц ITEM и INV_TYPE, представленных на рис. 8.11 и 8.12, а также из

того, что ITEM_ID = 4238131 соответствует TY_GROUP = SUCALPXX. В транзакцию 326 вовлечены три предмета, поэтому таблица TR_ITEM содержит три строки сведений для этой транзакции.

TRANS_ID	ITEM_ID	LOC_ID	TRANS_QTY
325	4238131	KOM245A-1	1
326	3154567	KOM245B-1	5
326	4238131	KOM245A-1	3
326	4238132	KOM245A-1	2
327	4238132	KOM106-1	1
328	3154567	KOM106-1	1
328	4238130	KOM106-1	1
401	4228753	KOM245A-1	1
402	4228753	KOM245A-1	1
403	4358255	KOM245B-1	1
403	4358258	KOM245B-1	1
404	4228753	KOM245A-1	1
405	4358255	KOM245B-1	1
406	4112151	KOM245A-2	1
407	4112151	KOM245A-2	1
408	3154567	KOM245A-2	35
408	4567920	KOM245B-2	1
408	4567921	KOM245B-2	1

Рис. 8.27. Пример данных таблицы TR_ITEM

Посмотрим, как обрабатывается выдача под отчет и возврат оборудования. На рис. 8.26 транзакция INV_TRANS 401 записывает TRANS_PURPOSE = CC и TRANS_TYPE = O, что указывает на операцию выдачи под отчет. Эта транзакция вызвала следующие записи: выдача под отчет ноутбука, ITEM_ID = 4228753 (см. рис. 8.21) 2 февраля 2002 года преподавателю факультета бухгалтерского учета, USER_ID = 301-23-4245. Ноутбук был возвращен 3 февраля 2002 года, и эта транзакция была записана под номером TRANS_ID = 402, TRANS_PURPOSE = CC и TRANS_TYPE = I, что говорит о том, что этот ноутбук был возвращен на склад. Кстати, поскольку в наличии имеется несколько ноутбуков, преподаватели могут не дожидаться возврата ноутбука для его получения под отчет, пока на складе имеются ноутбуки. Однако если все ноутбуки на руках, система может отследить, кто их взял и когда должен вернуть. Если директор захочет ограничить время пользования оборудованием, в этом проекте можно легко организовать оповещение пользователей о необходимости возврата оборудования.

В окончательной ER-диаграмме учтены все сделанные нами изменения. Несмотря на то что исходная ER-диаграмма более проста для понимания с точки зрения пользователя, полученная нами ER-диаграмма гораздо более полезна с процедурной точки зрения. Например, проделанные изменения абсолютно невидимы пользователю, поскольку пользователю недоступна сущность INV_TRANS. Окончательный вид ER-диаграммы представлен на рис. 8.28.

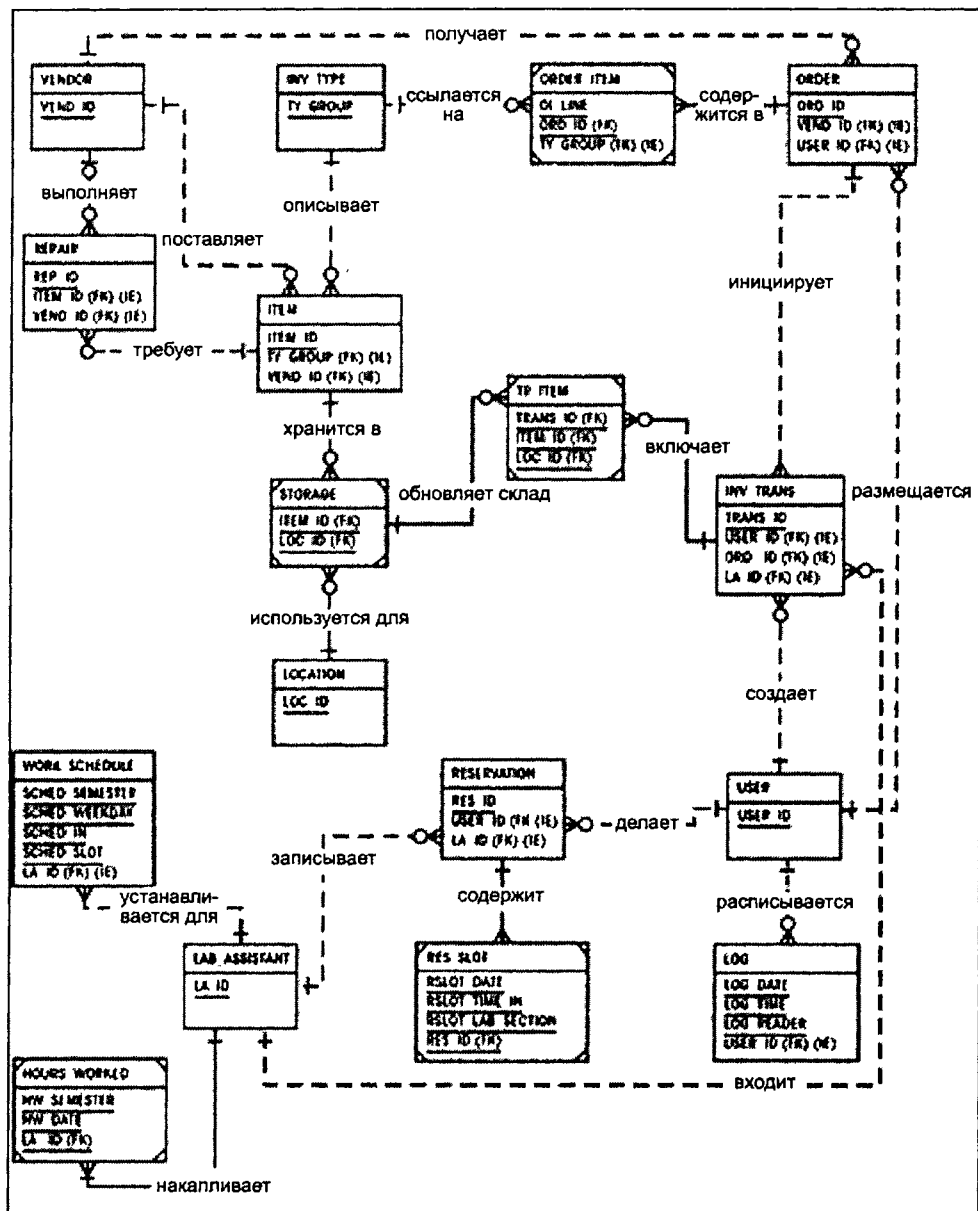


Рис. 8.28. Доработанная ER-диаграмма
Университетской Компьютерной Лаборатории

8.4. Логическое проектирование

По завершении этапа концептуального проектирования наша ER-диаграмма (на концептуальном уровне) отражает все бизнес-правила, которые в свою очередь определяют атрибуты, связи, обязательность, связность, мощность и ограничения. (Помните, что некоторые элементы проекта невозможно смоделировать, поэтому они реализуются на уровне приложений. Например, ограничение "оборудование, выданное под отчет, должно быть возвращено в течение 5 дней" не отражено в ER-диаграмме.) Кроме того, в концептуальную модель включены определения атрибутов, описывающие каждую сущность, что определяется информационными требованиями.

Следует помнить, что сущности концептуальной модели перед их реализацией должны быть нормализованы. Процесс нормализации может привести к появлению дополнительных сущностей и связей, что потребует изменений в начальной ER-диаграмме. Поскольку нам необходимо было сосредоточиться на проверке концептуального проекта с целью получения пригодного для реализации проекта, мы параллельно выполняли и проектирование, и нормализацию. На самом деле, так происходит и в действительности. Однако вы должны помнить из гл. 6, что процесс логического проектирования использовался для трансформирования концептуального проекта во внутреннюю модель выбранной СУБД. Процесс нормализации помогает установить надлежащие атрибуты, их характеристики и их домены, нормализация является переходом к этапу логического проектирования. Тем не менее, поскольку процесс концептуального моделирования также не исключает определение атрибутов, вы можете сделать вывод, что нормализация может выполняться и на этапе концептуального моделирования, и на этапе логического моделирования.

Еще раз повторим, что логическое проектирование трансформирует концептуальную модель в формат, пригодный для конкретной СУБД, которая будет использоваться при реализации системы. Поскольку мы будем использовать реляционную модель, этап логического проектирования состоит в создании реляционных табличных структур, индексов и представлений.

8.4.1. Таблицы

Следующие несколько примеров иллюстрируют проектирование логической модели с помощью SQL. (Необходимо убедиться, что таблицы отражают структуру ER-модели и подчиняются правилам внешнего ключа, если программное обеспечение вашей СУБД позволяет определять внешние ключи.)

Структуры таблиц в проектируемой базе данных можно создать с помощью SQL. Например, таблица STORAGE может быть создана с помощью такого оператора:

```
CREATE TABLE STORAGE (  
    LOC_ID CHAR(12) NOT NULL,  
    ITEM_ID CHAR(10) NOT NULL,  
    STOR_QTY NUMBER,  
    PRIMARY KEY (LOC_ID, ITEM_ID),
```

```

FOREIGN KEY (LOC_ID) REFERENCES LOCATION
    ON DELETE RESTRICT
    ON UPDATE RESTRICT,
FOREIGN KEY (ITEM_ID) REFERENCES ITEM
    ON DELETE CASCADE
    ON UPDATE CASCADE);

```

Большинство СУБД теперь используют интерфейсы, позволяющие вводить имена атрибутов в заданный шаблон и выбрать желаемые характеристики атрибута в списке. Вы можете даже вводить комментарии, которые будут воспроизводиться на экране, для того, чтобы подсказать пользователю правила ввода. Например, предыдущую структуру таблицы STORAGE можно создать в шаблоне Microsoft Access, как показано на рис. 8.29.

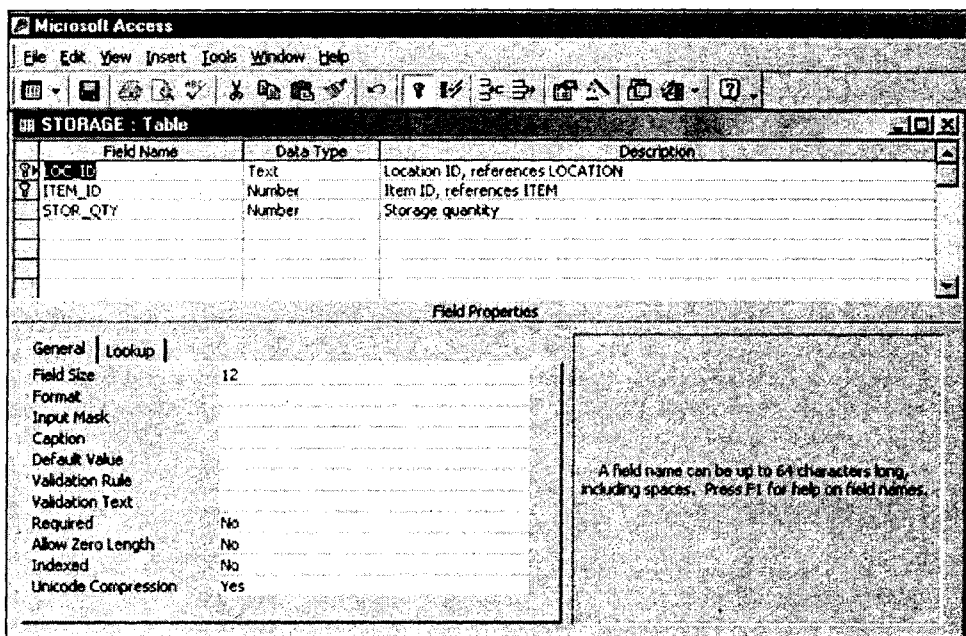


Рис. 8.29. Определение структуры таблицы STORAGE в Microsoft Access

Когда все необходимые для проекта таблицы созданы, устанавливаются все необходимые связи. Хороший CASE-инструментарий позволит вам выполнить эти задачи непосредственно в проекте. Например, проект, представленный на рис. 8.28, может быть записан в определенную структуру БД с помощью CASE-инструментария. Преимущества использования CASE состоит в следующем:

- ☐ база данных будет в точности соответствовать проекту;

- все связи будут протестированы инструментарием CASE и обеспечат их логическую корректность и реализуемость. (Например, в Visio для выполнения подобной операции можно использовать функцию "update foreign keys".);
- все определения и характеристики атрибута внешнего ключа будут соответствовать характеристикам и определениям первичного ключа (функция "update foreign keys" в Visio также позволяет выполнить эту операцию).

Независимо от того, каким способом вы трансформировали проект, представленный на рис. 8.28, в соответствующую структуру базы данных, реляционная схема БД должна соответствовать проекту. Например, на рис. 8.30 представлена реляционная схема в формате Access 2000.

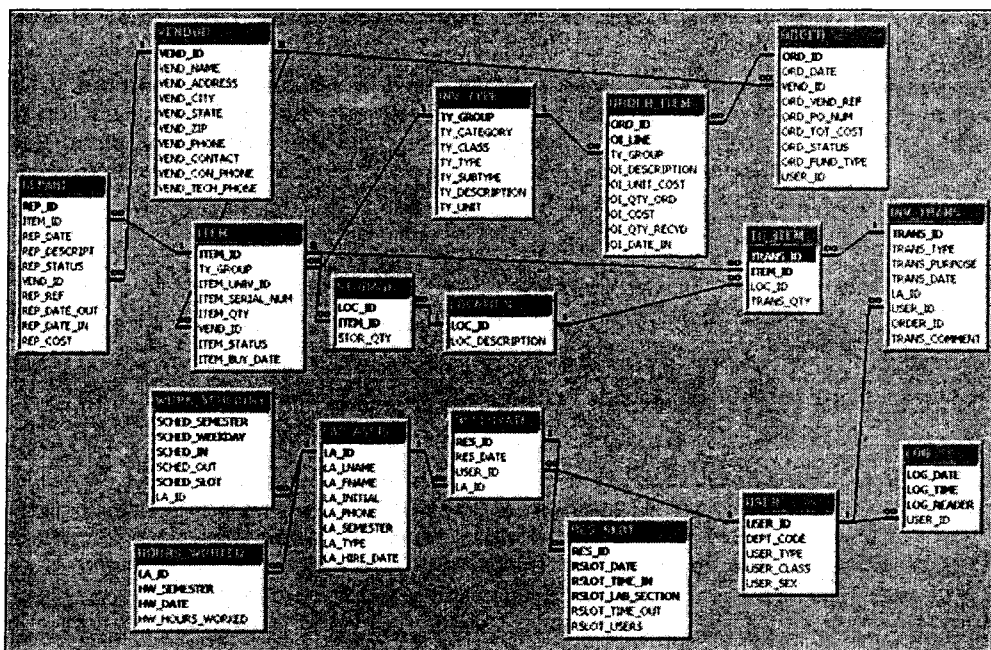


Рис. 8.30. База данных Университетской Компьютерной Лаборатории

На реляционной схеме (рис. 8.30) видно, что все ее таблицы и связи соответствуют проекту (см. рис. 8.28). Обратите внимание, что на схеме показаны дополнительные атрибуты, предназначенные для конечного пользователя и обеспечивающие надлежащее управление данными.

8.4.2. Индексы и представления

На этапе логического проектирования для повышения скорости обработки информации проектировщик может определить необходимые индексы. Индексы также позволяют произвести логическое упорядочивание выводимой информации. Напри-

мер, если необходимо создать график работы ассистентов, представленный в табл. 8.6, следует получить данные из двух таблиц: LAB_ASSISTANT и WORK_SCHEDULE. Поскольку выходной отчет упорядочивается по семестрам, ассистентам и времени, нужно иметь индексы по всем полям первичного ключа в обеих таблицах. С помощью SQL мы можем написать:

```
CREATE UNIQUE INDEX LA_DEX  
ON LAB_ASSISTANT (LA_ID);
```

и

```
CREATE UNIQUE INDEX WS_DEX  
ON WORK_SCHEDULE (SCHED_SEMESTER, LA_ID, SCHED_WEEKDAY, SCHED_IN);
```

Большинство современных СУБД автоматически индексируют таблицы по компонентам первичного ключа.

Представления (см. гл. 5) часто используются для обеспечения безопасности. Они также используются для улучшения производительности системы. Например, ограничения на вывод можно задать с помощью соответствующего представления. Для создания необходимых представлений графика работы ассистентов в весеннем семестре 2002 года можно использовать команду CREATE VIEW:

```
CREATE VIEW LA_SCHED AS  
SELECT LA_ID, LA_NAME, SCHED_WEEKDAY, SCHED_IN, SCHED_OUT  
FROM WORK_SCHEDULE  
WHERE SCHED_SEMESTER='SPRING02';
```

Проектировщик создает представления, необходимые для операций вывода БД.

Примечание

В отличие от других БД, реляционная модель базы данных не требует применять представления для доступа к базе данных. Однако использование представлений обеспечивает безопасность и более эффективный вывод информации.

8.5. Физическое проектирование

Физическое проектирование требует указания специфического места хранения или методов доступа, которые будут использоваться в базе данных (вспомните наш разговор о физическом проектировании в гл. 6). В рамках СУБД физическое проектирование должно включать в себя оценку памяти, необходимой для хранения БД. Эта оценка трансформируется затем в физический размер памяти на конкретном устройстве хранения.

Физические характеристики хранения зависят от СУБД и от используемой операционной системы. Информацию по параметрам хранения можно найти в техническом руководстве по программному обеспечению. Например, в OS/2 DataBase Manager v. 1.2 фирмы IBM необходимую память для создания базы данных (пустой) вы можете оценить по данным, представленным в табл. 8.28.

Таблица 8.28. Дисковое пространство, необходимое для размещения БД в OS/2 DBM v. 1.2

Задача	Дисковое пространство, Кбайт
Фиксированное пространство на одну таблицу в БД	535
17 таблиц × 4 Кбайт на таблицу	68
Всего используется БД	603

Затем необходимо оценить требования к памяти для данных в каждой таблице. В табл. 8.29 представлены расчеты только для таблицы USER.

Таблица 8.29. Требования к физической памяти: таблица USER

Имя атрибута	Тип данных	Требует для хранения (байты)
USER_ID	CHAR(11)	11
DEPT_CODE	CHAR(7)	7
USE_TYPE	CHAR(5)	5
USER_CLASS	CHAR(5)	5
USER_SEX	CHAR(1)	1
Длина строки		29
Число строк		15 950
Общий объем требуемой памяти		462 550

Если СУБД не позволяет автоматизировать процесс определения местоположения хранения и путей доступа, то при физическом проектировании от проектировщика требуются высокое мастерство и точные знания на физическом уровне базы данных, операционной системы и оборудования, используемого БД. К счастью, последние версии программного обеспечения реляционных СУБД позволяют не вникать в сложные проблемы этапа физического проектирования.

Можно хранить БД на единственном томе непрерывного физического пространства; также можно использовать несколько томов, распределяя данные таким образом, чтобы уменьшить время поиска. Некоторые СУБД также позволяют создавать кластерные таблицы и индексы. В *кластерных таблицах* на непрерывном пространстве диска совместно хранятся строки различных таблиц. Такое размещение повышает скорость доступа к данным и в основном используется в связях типа "главный/подчиненный" (master/detail), например, ORDER и ORDER_ITEM или INV_TRANS и TR_ITEM.

Проектировщик БД должен принимать решения, влияющие на скорость доступа к данным, посредством настройки размера буферного пула, размера страничного блока и т. д. Эти решения будут основаны на выборе аппаратной платформы и СУБД.

Обязательно изучите руководство по оборудованию и техническое описание СУБД для выяснения особенностей хранения и методов доступа к данным.

В системе управления компьютерной лабораторией для ускорения доступа к данным также можно создать несколько индексов:

❑ индексы для всех первичных ключей увеличат скорость доступа к данным при создании ссылок на внешний ключ в таблицах. Это автоматически выполняется СУБД;

❑ индексы также могут быть созданы для всех альтернативных ключей поиска. Например, если мы хотим осуществлять поиск в таблице LAB_ASSISTANT по имени пользователя, то нужно создать индекс для атрибута LA_LNAME, например, так:

```
CREATE INDEX LA001 ON LAB_ASSISTANT (LA_LNAME);
```

❑ индексы можно создать для всех вторичных ключей доступа, используемых в отчетах и запросах. Например, отчет о движении материальных средств, вероятнее всего, должен быть отсортирован по типу инвентаря и инвентарному номеру. Поэтому для таблицы ITEM необходимо создать следующий индекс:

```
CREATE INDEX INV002.ON ITEM(TY_GROUP, ITEM_ID);
```

❑ индексы можно создать для всех столбцов, используемых в выражениях WHERE, ORDER BY и GROUP BY оператора SELECT.

8.6. Реализация

Одно из самых значительных преимуществ использования баз данных состоит в том, что БД позволяют пользователям совместно использовать данные. Если данные предназначены для общего использования, а не только для отдельного подразделения предприятия, управление данными становится более специализированной задачей. Среда базы данных приводит к созданию новой организационной структуры, предназначенной для управления ресурсами базы данных. Функции управления БД выполняет администратор БД (DBA). Администратор должен определять стандарты и процедуры, необходимые для взаимодействия с БД (см. гл. 16).

После того как проектировщик завершит этапы концептуального, логического и физического проектирования, DBA принимает соответствующий план реализации. Этот план включает в себя формальное определение процессов и стандартов, которым необходимо следовать при эксплуатации БД, последовательность действий (создание, загрузка, тестирование и оценка), разработку необходимых стандартов на документацию, потребность в документации и точное распределение ответственности при внедрении и обслуживании БД.

Помните, что технические подробности реализации менее всего касаются конечных пользователей. После реализации проекта конечные пользователи должны иметь возможность пользоваться базой данных и ее содержимым (в соответствии с их потребностями и возможностями) без труда и с пользой для дела. Поэтому необходимо проделать большую работу по созданию дружественного интерфейса. На рис. 8.31—8.34 представлены главное меню, выбор в этом меню, примеры ввода данных и окончательная запись для некоторой чартерной компании.

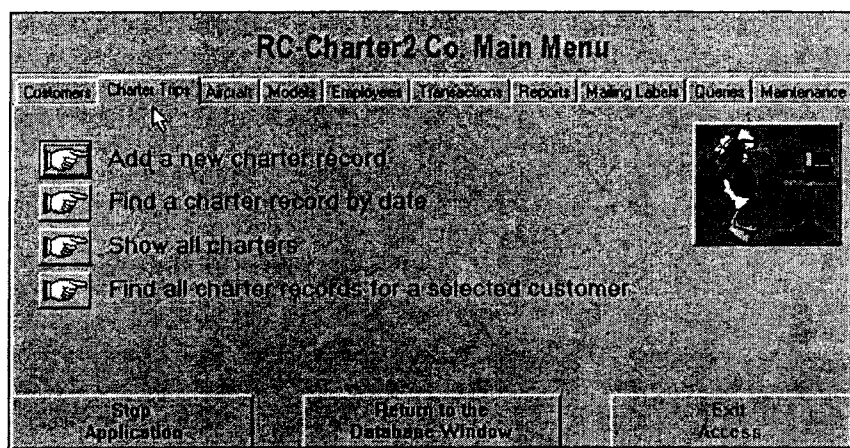


Рис. 8.31. Главное меню базы данных компании RC-Charter2

Trip #: 10296
 Trip date: 27 Jun 2001

RC-Charter2 Co. charter data

Customer: 10011 | Current balance: \$2,962.19 | Credit limit: \$5,000.00 | Aircraft: ☐ Model: ☐

AC num	Model	Chg/mile	Chg/hr.	Cruise (kts)	Fuel/l
14831	BN2A III 3	\$4.15	\$95.00	160	54
37861	BN2A III 3	\$4.15	\$95.00	160	54
22931	C-50A	\$3.17	\$105.50	175	66
33850	EV-580	\$37.55	\$548.75	350	350

Assignment: ☐ Trip # Employee Last Name
 Crew member: ☐ Trip # Employee Last Name

Pax: 0 | Cargo (lbs.): 0 | Payload (lbs.): 0 | Gross T.O. wt. (lbs.): 0 | Center of gravity (in): 0.0

Destination: Distance flown: Fuel used (Gallons): Oil used (qts.): Payment type: ☐

Hobbs out: 0.0 | Hobbs return: 0.0 | Hours flown: 0.0 | Waiting hours: 0.0 | Check on CC #: ☐

Charge/Mile	\$0.00	Mileage charge	\$0.00	Charge Subtotal	\$0.00	Amount paid	\$0.00
Charge/Waiting hour	\$0.00	Waiting charge	\$0.00	Tax	\$0.00	Balance owed	\$0.00
				Total charge	\$0.00		

Рис. 8.32. Новый чартерный рейс компании RC-Charter2

Обратите внимание, что в интерфейсе конечного пользователя, представленном на этих рисунках, используются некоторые специальные технические приемы для обеспечения удобного ввода данных.

- **Раскрывающиеся списки**, ограничивающие ввод. На рис. 8.32 вы видите, что пользовательские данные уже введены. В данном случае в раскрывающемся списке **Customer** (Клиент) был выбран номер 10011. Раскрывающийся список открывается нажатием кнопки со стрелкой, расположенной справа от поля ввода (естественно,

если клиент новый; то запись о нем должна быть создана до этого). Обратите внимание, что финансовая информация клиента, представленная после выбора клиента в списке, позволяет конечному пользователю санкционировать расходы или затребовать полную оплату чартерного рейса. Также щелчок на кнопке со стрелкой, расположенной справа от поля ввода **Aircraft** (самолет) раскрывает список всех доступных типов самолетов и соответствующей информацией о самолете каждого типа. Такие возможности позволяют конечным пользователям отвечать на некоторые вопросы пользователей, не выходя из экрана ввода.

Выбор экипажа

Assignment:

Crew member:

3 Loadmaster

4 Check pilot

Назначение экипажа

Assignment:

Crew member:

Trip #	Employee	Last Name	First Name	Initial	Job Code	Job Description
10296	101	Lewis	Rhonda		3	Dispatcher
10296	105	Williams	Robert	D	1	Part 135 Pilot

Рис. 8.33. Пример ввода данных по чартерному рейсу

HC-Charter2 Co. charter data

Trip #: Trip date:

Customer: Current balance: Credit limit: Aircraft: Model:

Trip #	Employee	Last Name	First Name	Crew Assignment
10296	101	Lewis	Rhonda	Pilot in Command
10296	105	Williams	Robert	Copilot

Pax: Cargo (lbs.): Payload (lbs.):

Gross T.O wt. (lbs.): Center of gravity (in.):

Destination: Distance flown: Fuel used (Gallons): Oil used (qts.):

Hobbs out: Hobbs return: Hours flown: Waiting hours:

Charge/Mile: Mileage charge: Charge Subtotal:

Charge/Waiting hour: Waiting charge: Tax:

Total charge:

Payment type: Check or CC #:

Amount paid: Balance owed:

Records: of 17

Рис. 8.34. Окончательная запись о рейсе

- *Автоматическое завершение ввода данных* на основе сделанного выбора. Например, после выбора типа самолета в раскрывающемся списке все расходы рассчитываются системой, что предотвращает ошибки расчетов конечных пользователей (например, полетное время рассчитывается как **Hours flown = Hobbs return — Hobbs out**. Измеритель Хоббса — это инструмент для записи полетного времени). Точно так же после ввода размера оплаты автоматически рассчитывается баланс и выводится в поле **Balance owed** (Кредит).

Примечание

Помните, что даже самый красивый интерфейс не сможет покрыть недостатки проектирования. К сожалению, очень многие организации пытаются с помощью разработки приложений преодолеть ограничения, вызванные плохим проектом БД. Такие попытки неизбежно приведут к ошибкам в организации данных предприятия (вновь прибегая к архитектурной аналогии, можно сказать, что невозможно преодолеть недостатки плохого проекта здания, наняв отличных каменщиков, — все равно у вас будет плохое здание, хотя и с красивой кладкой). Об этом нужно помнить всегда!

По мере того, как организации все более ориентируются на Интернет, большинство транзакций базы данных становятся все более привязанными к Web-интерфейсу. Мы продемонстрируем создание такого интерфейса в гл. 14.

8.6.1. Создание базы данных

На этом этапе создаются все таблицы, индексы и представления, определенные на этапе логического проектирования. Здесь иницируются команды (или используются программные утилиты) для создания места хранения и методов доступа, определенные на этапе физического проектирования.

8.6.2. Загрузка базы данных и конвертирование

Вновь созданная БД содержит (все еще пустые) табличные структуры. Эти структуры могут заполняться путем ввода данных (с клавиатуры) в каждую таблицу или копированием данных из существующей базы данных или файлов. Если структура таблиц и форматы новой БД несовместимы со старой БД или системой файлов, то процедура копирования потребует использования специальных утилит преобразования.

Поскольку многие процессы требуют точной последовательности, данные загружаются в определенном порядке. Из-за требований к внешнему ключу база данных Университетской Компьютерной Лаборатории должна загружаться в следующей последовательности:

1. Пользователь, поставщик и данные о местоположении.
2. Ассистенты и график работы.
3. Данные о типе инвентаря.
4. Данные об объектах.

После того как загружены эти основные сущности, система готова к тестированию и оценке.

8.6.3. Системные процедуры

Системные процедуры описывают шаги, необходимые для управления, доступа и обслуживания системы базы данных. Эти процедуры параллельно разрабатываются на ранних этапах системного проектирования.

Хорошо организованная структура базы данных требует использования строгих стандартов и процедур для обеспечения надлежащего управления хранилищем данных. Хотя технические и организационные действия логически связаны, важно точно определить отдельные управленческие действия и процедуры.

Для системы управления Университетской Компьютерной Лабораторией должны быть определены следующие процедуры:

- ☐ тестирование и оценка базы данных;
- ☐ настройка базы данных;
- ☐ обеспечение безопасности данных и их целостности;
- ☐ доступ, использование системы базы данных.

В одной конфигурации может существовать несколько баз данных. Каждая БД должна иметь собственный набор системных процедур и оцениваться по его соответствию информационной системе организации.

8.7. Тестирование и оценка

Цель тестирования и оценки состоит в определении того, насколько база данных соответствует своему назначению. Хотя тестирование и оценка представляют собой отдельный этап, реализация, тестирование и оценка базы данных выполняются совместно и тесно связаны. Тестирование и оценка БД должны проводиться постоянно. Вполне приемлемая сегодня база данных может нас совсем не устроить через несколько лет из-за быстрой смены информационных требований. Фактически одна из важнейших причин доминирования баз данных — их гибкость (поскольку таблицы реляционной БД независимы друг от друга, изменения можно выполнять достаточно быстро и эффективно).

8.7.1. Оценка производительности

Производительность связана с возможностью извлечения информации за разумное время и по разумной цене. На производительность системы базы данных могут влиять такие факторы, как скорость линий коммуникации, количество параллельно работающих пользователей, ограничение по ресурсам и т. д. Производительность прежде всего определяется временем отклика базы данных на запрос и в общем случае оценивается путем вычисления количества транзакций в секунду.

На производительность базы данных UCL влияет тип процессора, используемого на сервере базы данных, размер доступной памяти на сервере, а также компьютеры рабочих станций, скорость сетевых коммуникаций и количество одновременно работающих пользователей.

Некоторые меры могут увеличить производительность системы управления лабораторией UCL. Помните, что на физическом уровне любая комбинация СУБД/оборудование имеет свои специфические преимущества. Например, мы можем использовать следующие методы:

- ☐ определить конфигурационные параметры системы для повышения производительности. Например, можно:
 - ограничить нагрузку на доступные ресурсы компьютера, уменьшив максимальное количество одновременно действующих баз данных;
 - повысить эффективность запросов, увеличив размер совместно используемой памяти БД;
- ☐ увеличить размер рабочей области, используемой в выражениях ORDER BY в операторе SELECT, изменив параметр разделяемого сегментирования БД;
- ☐ повысить производительность за счет реорганизации БД;
- ☐ повысить производительность, настроив параметры локальной сети и увеличив размер оперативной памяти (RAM) на каждом компьютере локальной сети.

Проектировщик должен настроить БД и саму систему так, чтобы добиться приемлемой производительности. Помните о разнице между приемлемой производительностью и оптимальной производительностью. Производительность — не единственный фактор, определяющий качество базы данных, и не обязательно самый важный. На самом деле обеспечение управления параллельным выполнением операций, целостность данных, безопасность, резервное копирование и восстановление могут оказаться важнее всех требований к производительности.

8.7.2. Меры по обеспечению безопасности

Меры по обеспечению безопасности предназначены для гарантии надежного хранения данных. Безопасность особенно важна в многопользовательских базах данных, когда кто-нибудь может сознательно ввести некорректную информацию. Администратор БД должен выработать (с помощью конечных пользователей) информационную политику компании, определяющую способ хранения данных, доступ и управление в рамках безопасности данных и обеспечения конфиденциальности.

Доступ может быть ограничен при помощи прав доступа или авторизации доступа. Такие права устанавливаются в зависимости от потребностей пользователя в информации или уровня ответственности пользователя. Для базы данных UCL права доступа должны быть установлены для ассистентов, директора и секретаря. Но эти права ограничены. Например, ассистенты могут читать свой график работы, но не могут изменять данные, хранящиеся в таблицах LAB_ASSISTANT или HOURS_WORKED.

Администратор БД может, например, предоставить право использования предварительно созданного представления LA_SCHED ассистенту лаборатории Anne Smith при помощи следующего оператора SQL:

```
GRANT SELECT ON LA_SCHED TO LA_ASMITH;
```

В данном случае пользоваться представлением LA_SCHED для проверки графиков работы ассистентов может *только* ассистент, идентифицированный как LA_ASMITH.

Подобная процедура используется для разрешения проверки графика работ другим ассистентам лаборатории.

Физическая безопасность связана с контролем доступа в комнаты или здания, где размещается или обрабатывается информация. Представьте, что кто-то выключит главный компьютер во время выполнения транзакции обновления данных. Физическая безопасность также включает в себя защиту среды базы данных от пожара, землетрясений и других катастроф.

8.7.3. Процедуры резервного копирования и восстановления

Резервное копирование базы данных является важной процедурой, предназначенной для обеспечения дальнейшего использования базы данных после сбоя оборудования или программного обеспечения. Резервное копирование должно производиться на регулярной основе, а файлы резервного копирования должны храниться в строго определенном месте. Процедуры восстановления должны определять шаги, обеспечивающие полное восстановление системы после сбоя питания или природных катаклизмов.

Сценарий резервного копирования и восстановления для лаборатории UCL включает в себя следующие процедуры:

- ☐ каждый компьютер системы оборудован источником бесперебойного питания (ИБП) для обеспечения работы компьютеров при отключении электропитания;
- ☐ выполнение периодического резервного копирования: ежедневное для большинства активно используемых таблиц и еженедельное для менее задействованных таблиц. Резервные копии создаются во время невысокой загрузки системы и хранятся в различных местах для обеспечения физической безопасности;
- ☐ в системе управления базой данных используется журнал транзакций, позволяющий восстановить базу данных в том состоянии, которое она имела на момент возникновения аварийной ситуации.

8.8. Работа с базой данных

Работа с базой данных, также называемая *управлением базой данных (database management)* — это непрерывный процесс, включающий в себя административные и технические функции, предназначенные для обеспечения целостности БД. После того как база данных объявляется готовой к использованию, она должна пройти все эксплуатационные тесты и оценки. Результаты тестирования и оценки должны быть официально утверждены руководством компании.

8.8.1. Работоспособная база данных

Работоспособная база данных обеспечивает повседневное функционирование системы и обслуживает все необходимые процедуры. Если база данных разработана и спроектирована должным образом, ее существование не только расширяет возмож-

ности управления и оперативного получения необходимой информации, но и укрепляет информационную структуру предприятия в целом.

8.8.2. Рабочие процедуры

Рабочие процедуры БД состоят в составлении документов, описывающих все ежедневные операции с базой данных. В рабочих процедурах определены шаги, необходимые для выполнения специфических задач, таких как ввод данных, обслуживание данных, резервное копирование и восстановление данных и мероприятия по обеспечению безопасности.

8.8.3. Управление базой данных: обслуживание и развитие

После ввода базы данных в строй для повышения ее эффективности необходимо провести ряд мероприятий, связанных с управлением и контролем. Администратор БД отвечает за координацию всех оперативных и управленческих аспектов вновь созданной среды СУБД. В обязанности администратора БД входят:

- ☐ слежение за базой данных и ее настройка;
- ☐ планирование и размещение ресурсов при изменении и расширении БД;
- ☐ планирование и размещение ресурсов при периодическом обновлении ресурсов;
- ☐ обеспечение превентивного и корректирующего обслуживания (процедуры резервного копирования и восстановления);
- ☐ обеспечение управления конечными пользователями путем создания и определения пользователей, паролей, привилегий и т. д.;
- ☐ проведение периодических проверок безопасности;
- ☐ проведение необходимого обучения;
- ☐ разработка и исполнение стандартов БД;
- ☐ распространение БД среди пользователей организации;
- ☐ обеспечение завершения проекта БД в рамках временных ограничений и установленного бюджета.

Говоря кратко, администратор БД выполняет обязанности технического руководителя, что гарантирует надлежащее выполнение базой данных своих функций. Следовательно, действия администратора БД должны быть направлены на поддержку сообщества конечных пользователей, что обеспечивается созданием и должным исполнением политик БД, процедур, стандартов, безопасности и целостности, что в свою очередь является залогом надлежащего использования информации. Функции администратора БД детально будут обсуждаться в *гл. 16*.

Резюме

Проблемы, обсуждаемые в этой главе, абсолютно реальны. Вы должны понимать, что даже самая совершенная ER-модель может оказаться бесполезной на уровне реализации, если в ней не могут выполняться основные приложения и если она не

позволяет выполнить необходимые системные процедуры. Начальная диаграмма редко когда является идеальной основой для последующего логического и физического проектирования базы данных по нескольким причинам:

- ☐ некоторые связи не вполне определены, а некоторые и вовсе не определены;
- ☐ модель избыточна (вспомните схожесть сущностей `WITHDRAW` и `CHECK_OUT` в начальной `ER`-диаграмме);
- ☐ модели не хватает семантической точности, поэтому она не соответствует реальным операциям;
- ☐ модель не удовлетворяет требованиям конечных пользователей (например, с точки зрения производительности и безопасности).

Даже прекрасно спроектированная начальная `ER`-диаграмма вряд ли сможет поддерживать все транзакции БД. Поэтому концептуальный проект базы данных мы должны проверить по отношению ко всем транзакциям БД.

Перед тем как приступить к проверке, мы должны определить все атрибуты, первичные и внешние ключи для каждой сущности `ER`-диаграммы. После этого проектировщик должен создать образец данных. Проверка образца данных облегчает оценку адекватности сущностей и их атрибутов по отношению к требованиям конечных пользователей. Проектировщик использует нормализацию для обеспечения минимума избыточности данных, что является залогом хорошей производительности системы.

Процесс проверки `ER`-модели также требует тщательной проверки всех модулей системы. Поэтому необходимо выявить и проверить следующие компоненты:

- ☐ основная сущность системы;
- ☐ все модули и их компоненты;
- ☐ все процессы модулей.

Процесс проверки часто приводит к созданию новых сущностей и атрибутов или может оказаться, что одну или несколько сущностей необходимо удалить из проекта. Например, мы обнаружили, что сущность `WITHDRAW` нельзя реализовать в том виде, как она представлена на `ER`-диаграмме. Добавление новой сущности `INV_TRANS` и нового набора связей, основанных на этой сущности, позволили разрешить проблемы реализации.

После проверки концептуальной модели, основанной на `ER`-диаграмме, проектировщик приступает к логическому проектированию. На этапе логического проектирования определяются все необходимые табличные структуры, индексы и представления и определяется тип программного обеспечения.

По завершении логического проектирования дизайнер выполняет физическое проектирование. Физическое проектирование связано с определением требований к условиям хранения, с распределением памяти и методами доступа. Поскольку эти возможности зависят от операционной системы и физической конфигурации, физическое проектирование зависит как от программного обеспечения, так и от оборудования.

После физического проектирования начинается процесс реализации БД. В табличные структуры, созданные на этапе логического проектирования, с помощью соответствующих утилит загрузки и преобразования вводятся данные. Порядок ввода данных

Перед принятием проекта БД он должен быть протестирован и оценен. На самом деле тестирование и оценка выполняются постоянно, но они особенно важны перед вводом БД в строй. Хотя производительность базы данных (скорость и эффективность) имеет высокий приоритет, все же она является не единственным (и даже не самым важным) критерием оценки качества БД. Тестирование и оценка должны подтвердить, что база данных обеспечивает безопасность и целостность, что имеются процедуры резервного копирования и восстановления, и что доступ к БД и ее использование должным образом защищены.

Когда все соответствующие рабочие и управленческие проблемы выявлены и разрешены, база данных готова к вводу в строй. Существование БД не только гарантирует получение структурированной информации, которая является основой принятия решений, она к тому же позволяет считать данные корпоративным ресурсом и уплотняет организационную структуру всей организации.

Кластерные таблицы — cluster tables

Проверка — verification

1. Почему концептуальную модель необходимо проверять? Какие действия выполняются в ходе проверки?
2. Какие действия необходимо выполнить перед тем, как проект БД будет полностью реализован? (Убедитесь, что ваши действия выполняются в должной последовательности, и кратко опишите каждое из них.)
3. Какие основные факторы влияют на оценку производительности системы базы данных? Обсудите каждый из этих факторов.
4. Как бы вы проверяли ER-диаграмму, представленную на рис. 8.35? Дайте соответствующие рекомендации.
5. Опишите и обсудите соглашения ER-модели для следующей иерархии инвентарь/заказ лаборатории UCL:
 - категория;
 - тип;
 - класс;
 - подтип.
6. На современном предприятии постоянно проводится обучение сотрудников для поддержания уровня их квалификации в быстро изменяющемся мире конкуренции. Кроме того, правительство часто требует периодического повышения квалификации и обучения (например, пилоты должны проходить полугодовые курсы по полетам в условиях любой погоды, изучению полетных инструкций и т. д.). Чтобы предприятие могло отслеживать процесс обучения каждого из сотрудников, проработайте фрагмент ER-диаграммы, представленный на рис. 8.36, руководствуясь начальным бизнес-правилом:

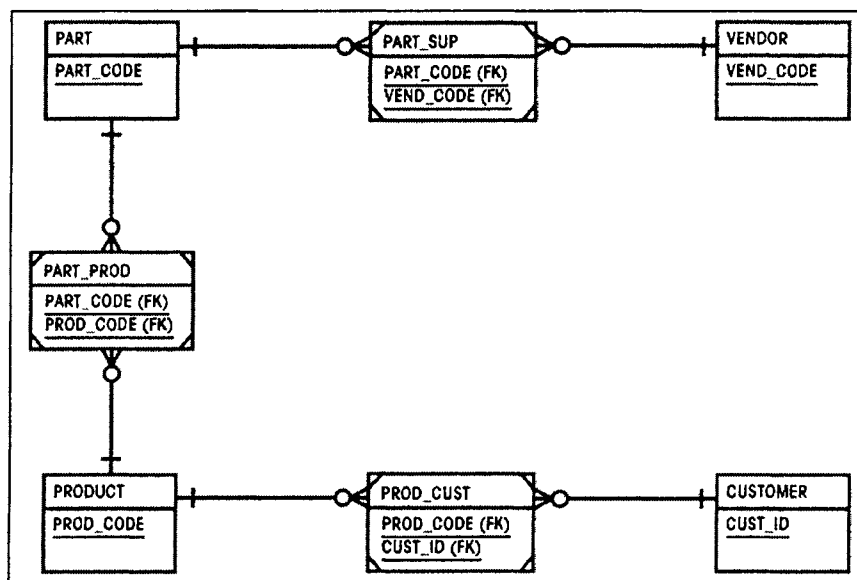


Рис. 8.35. ER-диаграмма к вопросу 4

Сотрудник может обучаться на нескольких курсах, и на каждом курсе могут обучаться несколько сотрудников.

После выполнения разработки данного сегмента ER-диаграммы проверьте его и создайте образец данных для каждой из трех таблиц для иллюстрации возможности реализации проекта.

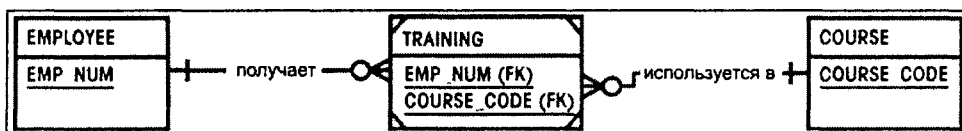


Рис. 8.36. Фрагмент ER-диаграммы для вопроса 6

7. Из этой главы вы узнали, что исследование требований к отчетности в модуле Inventory лаборатории UCL выявило следующие проблемы:

- в модуле Inventory создаются три отчета, один из которых представляет собой отчет о движении материальных средств. Но движение инвентаря охватывает две различные сущности (CHECK_OUT и WITHDRAW). Это затрудняет организацию вывода данных и ухудшает производительность системы;
- количество предметов, имеющих в наличии, обновляется при каждом движении материальных средств (покупка, выдача, выдача под отчет или ревизия склада). В системе представлены только операции выдачи и выдачи под отчет.

Какое решение можно предложить в данном случае? Какую пользу оно может принести для других типов инвентарного учета?

Задачи

1. Проверьте концептуальную модель, созданную в задаче 3 гл. 7. Создайте словарь данных для проверенной модели.
2. Проверьте концептуальную модель, созданную в задаче 4 гл. 7. Создайте словарь данных для проверенной модели.
3. Проверьте концептуальную модель, созданную в задаче 5 гл. 7. Создайте словарь данных для проверенной модели.
4. Проверьте концептуальную модель, созданную в задаче 6 гл. 7. Создайте словарь данных для проверенной модели.
5. Проверьте концептуальную модель, созданную в задаче 7 гл. 7. Создайте словарь данных для проверенной модели.
6. Спроектируйте (на логическом уровне) систему, которая позволит кураторам получать полный доступ к записи об успеваемости студента. Результаты вывода должны включать следующую информацию:

Name: XXXXXXXXXXXXXXXX X XXXXXXXXXXXXXXXX Page # of ##

Department: XXXXXXXXXXXXXXXX Major: XXXXXXXXXXXXXXXX

Social Security Number: ###-##-#### Report Date: ## XXXXXXXXXXXXXXXX ####

Fall, 1999

Course	Hours	Grade	Grade Points
CIS 200 (Intro to Microcomputers)	3	B	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
Total this semester:	##		GPA #.##
Total to date:	###		Cumulative GPA #.##

Spring, 2001

Course	Hours	Grade	Grade Points
CIS 300 (Computers in Society)	3	A	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
XXX #### (XXXXXXXXXXXXXXXXXX)	#	X	##
Total this semester:	##		GPA #.##
Total to date:	###		Cumulative GPA #.##

Summer, 2001

Course	Hours	Grade	Grade Points
CIS 400 (Systems Analysis)	4	A	##

XXX ####	(XXXXXXXXXXXXXXXXXX)	#	X	##
XXX ####	(XXXXXXXXXXXXXXXXXX)	#	X	##
XXX ####	(XXXXXXXXXXXXXXXXXX)	#	X	##
XXX ####	(XXXXXXXXXXXXXXXXXX)	#	X	##
Total this semester:		##		GPA #.##
Total to date:		###		Cumulative GPA #.##

7. Спроектируйте и проверьте приложение базы данных для одной из ваших местных некоммерческих организаций (Красный крест, Армия Спасения, церковь и т. д.). Создайте словарь данных для проверенного проекта.
8. Используя информацию, полученную на этапе физического проектирования (см. разд. 8.5), оцените необходимый объем памяти для хранения следующих сущностей:

RESERVATION

INV_TRANS

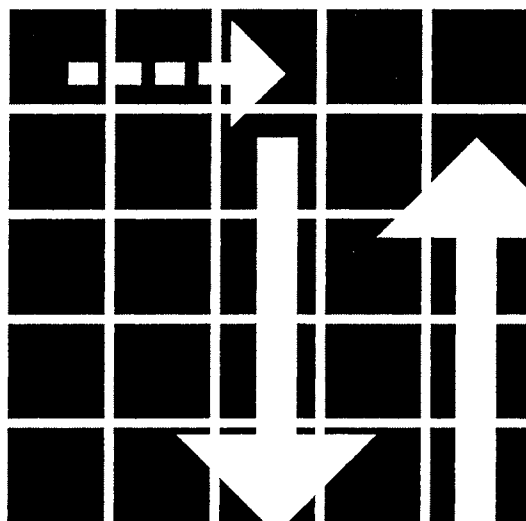
TR_ITEM

LOG

ITEM

INV_TYPE

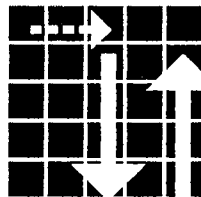
(Подсказка: можно обратиться к табл. 7.3 и посмотреть образец заполнения журнала LOG.)



ЧАСТЬ IV

РАСШИРЕННОЕ ИСПОЛЬЗОВАНИЕ КОНЦЕПЦИЙ БАЗ ДАННЫХ

Глава 9



Управление транзакциями и параллельным выполнением

В этой главе вы узнаете:

- ☐ что такое транзакция базы данных и каковы ее свойства;
- ☐ как управлять транзакциями в БД;
- ☐ что такое управление параллельным выполнением транзакций и какова его роль в обеспечении целостности базы данных;
- ☐ что собой представляют методы блокировки и как они работают;
- ☐ как используется управление восстановлением базы данных для обеспечения целостности.

Обзор

Транзакции базы данных отражают реальные операции, вызываемые такими событиями, как покупка товара, регистрация студента на курсе или внесение денег на банковский счет. Транзакции, как правило, состоят из нескольких частей. Например, при транзакциях продаж могут потребоваться обновление счета пользователя, корректировка товарного склада, обновление дебиторских счетов продавца и т. д. Для предотвращения проблем целостности данных все части транзакции должны завершиться успешно. Поэтому выполнение транзакций и управление ими является важным элементом работы с системой базы данных.

В этой главе будет показано, что основными свойствами транзакции БД являются атомарность, долговечность, сериализуемость и изолированность. После определения основных свойств транзакций мы продемонстрируем, как можно выполнять транзакции с помощью SQL. Мы также рассмотрим использование журнала транзакций, с помощью которого СУБД может восстанавливать прерванные транзакции.

Управление параллельным выполнением транзакций также является важной частью работы БД. Поскольку число одновременно выполняемых транзакций быстро растет при одновременном подключении к базе данных нескольких пользователей, управление параллельным выполнением транзакций особенно актуально в многопользовательских средах (только представьте себе число маршрутизируемых транзакций, проводимых через Интернет продавцами и поставщиками услуг!). Мы рассмотрим случаи невыполнения обновлений, потерю данных и составление противоречивых отчетов,

которые могут происходить во время параллельного выполнения транзакций. Мы покажем, что такие проблемы могут быть решены, если в СУБД используется планировщик управления параллельным выполнением транзакций (планировщик регулирует выполнение операций в двух или более конфликтующих транзакциях).

Мы также покажем, что большинство алгоритмов управления параллельным выполнением используют блокировки, метки времени или оптимистические методы. Поскольку блокировки являются основой для большинства методов управления параллельным выполнением, мы исследуем различные уровни и типы блокировок. Так как блокировки могут стать причиной возникновения тупиков (deadlock), мы обсудим три стратегии контроля тупиков: выявление, предотвращение и устранение. Затем мы обсудим результаты использования меток времени и оптимистических методов в управлении параллельным выполнением.

Содержимое базы данных может быть повреждено или разрушено полностью критическими ошибками при выполнении операций, включая ошибки управления транзакциями. Поэтому мы рассмотрим, как можно использовать восстановление базы данных для сохранения содержимого БД при помощи процедур резервного копирования. Сюда относятся как процедуры создания полной резервной копии БД, так и процедуры резервного копирования журнала транзакций.

9.1. Что такое транзакция?

Если вы читаете информацию из базы данных и/или записываете данные в БД (обновление), то вы порождаете транзакцию. Транзакция может состоять из простого оператора `SELECT` для вывода содержимого таблицы или из серий взаимосвязанных командных последовательностей обновления (`UPDATE`). Например, если вы продаете товар клиенту, то ваша транзакция будет состоять, по крайней мере, из двух частей: вам необходимо обновить склад, уменьшив количество единиц товара в таблице `PRODUCT` (товар), вычтя из него количество проданных единиц, а также обновить таблицу счетов для того, чтобы затем выставить счет клиенту.

Большинство транзакций действующих БД состоят из двух или более запросов. *Запрос к базе данных* эквивалентен одному SQL-оператору в прикладной программе или транзакции. Таким образом, только что описанная транзакция покупки формируется из двух запросов к БД (два последовательных оператора `UPDATE`), каждый из которых обновляет базу данных. Каждый запрос к базе данных порождает несколько операций ввода/вывода (input/output, I/O).

Транзакция это логическая единица работы с БД, которая должна быть или полностью выполнена, или отменена; при этом недопустимы никакие промежуточные состояния. Другими словами, многокомпонентная транзакция, такая, например, как описанная выше операция продажи, не может выполняться по частям. Только обновление склада или только обновление таблицы счетов недопустимо.

На рис. 9.1 представлена типичная транзакция инвентарного учета, при которой происходит обновление базы данных уменьшением имеющихся в наличии товаров (40 единиц) на 10 единиц (проданное количество), при этом на складе остается 30 единиц. Транзакция, которая изменяет содержимое базы данных, переводит БД из одного устойчивого состояния (consistent state) в другое. *Устойчивым состоянием*

базы данных считается состояние, при котором выполняются все условия ограничений целостности.

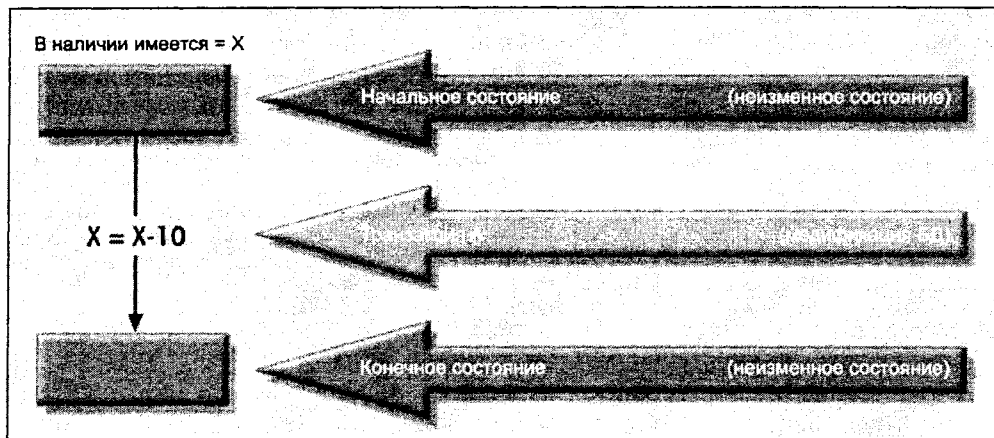


Рис. 9.1. Пример транзакции

Чтобы гарантировать устойчивость базы данных, каждая транзакция должна начинаться только в устойчивом состоянии БД. Если база данных не находится в устойчивом состоянии, то транзакция приведет к нарушению целостности БД и несоблюдению бизнес-правил. По причине этих ограничений, которые мы в дальнейшем будем обсуждать подробно, в целях обеспечения целостности все транзакции должны контролироваться и выполняться только в рамках СУБД.

Если в процессе транзакции получен доступ к БД, то после этого можно манипулировать ее содержимым. На примере рис. 9.1 элемент данных X должен быть считан из постоянного хранилища в оперативную память; его значение затем обновляется путем вычитания 10 единиц, и новое значение X записывается обратно из оперативной памяти в постоянное хранилище. *Во время этой транзакции СУБД должна гарантировать, что ни одна другая транзакция не получит доступа к объекту X .* После завершения транзакции все использованные в ней данные высвобождаются, чтобы к ним можно было получить доступ из других транзакций. База данных возвращается в устойчивое состояние.

9.1.1. Оценка результатов транзакции

Предположим, что нам необходимо проверить текущий баланс счета № 0908110638, размещенного в таблице CHECKACC. Такая транзакция может быть выполнена с помощью SQL-кода

```
SELECT ACC_NUM, ACC_BALANCE
FROM CHECKACC
WHERE ACC_NUM = '0908110638';
```

Несмотря на то что мы не сделали никаких изменений в таблице `ЧЕКАСС`, SQL-код представляет собой транзакции, поскольку мы *получали доступ* к базе данных. Если база данных находилась в устойчивом состоянии перед тем, как к ней был получен доступ, она останется в устойчивом состоянии и после этого, поскольку эта транзакция не изменяет содержимое БД.

Второй пример иллюстрирует более сложный набор транзакций. Предположим, что бухгалтеру необходимо оформить продажу в кредит 100 единиц товара X клиенту Y на сумму \$500. Необходимые транзакции могут быть выполнены в следующей последовательности:

- ☐ уменьшить количество товара X (quantity on hand, QOH) на 100;
- ☐ добавить сумму \$500 на дебиторский счет клиента Y.

С помощью SQL мы можем написать:

```
UPDATE PRODUCT
  SET PROD_QOH = PROD_QOH - 100
  WHERE PROD_CODE = 'X';

UPDATE ACCT_RECEIVABLE
  SET ACCT_BALANCE = ACCT_BALANCE + 500
  WHERE ACCT_NUM = 'Y';
```

Обе SQL-транзакции должны быть завершены полностью, чтобы отражать реальную операцию продажи. Если обе транзакции будут выполнены не полностью, то транзакция приведет к неустойчивому состоянию БД.

Теперь предположим, что СУБД завершила первую транзакцию, обновив количество товара X на складе. А во время выполнения второй части транзакции (обновление — UPDATE — таблицы дебиторских счетов) произошло отключение электропитания. Если компьютер не оборудован устройством бесперебойного питания, вторая часть транзакции не будет завершена. Поэтому таблица `PRODUCT` перейдет в обновленное состояние (соответствующим образом будет уменьшено количество товара на складе), но на дебиторский счет клиента сумма покупки не будет выставлена. База данных перейдет в неустойчивое состояние и не сможет корректно выполнять последующие транзакции. *СУБД должна обеспечивать возможность восстановления базы данных в предыдущее устойчивое состояние.*

Хотя СУБД разработана таким образом, что она позволяет восстановить предыдущее устойчивое состояние базы данных при прерывании выполнения набора транзакций, тем не менее, сами транзакции, определяются конечным пользователем или программистом и должны быть семантически корректны. СУБД не гарантирует, что семантический смысл транзакции действительно отражает реальную процедуру. Например, предположим, что при следующей продаже 10 единиц товара X команда обновления товара на складе (UPDATE) была записана следующим образом:

```
UPDATE PRODUCT
  SET PROD_QOH = PROD_QOH + 10
  WHERE PROD_CODE = 'X';
```

Примечание

Продажа должна привести к уменьшению значения атрибута PROD_QOH (наличие товара) на 10. Вместо этого эта команда UPDATE добавляет 10 единиц к значению атрибута PROD_QOH.

Хотя синтаксис этой команды UPDATE правильный, она приведет к неверному результату. Тем не менее, в любом случае СУБД выполнит эту транзакцию. Для СУБД не имеет значения, верно ли отражает данная транзакция реальные операции — это задача конечного пользователя. Конечные пользователи и программисты могут совершить в этом случае множество ошибок. Представьте себе последствия уменьшения количества товара S на складе вместо того, чтобы уменьшить количество товара X или зачисления кредитной суммы на дебиторский счет клиенту U вместо клиента Y (эти буквы находятся рядом на клавиатуре).

Очевидно, что неверные или неполные транзакции могут привести к полному нарушению целостности БД. Некоторые СУБД, особенно реляционные, предоставляют средства, с помощью которых пользователи могут задавать санкционированные ограничения на основе бизнес-правил. Другие правила целостности, такие как управление целостностью на уровне ссылок и сущностей, выполняются СУБД автоматически (если табличные структуры определены должным образом), что позволяет СУБД самостоятельно ратифицировать некоторые транзакции. Например, если в процессе транзакции в таблицу клиентов вводится новый клиент, а клиент с таким номером уже существует, то СУБД прервет транзакцию с кодом ошибки, указывающим на нарушение правила целостности первичного ключа.

9.1.2. Свойства транзакции

Все транзакции должны обладать свойствами атомарности, продолжительности, сериализуемости и изолированности. Мы кратко исследуем каждое из этих свойств.

- *Атомарность (atomicity)* требует, чтобы все операции (части) транзакции были завершены. В противном случае транзакция отменяется. Другими словами, транзакция представляет собой отдельную, неделимую логическую единицу работы.
- *Долговечность (durability)* указывает на непрерывность устойчивого состояния БД. После завершения транзакции база данных должна переходить в устойчивое состояние и это состояние не должно нарушаться даже при сбоях системы.
- *Сериализуемость (serializability)* представляет собой возможность одновременного выполнения нескольких транзакций. Точнее, под одновременными транзакциями понимается их сериализованное, последовательное выполнение (одна за другой). Это свойство имеет большое значение в многопользовательских и распределенных базах данных, где несколько транзакций с большой долей вероятности могут выполняться параллельно.
- *Изолированность (isolation)* означает, что данные, использующиеся в одной транзакции, не могут использоваться другой транзакцией до тех пор, пока первая не будет завершена. Другими словами, если выполняется транзакция T1 и использует объект данных X, то этот объект данных не может быть доступен никакой другой транзакции (T2, ..., Tn) до завершения транзакции T1. Это свойство имеет практическое значение в многопользовательских базах данных, поскольку здесь несколько пользователей одновременно могут получать доступ к базе данных.

По своей природе однопользовательская база данных автоматически гарантирует сериализованность и изолированность, поскольку в данный момент времени выполняется только одна транзакция. Атомарность и долговечность транзакций должны обеспечиваться однопользовательской СУБД (даже однопользовательские СУБД должны обеспечивать управление восстановлением при системных сбоях, перебоях в электропитании и некорректном использовании приложений).

В многопользовательских базах данных, функционирующих на универсальных машинах или в локальной сети, обычно одновременно выполняется множество транзакций. Поэтому многопользовательская СУБД должна предоставлять средства управления сериализацией и изолированностью транзакций — в дополнение к атомарности и долговечности — для того, чтобы сохранить устойчивость и целостность базы данных. Например, если на одном наборе данных параллельно выполняются несколько транзакций и вторая транзакция обновляет базу данных до завершения первой транзакции, то свойство изолированности будет нарушено и состояние базы данных не будет устойчивым. СУБД должна управлять транзакциями с помощью технологии управления выполнением параллельных операций для предотвращения таких нежелательных ситуаций.

9.1.3. Управление транзакциями с помощью SQL

Национальный Институт Стандартизации США (The American National Standards Institute, ANSI) устанавливает стандарты, определяющие управление транзакциями база данных с помощью SQL. Поддержка транзакций обеспечивается двумя SQL-операторами: COMMIT и ROLLBACK. Стандарты ANSI требуют, чтобы после инициализации транзакции пользователем или прикладной программой она выполнялась последовательностью SQL-операторов, пока не произойдет одно из следующих четырех событий:

1. Достигнут оператор COMMIT. В этом случае все изменения записываются в БД. Оператор COMMIT автоматически завершает SQL-транзакцию.
2. Достигнут оператор ROLLBACK. В этом случае все изменения отменяются, и база данных возвращается в предыдущее устойчивое состояние.
3. Достигнут конец программы. В этом случае все изменения записываются в базу данных для постоянного хранения. Это событие эквивалентно выполнению оператора COMMIT.
4. Выполнение программы неожиданно прервано. В этом случае все изменения отменяются, и база данных возвращается в предыдущее устойчивое состояние. Это событие эквивалентно выполнению оператора ROLLBACK.

Применение оператора COMMIT иллюстрируется следующими операциями продажи, при которых обновляется количество товара на складе (таблица PROD_QOH) и на дебиторский счет клиента записывается \$3500 при покупке клиентом 100 единиц товара 345TYX:

```
UPDATE PRODUCT
```

```
SET PROD_QOH = PROD_QOH - 100
```

```
WHERE PROD_CODE = '345TYX';
```

```
UPDATE ACCT_RECEIVABLE  
  SET ACCT_BALANCE = ACCT_BALANCE + 3500  
  WHERE ACCT_NUM = '60120010';  
COMMIT;
```

На самом деле оператор COMMIT, использованный в данном примере, необязателен, если оператор UPDATE является последним действием приложения и приложение завершается нормальным образом.

Транзакция начинает выполняться явно, как только встретится первый SQL-оператор. Не все реализации SQL соответствуют стандартам ANSI. В некоторых версиях используются операторы управления транзакциями, например, оператор

```
BEGIN TRANSACTION;
```

указывает начало новой транзакции. В других реализациях SQL, например, VAX/SQL, разрешается присваивать свойства транзакциям в качестве параметров оператора BEGIN.

9.1.4. Журнал транзакций

Для отслеживания всех транзакций, обновляющих БД, в СУБД используется журнал транзакций. Информация, содержащаяся в этом журнале, используется СУБД для восстановления БД, выполняемого оператором ROLLBACK при прерывании выполнения программы или системном сбое (проблемы сети или поломка жесткого диска). Некоторые РСУБД используют журнал транзакций для восстановления базы данных *вперед* (forward) по отношению к текущему устойчивому состоянию. После сбоя на сервере РСУБД Oracle, например, автоматически отменяет незавершенные транзакции и выполняет транзакции, которые выполнены, но еще не записаны в физическую память.

Когда СУБД выполняет транзакции, вызывающие изменения в БД, происходит автоматическое обновление журнала транзакций. В журнале хранится следующая информация:

- ☐ запись о начале транзакции;
- ☐ для каждого компонента транзакции (SQL-оператора) хранятся:
 - тип выполняемой операции (обновление, удаление, вставка);
 - имена объектов, на которые влияет транзакция (названия таблиц);
 - значения "до" и "после" для обновляемых полей;
 - указатели на предыдущую и последующие записи для такой же транзакции;
- ☐ завершение транзакции (COMMIT).

Хотя использование журнала транзакций увеличивает непроизводительные издержки при работе СУБД, однако возможность восстановления поврежденной БД того стоит.

В табл. 9.1 представлен упрощенный вид журнала транзакций, где отражены представленные выше два SQL-оператора UPDATE. В случае системного сбоя СУБД обращается к журналу транзакций для поиска всех незавершенных или не полностью завершенных транзакций и на основе этой информации осуществляет откат

(ROLLBACK) базы данных к предыдущему состоянию. После завершения процесса восстановления СУБД записывает в журнал все завершённые транзакции, которые не были записаны в базу данных перед возникновением сбоя.

Таблица 9.1. Журнал транзакций

TRL ID	TRX NUM	Пред. PTR	След. PTR	Операция	Таблица	Идент. столбца	Атрибут	"до"	"после"
341	101	Null	352	START	*** Start Transaction				
352	101	341	363	UPDATE	PRODUCT	345TYX	PROD_QOH	243	143
363	101	352	365	UPDATE	ACCT_RECEIVABLE	60120010	ACCT_BALANCE	1200	4700
365	101	363	Null	COMMIT	*** End transaction				

Примечание:

TRL ID = идентификатор записи в журнале транзакций

TRX NUM = номер транзакции

PTR = указатель идентификатора записи в журнале транзакций

(Номер транзакции TRX NUM автоматически присваивается СУБД).

Если команда ROLLBACK подана перед прерыванием транзакции, СУБД выполнит откат базы только для отдельной транзакции, а не для всех транзакций, с тем, чтобы обеспечить долговечность предыдущих транзакций. Другими словами, завершённые транзакции не отменяются.

Журнал транзакций сам по себе представляет собой базу данных, управляемую с помощью СУБД, как и любая другая БД. Для журнала транзакций точно так же представляет опасность неисправность жесткого диска и его переполнение. Поскольку в журнале транзакций содержится весьма важная информация, некоторые реализации СУБД поддерживают размещение журналов на нескольких дисках или на магнитных лентах для уменьшения риска потери информации при сбое системы.

9.2. Управление параллельным выполнением транзакций

Управление параллельным выполнением транзакций это координирование одновременного исполнения транзакций в распределенных базах данных. Цель управления параллельным выполнением состоит в обеспечении сериализации транзакций в многопользовательских базах данных. Контроль параллельного выполнения очень важен, поскольку одновременное выполнение транзакций в распределенных базах данных может стать причиной проблем с целостностью и непротиворечивостью данных. Тремя основными проблемами можно считать потерю изменений (lost update), несвязность данных (uncommitted data) и неоднозначность поиска (inconsistent retrievals).

9.2.1. Потеря изменений

Для иллюстрации потери изменений исследуем простую таблицу PRODUCT (товар). Одним из атрибутов этой таблицы является количество товара на складе (PROD_QOH). Предположим, что у нас есть товар, количество которого в настоящий момент (PROD_QOH) равно 35. Пусть имеют место две параллельные транзакции T1 и T2, которые вызывают обновление значения PROD_QOH для некоторых объектов таблицы PRODUCT. Эти транзакции выглядят так:

Транзакция	Вычисление
T1: Покупка 100 единиц	→ PROD_QOH = PROD_QOH + 100
T2: Продажа 30 единиц	→ PROD_QOH = PROD_QOH – 30

В табл. 9.2 представлено последовательное выполнение этих транзакций в нормальных условиях, которое дает правильный результат PROD_QOH = 105.

Таблица 9.2. Нормальное выполнение двух транзакций

Время	Транзакция	Шаг	Хранимое значение
1	T1	Чтение PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Запись PROD_QOH	135
4	T2	Чтение PROD_QOH	135
5	T2	PROD_QOH = 135 – 30	
6	T2	Запись PROD_QOH	105

Но представим себе, что транзакции могут считывать значение PROD_QOH из таблицы до завершения предыдущей транзакции (использующей тот же товар). Последовательность операций, представленная в табл. 9.3, показывает, как возникает проблема потери изменений. Обратите внимание, что первая транзакция (T1) еще не завершилась, когда уже выполняется вторая транзакция (T2). Поэтому T2 все еще оперирует со значением 35 и уменьшение этого значения на 30 дает результат 5, хранящийся в оперативной памяти. Тем временем T1 записывает значение 135 на диск, которое тут же переписывается значением 5 транзакции T2. Короче говоря, добавление 100 единиц "теряется" при такой последовательности операций.

Таблица 9.3. Потеря изменений

Время	Транзакция	Шаг	Хранимое значение
1	T1	Чтение PROD_QOH	35
2	T2	Чтение PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	

Таблица 9.3 (окончание)

Время	Транзакция	Шаг	Хранимое значение
4	T2	PROD_QOH = 35 – 30	
5	T1	Запись PROD_QOH (Потеря изменений)	135
6	T2	Запись PROD_QOH	5

9.2.2. Несвязность данных

Феномен несвязности данных возникает в том случае, когда две транзакции — T1 и T2 — выполняются параллельно и первая транзакция (T2) отменяется после того, как вторая транзакция (T2) уже получила доступ к несвязанным данным, таким образом нарушая свойство изолированности транзакции. Для иллюстрации этой возможности мы будем использовать те же транзакции, что и при обсуждении потери изменений. Однако на этот раз транзакция T1 отменяется, чтобы устранить добавление 100 единиц товара. Поскольку T2 вычитает 30 единиц из исходных 35, правильный ответ должен быть 5.

Транзакция

Вычисление

T1: Покупка 100 единиц → $\text{PROD_QOH} = \text{PROD_QOH} + 100$ (Отменена)

T2: Продажа 30 единиц → $\text{PROD_QOH} = \text{PROD_QOH} - 30$

В табл. 9.4 показано, как при нормальных условиях выполнение этих транзакций приводит к правильному результату.

Таблица 9.4. Корректное выполнение двух транзакций

Время	Транзакция	Шаг	Хранимое значение
1	T1	Чтение PROD_QOH	35
2	T1	$\text{PROD_QOH} = 35 + 100$	
3	T1	Запись PROD_QOH	135
4	T1	***** ROLLBACK *****	35
5	T2	Чтение PROD_QOH	35
6	T2	$\text{PROD_QOH} = 35 - 30$	
7	T2	Запись PROD_QOH	5

В табл. 9.5 показано как возникает проблема несвязности данных, если откат (ROLLBACK) завершается после того, как начнется выполнение транзакции T2.

Таблица 9.5. Проблема несвязности данных

Время	Транзакция	Шаг	Хранимое значение
1	T1	Чтение PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Запись PROD_QOH	135
4	T2	Чтение PROD_QOH (чтение несвязанных данных)	135
5	T2	PROD_QOH = 135 – 30	
6	T1	***** ROLLBACK *****	35
7	T2	Запись PROD_QOH	105

9.2.3. Неоднозначность поиска

Неоднозначность поиска возникает, когда в транзакции выполняется агрегатная функция над множеством данных, в то время как другая транзакция обновляет эти данные. Проблема состоит в том, что транзакция может считывать некоторые данные перед их изменением, а другие данные *после* их изменения, что приводит к несовместимости данных.

Для иллюстрации такой проблемы представим следующий сценарий.

- 1. В транзакции T1 рассчитывается общее количество товара в таблице PRODUCT.
- 2. Одновременно транзакция T2 обновляет количество (PROD_QOH) для двух товаров в таблице PRODUCT.

Эти две транзакции представлены в табл. 9.6.

Таблица 9.6. Поиск данных во время обновления

Транзакция T1	Транзакция T2
SELECT SUM (PROD_QOH)	UPDATE PRODUCT
FROM PRODUCT	SET PROD_QOH = PROD_QOH + 30
	WHERE PROD_CODE = '125TYZ'
	UPDATE PRODUCT
	SET PROD_QOH = PROD_QOH - 30
	WHERE PROD_CODE = '345TYZ'
	COMMIT;

Пока T1 рассчитывает общее количество товара (PROD_QOH) по всем позициям, транзакция T2 производит коррекцию ошибок ввода: пользователь добавил 30 единиц товара

345TYX (PROD_QOH) вместо добавления 30 единиц товара 125TYZ (PROD_QOH). Начальное и конечное значения PROD_QOH представлены в табл. 9.7

Таблица 9.7. Результаты транзакций: исправление введенных данных

	Перед	После
PROD_CODE	PROD_QOH	PROD_QOH
104XCV	100	100
110YGH	120	120
125TYZ	70	$(70 + 30) \rightarrow 100$
345TYX	35	$(35 - 30) \rightarrow 5$
350TYX	100	100
355TYX	30	30
ВСЕГО	455	455

Хотя окончательные результаты после исправлений правильные, в табл. 9.8 продемонстрировано, что во время выполнения транзакций возможен неоднозначный поиск, что делает результаты транзакции T1 неверными.

Таблица 9.8. Неоднозначный поиск

Время	Транзакция	Действие	Значение	Всего
1	T1	Чтение PROD_QOH для PROD_CODE = '104XCV'	100	100
2	T1	Чтение PROD_QOH для PROD_CODE = '110YDC'	120	220
3	T2	Чтение PROD_QOH для PROD_CODE = '125TYZ'	70	
4	T2	PROD_QOH = $70 + 30$		
5	T2	Запись PROD_QOH для PROD_CODE = '125TYZ'	100	
6	T1	Чтение PROD_QOH для PROD_CODE = '125TYZ'	100	320 (после)
7	T1	Чтение PROD_QOH для PROD_CODE = '345TYX'	35	355 (до)
8	T2	Чтение PROD_QOH для PROD_CODE = '345TYX'	35	
9	T2	PROD_QOH = $35 - 30$		
10	T2	Запись PROD_QOH для PROD_CODE = '345TYX'	5	

Таблица 9.8 (окончание)

Время	Транзакция	Действие	Значение	Всего
11	T2	***** COMMIT *****		
12	T1	Чтение PROD_QOH для PROD_CODE = '350TYX'	100	455
13	T1	Чтение PROD_QOH для PROD_CODE = '355TYX'	30	485

Рассчитанный ответ 455 очевидно ошибочен, поскольку мы знаем правильный ответ 455. Если СУБД не будет управлять параллельным выполнением транзакций, то многопользовательская база данных может полностью разрушить информационную систему.

9.2.4. Планировщик

Теперь вы понимаете, какие могут возникнуть проблемы при одновременном выполнении двух или более транзакций. Вы узнали, что в транзакцию базы данных входит серия операций ввода/вывода, которые переводят БД из одного устойчивого состояния в другое. Наконец, теперь вы знаете, что непротиворечивость базы данных может обеспечиваться только перед выполнением или после выполнения транзакции. В процессе выполнения транзакций база данных неизбежно проходит через временные неустойчивые состояния. Такие временные состояния существуют из-за того, что компьютер не может выполнять две операции одновременно и должен, следовательно, выполнять их последовательно. Во время этого последовательного процесса свойство изолированности транзакций предохраняет их от доступа к данным, которые все еще используются другими транзакциями.

В предыдущих примерах мы выполняли операции в рамках транзакции в произвольном порядке. Пока две транзакции T1 и T2 получают доступ к *несвязанным* данным, конфликтов не возникает и порядок выполнения транзакций не влияет на конечный результат. Но если транзакции оперируют связанными (или одними и теми же) данными, возможен конфликт между компонентами транзакции, и выбор порядка операций может привести к нежелательным последствиям. Как же определить правильный порядок, и кто должен установить этот порядок? К счастью, СУБД предоставляет такую возможность с помощью встроенного планировщика.

Планировщик (scheduler) устанавливает порядок, в котором выполняются операции в параллельных транзакциях. Планировщик чередует выполнение операций базы данных для обеспечения последовательности. Чтобы определить должный порядок, планировщик действует на основе алгоритмов управления параллельным выполнением, таких как блокировка или метки времени, которые мы рассмотрим в следующих разделах.

Планировщик также обеспечивает эффективную работу центрального процессора компьютера (CPU). Если невозможно спланировать выполнение транзакций, то все транзакции будут выполняться по принципу "первый пришел — первым обслужен". Проблема такого подхода в том, что время центрального процессора тратится попус-

ту на время ожидания окончания операций READ (чтение) или WRITE (запись). Короче говоря, порядок обработки по принципу "первый пришел — первым обслужен" приводит к увеличению времени отклика в многопользовательских СУБД. Поэтому необходим иной метод, который помог бы увеличить общую производительность системы, задействовав время простоя процессора.

Операции с базой данных, в которых используются чтение и/или запись, могут привести к конфликтам. Например, в табл. 9.9 представлены возможные конфликты при одновременном выполнении двух транзакций T1 и T2 над одними и теми же данными. На основе табл. 9.9 можно сделать вывод, что две операции конфликтуют при получении доступа к одним и тем же данным, по крайней мере, во время одной из операций записи (WRITE).

Таблица 9.9. Конфликт чтения/записи: матрица конфликтов базы данных

Операции	ТРАНЗАКЦИИ	
	T1	T2
	<i>Read</i>	<i>Read</i>
	Read	Write
	Write	Read
	Write	Write

Примечание: в таблице жирным курсивом выделены неконфликтующие операции.

Есть несколько методов планирования выполнения конфликтующих операций в параллельных транзакциях. Сюда относятся блокировка, метки времени и оптимистические методы. Поскольку методы блокировки используются чаще всего, рассмотрим их в первую очередь.

9.3. Управление параллельным выполнением транзакций методом блокировки

Блокировка (lock) гарантирует эксклюзивное использование данных только в данной транзакции. Другими словами, транзакция T2 не получит доступа к данным, которые в настоящий момент используются в транзакции T1. Транзакция предварительно блокирует доступ к данным; блокировка снимается после завершения транзакции, вот почему следующая транзакция может блокировать данные для выполнения своих действий.

Вы должны помнить из предыдущих рассуждений, что в процессе транзакции не обеспечивается непротиворечивость данных: база данных может временно находиться в неустойчивом состоянии при выполнении обновления. Следовательно, блокировка необходима для того, чтобы другие транзакции не считывали противоречивую информацию.

Большинство многопользовательских СУБД автоматически инициирует и выполняет процедуры блокировки. Вся информация о блокировке обрабатывается *менеджером блокировок (lock manager)*, отвечающим за установку и контроль блокировок в транзакциях.

9.3.1. Степень детализации блокировок

Степень детализации блокировок (lock granularity) показывает уровень использования блокировки. Блокирование может осуществляться на следующих уровнях: база данных, таблица, страница, строка или даже поле (атрибут).

Уровень базы данных

При *блокировке на уровне базы данных* блокируется вся база данных, при этом предотвращается использование любых таблиц БД транзакцией T2, пока выполняется транзакция T1. Такой уровень блокировки годится для пакетных процедур, но плохо пригоден для работы многопользовательских СУБД. Вы можете себе представить, как м-е-д-л-е-н-н-о будет осуществляться доступ к данным, когда тысячи транзакций будут ожидать завершения транзакции, каждая из которых по очереди блокирует всю базу данных. На рис. 9.2 представлена схема блокировки на уровне базы данных. Обратите внимание, что транзакции T1 и T2 не могут получать доступ к БД одновременно, даже если они используют разные таблицы.

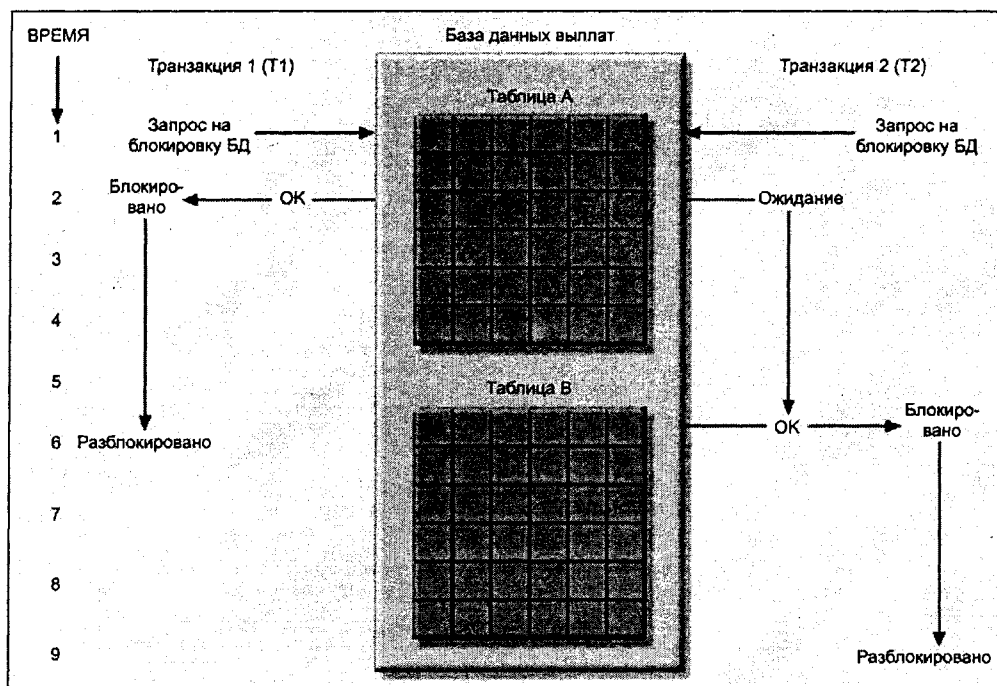


Рис. 9.2. Схема блокировки на уровне базы данных

Уровень таблицы

На этом уровне блокируется отдельная таблица БД, предотвращая доступ транзакции Т2 к любой строке, пока транзакция Т1 использует данную таблицу. Если транзакции требуется доступ к нескольким таблицам, то может блокироваться каждая из этих таблиц. Однако две транзакции могут получать доступ к одной и той же базе данных, если они работают с разными таблицами.

Блокировка на уровне таблиц менее ограничена, чем блокировка на уровне базы данных, при которой возникают проблемы с производительностью, когда множество транзакций ждут доступа к одной таблице. Однако блокировка таблицы вызывает особенное раздражение, когда различным транзакциям требуется доступ к разным частям одной и той же таблицы, т. е. когда транзакции не пересекаются друг с другом. Следовательно, блокировка на уровне таблицы также мало пригодна для многопользовательских СУБД. На рис. 9.3 представлена схема блокировки на уровне таблицы. Здесь нужно обратить внимание на то, что транзакции Т1 и Т2 не могут получить доступ к одной и той же таблице, даже если они используют разные строки; транзакция Т2 должна ждать, пока транзакция Т1 разблокирует таблицу.

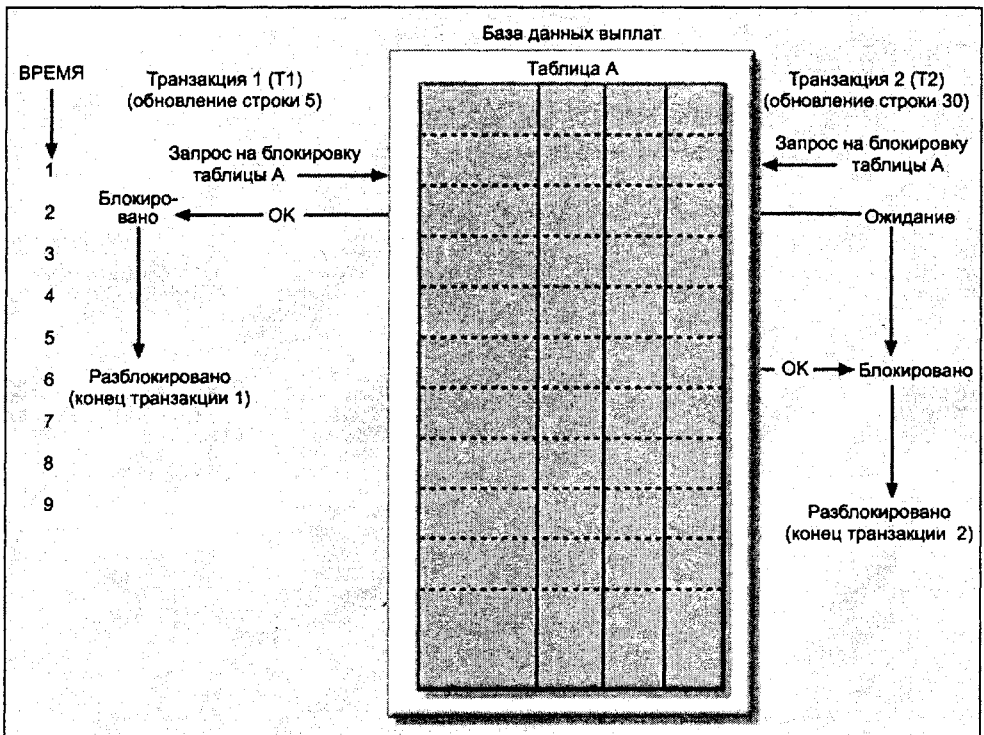


Рис. 9.3. Схема блокировки на уровне таблицы

Уровень страницы

При блокировке на уровне страницы СУБД будет блокировать дисковую страницу. *Дисковая страница*, или *страница* эквивалентна *дисковому блоку*, который представляет собой адресуемый раздел диска. Страница имеет фиксированный размер, например, 4 Кбайт, 8 Кбайт 16 Кбайт и т. д. Таблица может охватывать несколько страниц, а таблица может содержать строки одной или нескольких таблиц. Блокировка на уровне страницы в настоящее время — наиболее часто применяемый метод блокировки в многопользовательских СУБД. Схема блокировки на уровне страницы представлена на рис. 9.4. Обратите внимание, что транзакции T1 и T2 получают доступ к одной и той же таблице при блокировке разных дисковых страниц. Если транзакции T2 потребуется использование строки, расположенной на странице, заблокированной транзакцией T1, то T2 придется дожидаться, пока T1 ее разблокирует.

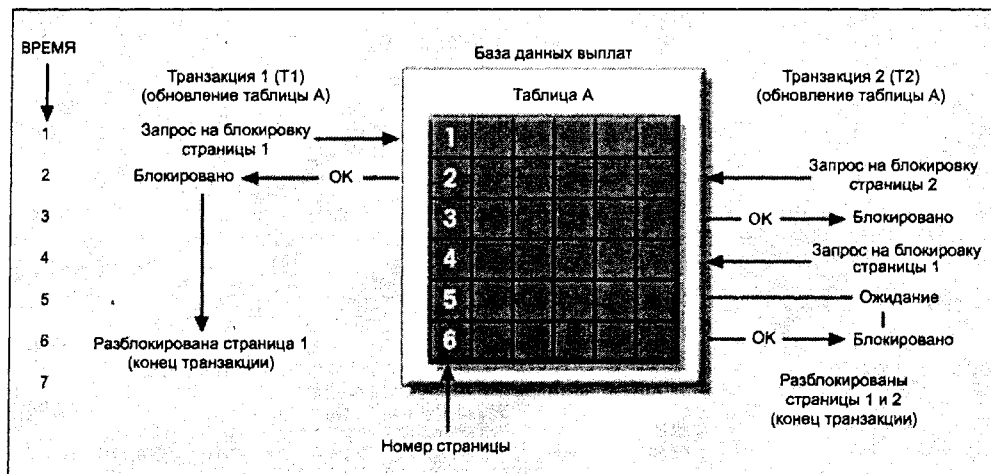


Рис. 9.4. Схема блокировки на уровне страницы

Уровень строки

Блокировка на уровне строки имеет меньше ограничений, чем все приведенные выше. СУБД позволяет параллельным транзакциям получать доступ к различным строкам одной и той же таблицы, даже если строки размещены на одной дисковой странице. Хотя блокировка на уровне строк улучшает возможности доступа к данным, управление такой блокировкой приводит к большим накладным расходам (блокировка требуется для каждой строки в каждой таблице БД!). На рис. 9.5 представлена схема блокировки на уровне строки.

Обратите внимание, что обе транзакции могут выполняться одновременно, даже если необходимые им строки находятся на одной странице. Транзакция T2 будет ожидать завершения транзакции T1 только в том случае, если они обращаются к одной и той же строке.

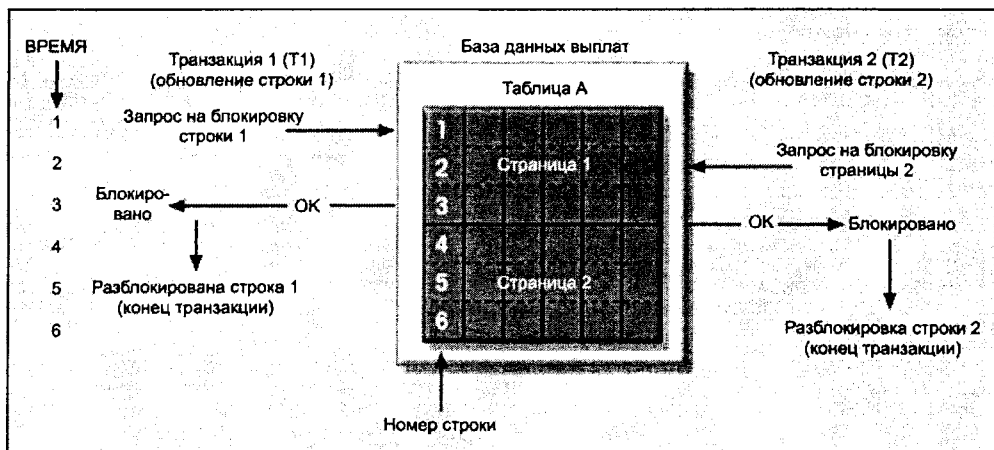


Рис. 9.5. Схема блокировки на уровне строки

Уровень поля

Блокировка на уровне поля позволяет параллельным транзакциям получать доступ к одной и той же строке, если они используют в этой строке разные поля (атрибуты). Очевидно, что такой метод блокировки предоставляет наиболее гибкий доступ к данным в многопользовательских средах; он применяется редко, поскольку требует неоправданно больших накладных расходов.

9.3.2. Типы блокировок

Независимо от уровня блокировки в СУБД могут применяться различные *типы блокировок*: двоичная блокировка (binary lock) или разделяемая/исключающая блокировка (shared/exclusive).

Двоичная блокировка

Двоичная блокировка имеет только два состояния: заблокировано (1) или разблокировано (0). Если некоторый объект (т. е. база данных, таблица, страница или строка) блокируется транзакцией, то ни одна другая транзакция не сможет получить доступ к этому объекту. Когда объект разблокирован, любая транзакция может заблокировать его для своих нужд. Как правило, транзакция должна разблокировать объект после своего завершения. Каждая операция БД требует блокировки объекта, на который она воздействует. Поэтому для каждой транзакции требуются операции разблокировки и блокировки для каждого объекта, к которому осуществляется доступ. Такие операции автоматически контролируются и планируются СУБД; пользователю нет необходимости беспокоиться о блокировке и разблокировке данных. (Каждая СУБД использует некоторый механизм блокировки по умолчанию. Если конечный пользователь захочет переопределить этот механизм, то для этого в SQL имеются соответствующие команды, например, LOCK TABLE и др.)

Технология двоичной блокировки представлена в табл. 9.10, где с ее помощью решается представленная ранее, в табл. 9.3, проблема потери изменения. Однако двоичная блокировка в настоящее время считается слишком строгой. Например, если две транзакции пытаются выполнить чтение одного и того же объекта БД, то СУБД не позволит это сделать даже в том случае, если ни одна из транзакций не выполняет обновление объекта (и, следовательно, при этом не возникают никакие проблемы, связанные с параллельным выполнением). Вспомните, что проблема параллельного выполнения в табл. 9.9 возникает только тогда, когда две параллельные транзакции обновляют базу данных.

Таблица 9.10. Пример двоичной блокировки

Время	Транзакция	Шаг	Хранимое значение
1	T1	Блокировка таблицы PRODUCT	
2	T1	Чтение PROD_QOH	35
3	T1	$PROD_QOH = 35 + 100$	
4	T1	Запись PROD_QOH	135
5	T1	Разблокировка таблицы PRODUCT	
6	T2	Блокировка таблицы PRODUCT	
7	T2	Чтение PROD_QOH	135
8	T2	$PROD_QOH = 135 - 30$	
9	T2	Запись PROD_QOH	105
10	T2	Разблокировка таблицы PRODUCT	

Разделяемая/исключающая блокировка

Термины "разделяемая" (shared) и "исключающая" (exclusive) указывают на природу блокировки. Исключающая блокировка имеет место при специальном резервировании доступа для транзакции, которая блокирует объект. Исключающая блокировка должна использоваться, когда есть вероятность конфликта (см. в табл. 9.9 варианты "чтение" и "запись"). Разделяемая блокировка имеет место, когда параллельные транзакции получают доступ по чтению на фоне общей блокировки. Разделяемая блокировка не вызывает конфликтов, поскольку все параллельные транзакции могут выполнять только операции чтения.

Разделяемая блокировка устанавливается, когда транзакция пытается считывать данные из БД, и для данного объекта не установлена исключающая блокировка. Исключающая блокировка устанавливается, когда транзакция записывает (обновляет) данные, а для данного объекта не установлена никакая блокировка другими транзакциями. При использовании технологии разделения/исключения блокировка может находиться в трех состояниях: разблокировка, разделение (чтение) и исключение (запись).

Как видно из табл. 9.9, две транзакции конфликтуют, только когда хотя бы одна из них является транзакцией записи (Write). Поскольку две транзакции чтения (Read) могут без проблем выполняться одновременно, разделяемая блокировка позволяет нескольким транзакциям чтения одновременно считывать один и тот же объект данных. Например, если транзакция T1 получает разделяемую блокировку объекта X, а транзакция T2 пытается читать объект X, то T2 может также получить разделяемую блокировку объекта X.

Если транзакция T2 обновляет объект X, то она потребует исключяющей блокировки объекта X. *Исключающая блокировка предоставляется тогда и только тогда, когда на объекте данных не установлено никакой другой блокировки.* Следовательно, если разделяемая или исключяющая блокировка установлены транзакцией T1 на объекте X, то для транзакции T2 исключяющая блокировка не может быть предоставлена, и T2 должна ожидать завершения транзакции T1.

Хотя возможность разделяемой блокировки обеспечивает более эффективный доступ к данным, схема разделяемой/исключающей блокировки увеличивает накладные расходы по управлению блокировкой по следующим причинам:

- ☐ перед предоставлением блокировки должен быть известен тип уже установленной блокировки;
- ☐ имеются три операции блокировки: READ_LOCK (проверка типа блокировки), WRITE_LOCK (установка блокировки) и UNLOCK (отмена блокировки);
- ☐ эту схему необходимо было расширить, чтобы разрешить повышение статуса блокировки (с разделяемой на исключительную) и понижения ее статуса (с исключяющей на разделяемую).

Несмотря на то что блокировки предотвращают противоречивость данных, их использование может вызывать две большие проблемы:

- ☐ окончательный план транзакций не всегда удастся сериализовать;
- ☐ в плане выполнения транзакций могут существовать тупики. Тупик базы данных, напоминающий дорожную пробку в большом городе, возникает, когда каждая из двух транзакций ожидает снятия блокировки с данных другой транзакцией.

К счастью, этих проблем можно избежать: сериализованность обеспечивается посредством протокола установки блокировок, называемого еще двухфазной блокировкой, а с тупиками можно справиться с помощью технологии выявления и предотвращения блокировок. Мы рассмотрим эти технологии в последующих двух разделах этой главы.

9.3.3. Обеспечение сериализуемости при помощи двухфазной блокировки

Двухфазная блокировка определяет способ, которым транзакции получают и снимают блокировки. Двухфазная блокировка гарантирует сериализуемость, но предотвращает тупики. Двухфазная блокировка состоит из двух этапов (фаз):

1. *Фаза роста (growing phase)*, при которой транзакция получает все необходимые ей блокировки без разблокирования каких-либо данных. Когда все блокировки получены, транзакция находится в *точке блокирования*.

2. *Фаза сжатия (shrinking phase)*, при которой транзакция снимает все блокировки и не получает каких-либо новых блокировок.

Протокол двухфазной блокировки подчиняется следующим правилам:

- две транзакции не могут иметь конфликтующих блокировок;
- в рамках одной транзакции ни одна операция разблокировки не может предшествовать операции блокировки;
- нельзя выполнять никаких действий над данными до тех пор, пока не получены все блокировки — т. е. до тех пор, пока транзакция не будет находиться в точке блокирования.

На рис. 9.6 представлен протокол двухфазной блокировки.

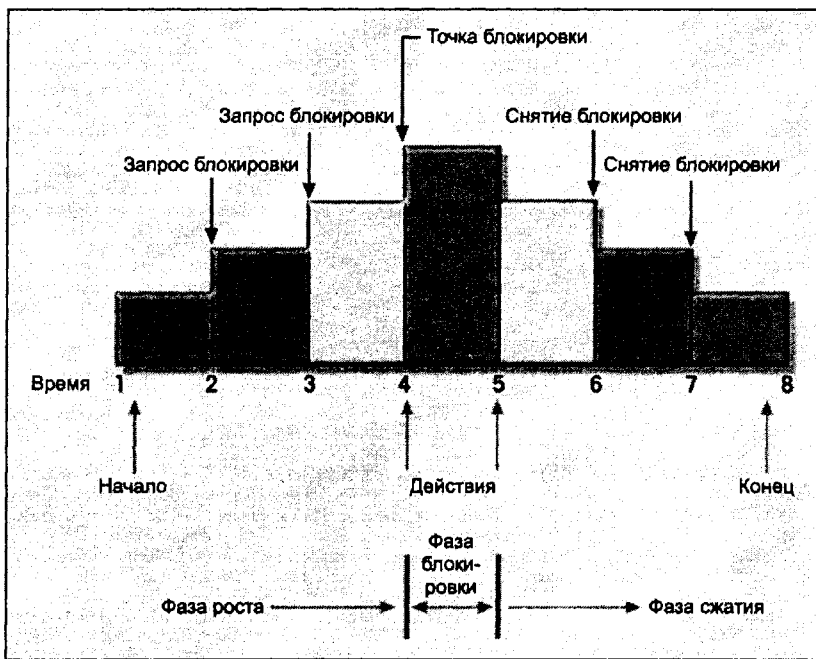


Рис. 9.6. Протокол двухфазной блокировки

В этом примере транзакции предоставляются все необходимые блокировки, до тех пор, пока она не достигнет точки блокирования (в этом примере транзакции требуется две блокировки). Когда достигнута точка блокирования, данные изменяются в соответствии с требованиями транзакции. В конце концов транзакция завершается после снятия всех предоставленных ей на первой фазе блокировок.

9.3.4. Тупики

Тупиком (*deadlock*) называется ситуация, при которой каждая из двух транзакций ожидает снятия блокировки данных другой транзакцией. Тупики происходят, когда две транзакции, T1 и T2, находятся в следующем состоянии:

T1 = доступ к объектам данных X и Y

T1 = доступ к объектам данных Y и X

Если транзакция T1 не разблокировала объект Y, то транзакция T2 не может начаться; если T2 не разблокировала объект X, то не может продолжить свою работу T1. Следовательно, T1 и T2 будут находиться в состоянии ожидания неопределенно долго, причем каждая транзакция будет ожидать разблокировки данных другой транзакцией. В табл. 9.11 продемонстрирован процесс возникновения тупиков.

Таблица 9.11. Пример двоичной блокировки

Время	Транзакция	Отклик	Статус блокировки	
0			Объект X	Объект Y
			Разблокирован	Разблокирован
1	T1:LOCK(X)	ОК	Заблокирован	Разбл.
2	T2:LOCK(Y)	ОК	Заблокирован	Разблокирован
3	T1:LOCK(Y)	Ожидание	Заблокирован	Заблокирован
4	T2:LOCK(X)	Ожидание	Заблокирован	Т
5	T1:LOCK(Y)	Ожидание	Заблокирован	У
6	T2:LOCK(X)	Ожидание	Заблокирован	П
7	T1:LOCK(Y)	Ожидание	Заблокирован	И
8	T2:LOCK(X)	Ожидание	Заблокирован	К
9	T1:(LOCK(Y)	Ожидание	Заблокирован	
...
...
...
...

В приведенном примере используются только две транзакции для демонстрации возникновения тупика. В реальных СУБД одновременно может выполняться гораздо больше транзакций, поэтому вероятность возникновения тупиков очень велика. Обратите внимание, что тупики возможны, только если одна из транзакций ожидает получения исключительной блокировки данных; при разделяемых блокировках тупики возникать не могут.

Для контроля тупиков есть три основных технологии.

- *Предотвращение тупиков.* Запрос транзакцией новой блокировки отменяется, если есть возможность возникновения тупика. Если транзакция прекращается, все изменения, сделанные этой транзакцией, отменяются и все блокировки, полученные этой транзакцией, снимаются. Транзакция затем планируется на выполнение заново. Технология предотвращения тупиков работает корректно, поскольку она исключает ситуации, приводящие к тупикам.
- *Выявление тупиков.* СУБД периодически проверяет базу данных на существование тупиков. Если тупики найдены, одна из транзакций ("жертва") прекращается (производится ее откат и повторный запуск), а другая транзакция продолжается.
- *Уклонение от тупиков.* Транзакция должна получать все необходимые блокировки, перед тем как начать свою работу. При такой технологии отпадает необходимость отката конфликтующих транзакций, т. к. блокировки предоставляются по порядку. Однако последовательное предоставление блокировок, необходимое для уклонения от тупика, увеличивает время отклика.

Выбор наилучшего метода контроля тупиков зависит от конфигурации базы данных. Например, если возможность возникновения тупиков невелика, то рекомендуется метод выявления тупиков. Однако если вероятность тупиков высока, рекомендуется метод предотвращения тупиков. Если время отклика системы не играет существенной роли, можно применять метод уклонения от тупиков.

9.4. Управление параллельным выполнением транзакций при помощи меток времени

При таком подходе каждой транзакции присваивается глобальная уникальная *метка (штамп) времени (time stamp)*. Значение метки времени явно определяет порядок, в котором транзакции предоставляются СУБД. Метки времени должны обладать двумя свойствами: уникальностью и монотонностью. *Уникальность (uniqueness)* гарантирует отсутствие одинаковых значений меток времени, а *монотонность¹ (monotonicity)* гарантирует, что значение метки времени всегда возрастает.

Все операции базы данных (чтение и запись) в рамках данной транзакции должны иметь одну и ту же метку времени. СУБД выполняет конфликтующие операции в порядке меток времени, что гарантирует их последовательное выполнение. Когда две транзакции конфликтуют, одна из них останавливается, откатывается, снова планируется на запуск и ей присваивается новая метка времени.

Недостаток использования меток времени в том, что каждое значение метки времени, хранящееся в базе данных, требует двух дополнительных полей — одно для хранения метки времени последнего считывания поля, а другое — для последнего обновления. Метки времени, следовательно, требуют увеличения памяти и приводят к увеличению дополнительных расходов. При этом, как правило, требуется больше

¹ См. W. H. Kohler, "A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems.", *Computer Surveys* 3(2), июнь 1981, стр. 149—183.

системных ресурсов, поскольку существует вероятность, что несколько транзакций могут быть остановлены, затем снова включены в план на исполнение и им присвоены новые метки времени.

9.5. Управление параллельным выполнением транзакций при помощи оптимистических методов

Оптимистический подход основан на предположении, что большинство операций баз данных не конфликтуют. При этом подходе не требуются блокировки или метки времени. Вместо этого транзакция выполняется без ограничений до полного ее завершения. При оптимистическом подходе каждая транзакция проходит две или три фазы: чтение (read), подтверждение (validation) и запись² (write).

- На фазе чтения транзакция читает базу данных, выполняет необходимые расчеты и делает обновления в индивидуальной копии значений БД. Все операции обновления в транзакции записываются во временный файл обновлений, недоступный остальным транзакциям.
- На фазе подтверждения транзакция проверяется на предмет возможного нарушения целостности и непротиворечивости базы данных. Если подтверждается, что транзакция не приводит к нарушениям, то она переводится в фазу записи. Если же проверка выявляет возможность нарушений, то транзакция перезапускается, а все сделанные изменения отменяются.
- На фазе записи в базу данных вводятся все необходимые изменения.

Оптимистический подход применим для большинства баз данных, где выполняется очень много операций обновления.

В интенсивно используемых базах данных управление тупиками (предотвращение и выявление) представляет собой важнейшую функцию СУБД. При этом используются одна или несколько технологий, которые мы только что обсудили, а также их различные вариации. Однако средства борьбы с тупиками иногда хуже самих тупиков. Поэтому иногда приходится восстанавливать БД в непротиворечивое состояние с помощью специальных методов.

9.6. Управление восстановлением базы данных

Восстановление базы данных представляет собой процесс перевода БД из текущего состояния (как правило, неустойчивого) в предыдущее устойчивое состояние. Методы восстановления основаны на *свойстве атомарности транзакций (atomic transaction property)*. Все фрагменты транзакции должны рассматриваться как простые логические единицы работы, в рамках которых должны быть завершены все операции, приводя-

² См., в частности, Н. Т. Kung и J. Т. Robinson, "Optimistic Methods for Concurrency Control", *ACM Transaction on Database Systems* 6(2), июнь 1981, стр. 213—226.

щие БД в устойчивое состояние. Если по каким-либо причинам какая-то операция транзакции не может быть завершена, то транзакция и все изменения, сделанные ею в БД, должны быть отменены (восстановлены в предыдущее состояние).

Хотя мы сейчас говорили о восстановлении транзакций, при возникновении некоторых видов критических ошибок методы восстановления применимы и к базе данных, и к системе. Функции резервного копирования и восстановления представляют собой важнейший компонент современных СУБД. Некоторые СУБД предоставляют функции, позволяющие администратору БД планировать автоматическое резервное копирование БД на энергонезависимые вспомогательные устройства хранения, такие как диски или магнитные ленты.

Есть разные уровни резервного копирования.

- ☐ *Полное резервное копирование (full backup)* — создается полная копия (дамп) базы данных.
- ☐ *Выборочное копирование (differential backup)* базы данных включает в себя только последние изменения в БД (по сравнению с последним резервным копированием).
- ☐ *Резервное копирование журнала транзакций (transaction log backup)* — копируются только операции журнала транзакций, не представленные в предыдущей резервной копии БД.

Резервная копия БД хранится в безопасном месте, обычно в другом здании, и должна быть защищена от пожара, хищения, наводнения и других возможных бедствий. Наличие резервной копии гарантирует восстановление базы данных после сбоев системы (оборудования или программного обеспечения).

Крах базы данных и системы, как правило, вызывается ошибками в программном обеспечении, сбоем оборудования, программными действиями, транзакциями или внешними факторами.

Программное обеспечение

- ☐ Ошибки программного обеспечения могут быть связаны с операционной системой, СУБД, прикладными программами или вирусами.

Оборудование

- ☐ К ошибкам, связанным с отказом оборудования, относятся неисправность чипов памяти, выход из строя жесткого диска, испорченные секторы жесткого диска, переполнение диска и т. д.

Программные действия

- ☐ Прикладные программы или конечные пользователи могут отменять транзакции при некоторых условиях. Например, процедура восстановления может инициироваться при попытке снятия клиентом денег с пустого счета или когда пользователь отменяет свои неправильные действия, нажав, например, сочетание клавиш <Ctrl>+<C> при запущенном UNIX-приложении, обновляющем базу данных на уровне оболочки (командный режим).

Транзакции

- Система может выявить наличие тупика и отменить одну из транзакций.

Внешние причины

- Резервное копирование особенно важно в случае полного разрушения системы во время пожара, землетрясения, наводнения и т. д.

В зависимости от типа и масштабов катастрофы процесс восстановления может быть как коротким, так и длительным. Независимо от масштабов процесса восстановления он невозможен без предварительного создания резервной копии.

Процесс восстановления БД обычно происходит по заранее определенному сценарию. Прежде всего, определяются тип и масштаб восстановления. Если необходимо восстанавливать всю БД целиком в устойчивое состояние, то в процессе восстановления используется самая последняя резервная копия устойчивого состояния БД. После этого резервная копия завершает все незаконченные транзакции на основе журнала транзакций. Если базу данных необходимо восстанавливать, но фиксированная часть БД все еще работоспособна, то в процессе восстановления используется журнал транзакций для аннулирования (undo) всех незафиксированных транзакций.

9.6.1. Восстановление транзакции

В разд. 9.1.4 мы рассмотрели структуру журнала и хранение в нем данных для восстановления БД. При восстановлении транзакции БД используются различные методы, применяемые при восстановлении БД из неустойчивого состояния с помощью журнала транзакций. Перед тем как двигаться дальше, рассмотрим четыре важные концепции, влияющие на процесс восстановления.

- *Протокол упреждающей записи.* Этот протокол гарантирует, что журналы транзакций всегда записываются *перед* реальным обновлением данных в БД. Это делается для того, чтобы в случае сбоя можно было впоследствии восстановить базу данных в устойчивое состояние с помощью данных журнала транзакций.
- *Резервные журналы транзакций.* Большинство СУБД хранят несколько копий журнала транзакций, чтобы застраховаться от возможного физического повреждения диска, которое не позволит СУБД произвести восстановление данных.
- *Буферы базы данных.* Буфер — это область основной памяти для временного хранения данных, предназначенная для ускорения операций с диском. Чтобы уменьшить время обработки, программное обеспечение СУБД считывает данные с физического диска и хранит их копию в "буфере" основной памяти. Когда транзакция обновляет данные, она на самом деле обновляет копию данных в буфере, поскольку эта операция намного быстрее, чем получение доступа к данным на физическом устройстве. Впоследствии содержимое всех буферов записывается на физический диск одной операцией, что позволяет значительно экономить время обработки данных.
- *Установка контрольной точки базы данных.* Контрольная точка БД устанавливается, когда СУБД записывает все обновленные буферы на диск. В это время СУБД не обрабатывает никакие другие запросы. Установка контрольной точки регистрируется в журнале транзакций. При этом физическая база данных и журнал транзак-

ций находятся в режиме синхронизации — помнить об этом необходимо, поскольку операции обновления обновляют копии данных в буфере, а не на физическом диске. СУБД автоматически создает несколько контрольных точек в час. Как будет показано впоследствии, контрольные точки также играют важную роль при восстановлении транзакций.

Процесс восстановления БД включает в себя перевод ее в устойчивое состояние после сбоя. Процедуры восстановления транзакций в основном используют отложенную запись или сквозную запись.

Если процедура восстановления использует *отложенную запись (deferred write)* или *отложенное обновление (deferred update)*, то операции транзакции записываются на физический диск не сразу. Вместо этого обновляется только журнал транзакций. База данных физически обновляется только после того, как транзакция достигает точки завершения (commit point), при этом используется информация журнала транзакций. Если транзакция прерывается до достижения точки завершения, то в базе данных не нужно выполнять никаких изменений (с помощью операций ROLLBACK или undo), поскольку база данных физически не обновлялась. Процесс восстановления для всех начатых и завершенных транзакций (до сбоя) выполняется в следующем порядке:

1. Определение последней контрольной точки в журнале транзакций. Это время последнего физического сохранения данных транзакций на диске.
2. Для транзакции, которая была начата и завершена перед последней контрольной точкой, не надо выполнять никаких операций, поскольку все данные о ней уже сохранены.
3. Для транзакции, которая завершилась после контрольной точки, СУБД использует записи в журнале транзакций для ее повторного выполнения и обновления базы данных на основе значений "после" в журнале транзакций. Изменения выполняются в возрастающем порядке от самых старых к самым новым.
4. Не требуется завершать транзакцию, в которой имеется операция ROLLBACK после последней контрольной точки или которая оставалась активной (отсутствуют операции COMMIT или ROLLBACK) перед сбоем, поскольку база данных фактически не обновлялась.

Если процедура восстановления использует *сквозную запись (write-through)* или *немедленное обновление (immediate update)*, то база данных обновляется каждый раз при выполнении операций в рамках транзакции, даже до достижения транзакцией точки завершения. Если транзакция прерывается перед достижением точки завершения, то для восстановления базы данных к устойчивому состоянию необходимо использовать операции ROLLBACK или undo (отмена). В этом случае операция ROLLBACK использует в журнале транзакций значения "после". Процесс восстановления данных выполняется в следующем порядке:

1. Определение последней контрольной точки в журнале транзакций. Это последнее время физического сохранения данных транзакций на диске.
2. Для транзакции, которая была начата и завершена перед последней контрольной точкой, не надо выполнять никаких операций, поскольку все данные о ней уже сохранены.

3. Для транзакции, которая завершилась после последней контрольной точки, СУБД повторяет ее выполнение с помощью значений "после" в журнале транзакций. Изменения выполняются в возрастающем порядке от самых старых к самым новым.
4. Для любой транзакции с операцией ROLLBACK после последней контрольной точки или оставшейся активной перед сбоем (нет операций COMMIT и ROLLBACK) СУБД использует записи в журнале транзакций для операций ROLLBACK и undo (отмена), в которых содержатся значения "перед". Изменения выполняются в обратном порядке от самых новых к самым старым.

Резюме

Транзакция представляет собой последовательность операций, получающих доступ к базе данных. Транзакция отражает реальные события. Транзакция представляет собой логически завершенную единицу работы; т. е. ни одна часть транзакции не может существовать сама по себе — или выполняются все части транзакции или транзакция прерывается. Транзакция переводит базу данных из одного устойчивого состояния в другое. Устойчивое состояние базы данных является одним из условий обеспечения целостности данных.

Транзакции обладают четырьмя основными свойствами.

- ☐ *Атомарность.* Все части транзакции должны быть завершены, иначе транзакция отменяется.
- ☐ *Долговечность.* Изменения, сделанные транзакцией, нельзя отменить после ее завершения.
- ☐ *Сериализуемость.* Результат параллельного выполнения транзакций точно такой же, как если бы они выполнялись последовательно.
- ☐ *Изолированность.* Данные, используемые одной транзакцией, не могут использоваться другой транзакцией до тех пор, пока первая транзакция не будет завершена.

SQL обеспечивает поддержку транзакций с помощью двух операторов:

- ☐ COMMIT — сохраняет данные на физическом носителе;
- ☐ ROLLBACK — восстанавливает предыдущее состояние базы данных.

Транзакция SQL формируется из нескольких SQL-операторов, или запросов к базе данных. Каждый запрос к БД порождает несколько операций чтения/записи.

С помощью журнала транзакций отслеживаются все транзакции, вносящие изменения в базу данных. Информация, хранящаяся в журнале транзакций, используется для восстановления (отката) базы данных в операциях ROLLBACK.

Управление параллельным выполнением транзакций координирует одновременно выполняющиеся транзакции. С параллельным выполнением транзакций связаны три основные проблемы: потеря обновлений, несвязность данных и неоднозначность поиска.

Планировщик отвечает за порядок выполнения одновременно выполняющихся транзакций. Порядок выполнения транзакции имеет существенное значение, т. к. он гарантирует целостность БД в многопользовательских системах. Для обеспечения

сериализуемости (последовательности) транзакций планировщик использует блокировку, метки времени и оптимистические методы.

Блокировка гарантирует уникальность доступа транзакции к элементу данных и предотвращает использование данных одной транзакцией во время их использования другой. Блокировка может выполняться на уровне базы данных, таблицы, страницы, строки или поля.

В системе базы данных применяется блокировка двух типов: двоичная и разделяемая/исключающая. Двоичная блокировка имеет только два состояния: 1 (заблокировано) и 0 (разблокировано). Разделяемая блокировка применяется, когда одна транзакция считывает данные из БД, и нет других транзакций, обновляющих те же данные. Некоторые разделяемые блокировки, или блокировки по чтению, могут существовать для отдельных элементов данных. Исключающая блокировка устанавливается, когда транзакции необходимо обновить данные (произвести запись) в БД и при этом на эти данные не установлена никакая другая блокировка (разделяемая или исключаяющая).

Возможность сериализации планируемых действий гарантируется посредством протокола блокировок, также называемого двухфазной блокировкой. Процесс двухфазной блокировки выглядит так:

1. Фаза роста (growing phase), на которой транзакция получает все необходимые блокировки без разблокирования каких-либо данных.
2. Фаза сжатия (shrinking phase), на которой транзакция снимает все свои блокировки без установки каких-либо новых блокировок.

Когда две или более транзакций неограниченно долго ожидают завершения снятия блокировок, говорят, что они находятся в тупике ("мертвая хватка"). Существуют три метода управления тупиками: предотвращение (prevention), выявление (detection) и уклонение (avoidance).

Управление параллельным выполнением транзакций с помощью оптимистических методов предполагает, что основная часть транзакций БД не конфликтуют друг с другом и транзакции выполняются параллельно, используя индивидуальные копии данных. В процессе завершения индивидуальные копии записываются на физические носители БД.

Восстановление базы данных переводит базу данных из текущего состояния в предыдущее устойчивое состояние. Резервные копии баз данных являются данными, предназначенными для постоянного хранения в безопасном месте, и должны использоваться в случае возникновения критических ошибок в основной базе данных.

Основные термины

Атомарность — atomicity

Буферы — buffers

Восстановление базы данных — database recovery

Выборочное резервное копирование — differential backup

Двоичная блокировка — binary lock

Дисковая страница, или страница памяти — diskpage
Дисковый блок — diskblock
Долговечность — durability
Журнал транзакций — transaction log
Запрос к базе данных — database request
Изолированность — isolation
Исключающая блокировка — exclusive lock
Контрольная точка — check point
Менеджер блокировок — lock manager
Метки времени — time stamping
Монотонность — monotonicity
Немедленное обновление — immediate update
Несвязность данных — uncommitted data
Несогласованный поиск — inconsistent retrievals
Оптимистический метод — optimistic approach
Отложенная запись — deferred write
Отложенное обновление — deferred update
Планировщик — scheduler
Полное резервное копирование — full backup
Потеря обновления — lost update
Протокол упреждающей записи — write-ahead-log protocol
Разделяемая блокировка — shared lock
Резервная копия журнала транзакций — transaction log backup
Резервные копии журнала транзакций — redundant transaction logs
Свойство атомарности транзакций — atomic transaction property
Сериализуемость — serializability
Сквозная запись — write-through
Степень детализации блокировки — lock granularity
Тип блокировки — lock type
Транзакция — transaction
Тупики — deadlocks, deadly embrace
Уникальность — uniqueness
Управление параллельным выполнением — concurrency control
Устойчивое состояние базы данных — consistent database state
Фаза записи — write phase
Фаза подтверждения — validation phase

Фаза роста — growing phase

Фаза сжатия — shrinking phase

Фаза чтения — read phase

Вопросы

1. Поясните следующее положение: транзакция представляет собой логическую единицу работы.
2. Что такое устойчивое состояние базы данных и как его можно достичь?
3. СУБД не гарантирует, что семантическое содержание транзакции должным образом отражает реальные события. Каковы возможные последствия такого ограничения? Приведите пример.
4. Перечислите и поясните четыре основных свойства транзакций.
5. Что такое журнал транзакций и каковы его функции?
6. Что такое планировщик, каково его предназначение и какую роль он играет в управлении параллельным выполнением транзакций?
7. Что такое блокировка и как она работает (в общих чертах)?
8. Что представляет собой управление параллельным выполнением транзакций и какова его цель?
9. Что такое исключаящая блокировка и при каких условиях она предоставляется?
10. Что такое тупик и как его можно избежать? Расскажите о некоторых стратегиях предотвращения тупиков.
11. Какие три уровня резервного копирования используются в управлении восстановлением БД? Кратко опишите предназначение каждого из этих уровней.

Задачи

1. Предположим, что вы производите товар ABC, который состоит из деталей A, B и C. Каждый раз при выпуске единицы товара ABC запас товара (атрибут PROD_QOH в таблице PRODUCT) должен увеличиваться. И каждый раз при выпуске товара запас деталей (атрибут PART_QOH в таблице PART) должен уменьшаться в соответствии с использованием деталей A, B и C. Пример содержимого базы данных приведен в табл. 9.12

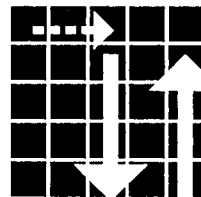
Таблица 9.12. База данных для задачи 1

Таблица PRODUCT		Таблица PART	
PROD_CODE	PROD_QOH	PART_CODE	PART_QOH
ABC	1205	A	567
		B	98
		C	549

На основе этих данных дайте ответы на следующие вопросы.

- Сколько запросов к БД можно определить при обновлении продукции в таблицах PRODUCT и PART?
 - Напишите каждый запрос к БД, который вы определили в предыдущем вопросе с помощью SQL.
 - Полностью опишите транзакцию (транзакции).
 - Создайте журнал транзакций, используя табл. 9.12 в качестве шаблона.
 - С помощью журнала транзакций, созданного вами в предыдущем вопросе, исследуйте его использование при восстановлении БД.
2. Опишите три главные проблемы, возникающие при параллельном выполнении транзакций. Поясните, как с помощью управления параллельным выполнением можно предотвратить эти проблемы.
 3. Какой компонент СУБД отвечает за управление параллельным выполнением транзакций? Как эта функция используется при разрешении конфликтов?
 4. На основе простого примера поясните использование двоичной и разделяемой/исключающей блокировок в СУБД.
 5. Предположим, что ваша база данных повреждена. Опишите процесс восстановления БД и использование методов отложенной и сквозной записи.

Глава 10



Системы управления распределенными базами данных

В этой главе вы узнаете:

- ☐ что такое система управления распределенной базой данных (СУРБД) и что представляют собой ее основные компоненты;
- ☐ как уровень распределения данных и процессов влияет на реализацию БД;
- ☐ как происходит управление транзакциями в среде распределенных баз данных;
- ☐ как конфигурация распределенной базы данных влияет на процесс проектирования БД.

Обзор

В этой главе мы покажем, что обычную базу данных можно разделить на несколько фрагментов. Эти фрагменты могут храниться на различных компьютерах, объединенных в сеть. Обработка данных также может распределяться по нескольким сетевым сайтам, или узлам. База данных, размещенная на нескольких узлах, представляет собой ядро распределенной системы БД.

Разработка и эксплуатация распределенных систем баз данных обусловлена, прежде всего, растущим разбросом бизнес-операций и быстрым развитием современных технологий, сделавших локальные и глобальные сети практичными и надежными. Сетевая распределенная система базы данных очень гибка: ее могут использовать как предприятия малого бизнеса, у которых имеются, например, два магазина в одном городе, так и организации сферы глобального бизнеса.

Хотя разброс базы данных в распределенной системе требует более сложной СУБД, для конечного пользователя работа с системой не должна усложниться. То есть увеличение сложности внутренней структуры распределенной системы базы данных для пользователя должно быть прозрачно.

Система управления распределенной базой данных (СУРБД) рассматривает распределенную базу данных как *единую логическую базу данных*; следовательно, основные концепции проектирования, которые мы рассмотрели в предыдущих главах, сохраняют свое значение. Однако несмотря на то, что конечным пользователям не нужно беспокоиться о специфических свойствах распределенной базы данных, рассредоточение данных по различным сайтам компьютерной сети несомненно увеличивает

сложность системы в целом. Например, при проектировании распределенной базы данных необходимо учитывать местоположение данных и распределение данных по фрагментам БД. В этой главе мы и займемся изучением этих проблемы.

10.1. Этапы развития систем управления распределенными базами данных

Система управления распределенной базой данных (СУРБД) управляет хранением и обработкой логически связанных данных с помощью взаимосвязанных компьютерных систем, в которых как данные, так и функции обработки данных распределены по нескольким сайтам. Чтобы понять, как и чем СУРБД отличается от СУБД, мы кратко проследим развитие инфраструктуры базы данных, определившее современный этап развития СУРБД.

В 70-х годах прошлого века корпорации использовали для обработки информации централизованные системы управления базой данных. Структурированная информация обычно представлялась в виде регулярных отчетов стандартного формата. Такая информация, разработанная с помощью языков программирования третьего поколения (3GL), создавалась специалистами в ответ на точно сформулированные запросы. Таким образом, структурированная информация создавалась централизованными системами.

Доступ к данным осуществлялся через последовательно подключенные неинтеллектуальные терминалы. Централизованный подход был хорош для структурирования информации корпорации, но плохо годился для реагирования на быстро меняющиеся события, требующие немедленного отклика и быстрого доступа к информации. Длительный процесс от запроса информации до ее получения специалистом или пользователем совершенно не устраивал руководство в условиях динамически меняющихся условий. Для оперативного получения информации необходим был быстрый, незапланированный заранее доступ к базе данных с помощью нерегламентированных запросов.

Системы управления базой данных, основанные на реляционной модели, могли предоставить инфраструктуру, в которой потребность в незапланированном доступе к информации удовлетворялась с помощью нерегламентированных запросов. Конечным пользователям была предоставлена возможность получения доступа к данным в любой момент времени. К сожалению, ранние реализации реляционных моделей баз данных имели более низкую производительность по сравнению с имеющимися иерархическими и сетевыми моделями.

В 1980-е годы произошли серьезные изменения в социальной и технологической сферах, сильно повлиявшие на разработку и проектирование баз данных:

- ☐ географическая децентрализация бизнес-операций;
- ☐ рост конкуренции в глобальных масштабах;
- ☐ возросшие потребности рынка и запросы пользователей, отдающие предпочтение децентрализованным системам управления;
- ☐ быстрое развитие технологий, приведшее к созданию недорогих микрокомпьютерных систем с возможностями мэйнфреймов. Это привело к быстрому росту

числа корпораций, использующих локальные сети в качестве основы компьютеризации бизнеса. Бурному развитию локальных сетей способствовал и рост стоимости систем на основе мэйнфреймов;

- большое число приложений, основанных на СУБД, и необходимость сохранить инвестиции, сделанные в развитие централизованных СУБД, сделали идею распределения данных очень привлекательной.

Эти факторы привели к появлению динамической бизнес-среды, в которой компании могли выживать в условиях пресса конкуренции и быстрого развития технологий. По мере преобразования больших бизнес-структур в небольшие и дешевые, быстро реагирующие на запросы подразделения, использующие распределение операций, стали очевидными два требования к базе данных, входящей в такую структуру:

- возможность быстрого нерегламентированного доступа к данным в контексте принятия оперативных решений в ответ на возникающие запросы;
- децентрализация управленческих структур, основанная на децентрализации подразделений бизнеса, требует децентрализованного коллективного доступа к распределенной базе данных.

В 1990-е годы влияние этих факторов еще более усилилось. Однако на способы реализации новых подходов сильное влияние оказали и такие важные явления:

- возрастающее признание Интернета и платформы World Wide Web (Сеть, WWW) в качестве платформы для доступа к данным и их распространения. WWW фактически стала хранилищем распределенных данных (мы рассмотрим влияние Интернета на доступ к данным и их распространение в *гл. 14*);
- повышенное внимание к анализу данных, что привело к появлению специфических технологий анализа информации в БД (data mining) и созданию обширных хранилищ данных (data warehouse).

С этой позиции длительное влияние Сети на проектирование и реализацию распределенных БД неочевидно. Возможно, успех Сети будет стимулировать использование распределенных БД по мере того, как пропускная способность в WWW будет становиться все более узким местом. А может быть, решение проблем с пропускной способностью поможет усилить позиции централизованных БД. В любом случае распределенные базы данных сегодня существуют, и множество их концепций и компонентов скорее всего найдут место в разработке будущих баз данных.

Децентрализованная база данных предпочтительнее централизованной, в которой имеются следующие проблемы:

- *ухудшение производительности* из-за возрастающего числа мест, удаленных на большие расстояния;
- *высокая стоимость*, связанная с обслуживанием и поддержкой центральной базы данных, размещенной на мэйнфрейме;
- *недостаточная надежность*, связанная с зависимостью от центрального сайта.

Динамичность бизнеса и недостатки централизованных БД привели к появлению приложений, основанных на доступе к распределенным данным из различных мест. Базы данных, которые располагаются на нескольких сайтах, при этом доступ к хра-

нящимся в них данным обеспечивается из различных мест, и называются *распределенными базами данных*. Распределенная база данных управляется системой управления распределенной базой данных (СУРБД).

10.1.1. Преимущества СУРБД

Система управления распределенной базой данных по сравнению с традиционными системами обладает рядом преимуществ:

- ❑ *данные располагаются близко к наиболее востребованному сайту.* Данные в распределенной БД распределяются в соответствии с потребностями предприятия;
- ❑ *быстрый доступ к данным.* Конечные пользователи часто работают только с некоторым подмножеством данных компании. При этом подмножество может храниться локально, и система базы данных обеспечит более оперативный доступ к данным, чем централизованная БД, где данные хранятся удаленно;
- ❑ *быстрая обработка данных.* Система распределенной БД обеспечивает возможность обработки данных на нескольких сайтах, тем самым распределяя нагрузку системы;
- ❑ *возможность роста.* К сети можно добавлять новые сайты, не влияя при этом на работу других сайтов. Такая гибкость позволяет компаниям расширяться быстро и относительно просто;
- ❑ *улучшение взаимодействия.* Поскольку локальные сайты невелики по размеру и расположены ближе к клиентам, они обеспечивают лучшее взаимодействие между подразделениями компании, а также между клиентами и компанией в целом. Быстрая и надежная связь зачастую помогает улучшить работу всей информационной системы. Например, при обработке локальных счетов можно использовать данные отдела сбыта напрямую, а не ждать данных из центрального офиса;
- ❑ *уменьшение стоимости операций.* Выгоднее добавлять рабочие станции к сети, чем модернизировать систему, основанную на мэйнфрейме. При этом стоимость выделенных линий связи и программного обеспечения мэйнфрейма пропорционально уменьшается. Разработка может оказаться дешевле и быстрее на недорогих персональных компьютерах, чем на мэйнфрейме. На самом деле, большинство корпораций запрещают выполнять разработки на мэйнфреймах. Эти машины, конечно же, не устранились из поля зрения баз данных, их мощь необходима. Тем не менее, существование сетей персональных компьютеров позволяет распределять нагрузку более рационально и оставлять за мэйнфреймами решение более специализированных задач;
- ❑ *дружественный интерфейс пользователя.* Персональные компьютеры и рабочие станции обычно имеют удобный графический интерфейс пользователя (GUI, Graphical User Interface). Графический интерфейс упрощает работу конечных пользователей и их обучение;
- ❑ *уменьшение опасности сбоя.* В централизованной системе выход мэйнфрейма из строя приведет к остановке всех операций. В отличие от этого, распределенная система позволяет переместить операции с вышедшего из строя компьютера на другой. Нагрузка системы распределяется между другими рабочими станциями

системы, поскольку одно из преимуществ распределенной системы состоит в том, что данные хранятся на нескольких сайтах;

- *независимость от узла обработки данных.* Конечный пользователь получает доступ к любой имеющейся копии данных, а запрос конечного пользователя обрабатывается любым узлом в месте расположения данных. Другими словами, запросы не зависят от конкретного узла обработки данных: любой доступный узел может обработать запрос пользователя.

10.1.2. Недостатки СУРБД

К сожалению, современные системы управления распределенной базой данных обладают рядом недостатков.

- *Сложность управления и контроля.* Управление распределенными данными — более трудная задача, чем управление централизованными данными. Приложения должны определять местонахождение данных и уметь затем связывать во-едино информацию, полученную с разных мест. Администраторы БД должны иметь возможность координировать действия базы данных, чтобы предотвратить ухудшение качества БД из-за аномалий данных. Управление транзакциями, параллельным выполнением, безопасностью, резервное копирование, восстановление, оптимизация запросов, выбор путей доступа и т. д. — всему этому необходимо уделять особое внимание. Другими словами, синхронизация действий всех компонентов распределенной базы данных — очень непростая задача.
- *Безопасность.* Если данные расположены в нескольких местах, то возможность ошибок в обеспечении безопасности возрастает. Ответственность за управление данными распределяется между различными людьми, находящимися в разных местах, а локальные сети все же не обладают такой изолированной системой безопасности, как централизованные системы, основанные на мэйнфреймах.
- *Недостаточная стандартизация.* Несмотря на то что распределенные базы данных зависят от эффективности коммуникаций, пока не существует стандартный протокол обмена информацией *на уровне баз данных*. (Хотя TCP/IP является фактически стандартным протоколом на уровне сетей, отсутствует стандарт протокола на уровне приложений.) Фактически для протоколов распределенных баз данных имеется множество стандартов, как для коммуникаций, так и для управления доступом к данным. К примеру, производители баз данных предлагают различные (зачастую несовместимые) способы управления и обработки распределенных данных в СУРБД. Следовательно, прежде чем пользователи смогут воспользоваться всеми преимуществами распределенных баз данных, им придется подождать (и поспособствовать этому!) появления стандартных протоколов обмена данными.
- *Повышенные требования к условиям хранения данных.* В распределенных БД несколько копий данных необходимо размещать на нескольких сайтах, что требует дополнительных ресурсов (место на жестком диске). Этот недостаток не столь существен, поскольку стоимость хранения информации на жестких дисках быстро уменьшается.

- *Сложность управления средой данных.* Организовать доступ к диску и хранение данных в распределенной среде сложнее, чем в централизованной схеме. Поэтому управление такой средой данных становится более сложным как с точки зрения человеческих ресурсов, так и с точки зрения программного обеспечения.
- *Повышение стоимости обучения.* Стоимость обучения в распределенной модели, как правило, выше, чем в централизованной, и часто сравнима со стоимостью оборудования.

В настоящее время распределенные БД с успехом эксплуатируются, но время, когда можно будет в полной мере использовать гибкость и мощь, которой они теоретически обладают, наступит еще не скоро. Присущая распределенным базам данных сложность требует безотлагательной разработки стандартов на протоколы управления транзакциями, параллельное выполнение, безопасность, резервное копирование и восстановление, оптимизацию, выбор путей доступа и т. д. На такие проблемы следует обратить самое пристальное внимание и постараться решить их до повсеместного распространения технологий СУРБД.

В оставшейся части этой главы мы исследуем основные компоненты и концепции распределенной БД. Поскольку распределенные БД, как правило, основаны на реляционной модели, при их обсуждении мы будем использовать терминологию реляционных баз данных.

10.2. Распределенная обработка данных и распределенные базы данных

При *распределенной обработке данных (distributed processing)* логические процессы базы данных распределяются среди двух или более физически независимых сайтов, объединенных в сеть. Например, распределенная обработка может выполнять ввод/вывод данных, выборку данных и проверку данных на одном компьютере, а затем на другом компьютере выпускать отчет на основе полученной информации.

Среда распределенной обработки данных схематически представлена на рис. 10.1. На этом рисунке показано, что система распределенной обработки данных разделяет рутинные процессы между тремя сайтами, связанными в единую сеть. Хотя база данных размещена только на одном сайте (Майами), каждый сайт может получать доступ к данным и обновлять базу данных. База данных расположена на компьютере А, который называется *сервером базы данных*.

В отличие от этого *распределенная база данных (distributed database)* размещает логически связанную базу данных на двух или более физически независимых сайтах, связанных между собой по компьютерной сети. Система распределенной обработки использует базу данных, размещенную на отдельном сайте, а процессы обработки информации разделяются между несколькими сайтами. В распределенных системах база данных состоит из нескольких частей, которые называются *фрагментами базы данных*. Фрагменты БД располагаются на разных сайтах. Пример конфигурации распределенной базы данных представлен на рис. 10.2.

База данных на рис. 10.2 разделена на три фрагмента (Е1, Е2 и Е3), расположенных на различных сайтах. Компьютеры соединяются друг с другом посредством локаль-

ной сети. В полностью распределенных базах данных пользователям Betty, Hernando и Alan не обязательно знать имя или местоположение каждого фрагмента для того, чтобы получить доступ к БД. К тому же пользователи могут работать на других сайтах (не только Майами, Нью-Йорк или Атланта) и получать доступ к базе данных как к логически единой структуре.

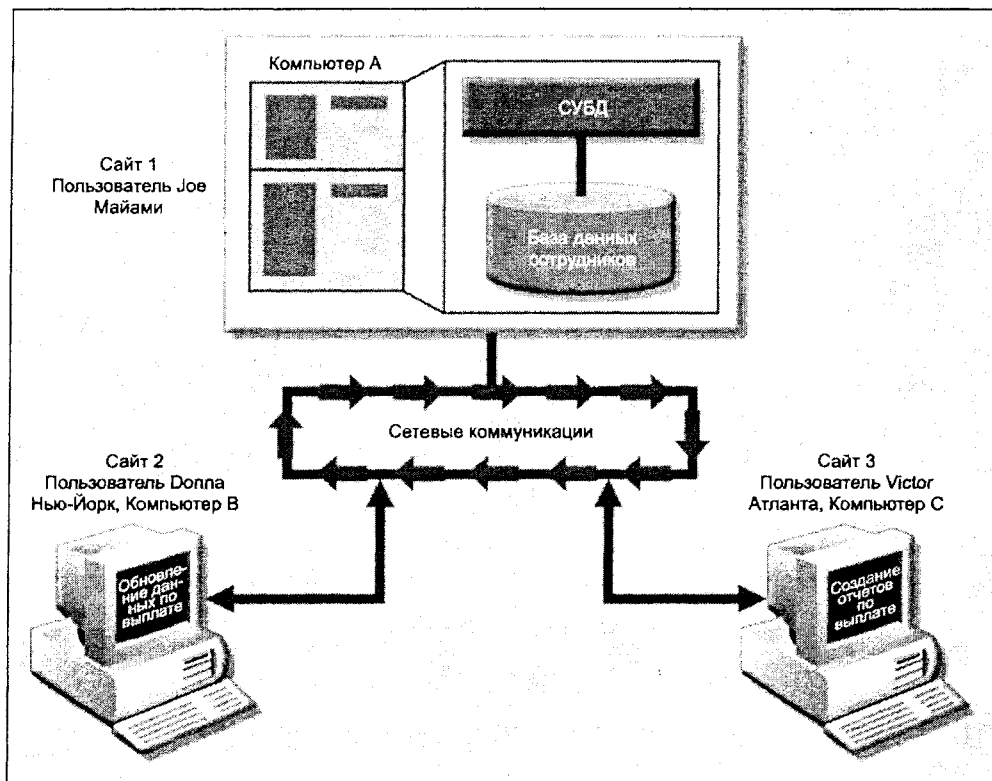


Рис. 10.1. Среда распределенной обработки данных

Если сравнить рис. 10.1 и 10.2, то необходимо обратить внимание на следующее:

- ☐ распределенная обработка данных не требует распределенной базы данных, но распределенная база данных обязательно требует распределенной обработки информации;
- ☐ распределенная обработка данных основана на единственной базе данных, размещенной на одном компьютере. Для того чтобы управлять распределенными данными, копии функций обработки базы данных или их некоторая часть должны быть распространены по всем сайтам, где хранятся данные;
- ☐ и при распределенной обработке данных, и в распределенных базах данных для связывания всех компонентов необходима локальная сеть.

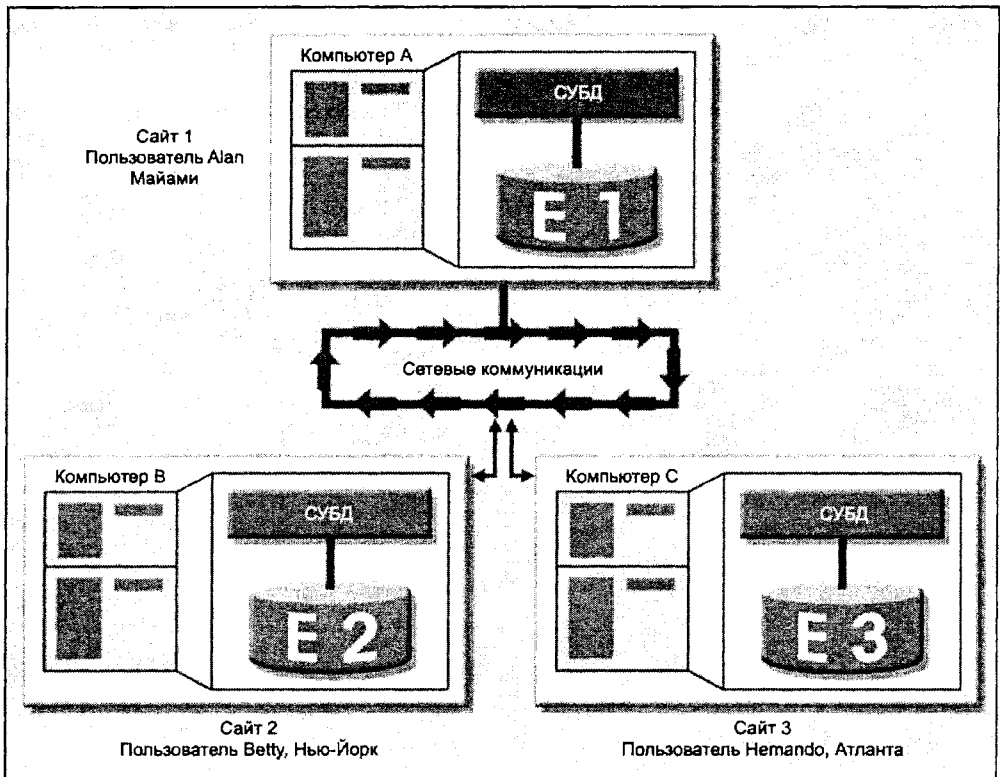


Рис. 10.2. Конфигурация распределенной базы данных

10.3. Что такое система управления распределенной базой данных?

Система управления распределенной базой данных (СУРБД) управляет хранением и обработкой логически связанных данных в сетевых компьютерных системах, где как данные, так и функции обработки распределяются по нескольким сайтам. Чтобы считаться распределенной, СУБД должна обладать, по крайней мере, следующими функциональными возможностями:

- ☐ *прикладной интерфейс*, обеспечивающий взаимодействие с конечным пользователем или прикладными программами, а также с другими СУБД в рамках распределенной базы данных;
- ☐ *проверка достоверности* при анализе запросов;
- ☐ *преобразования* для выяснения, какие компоненты запроса являются распределенными, а какие локальными;

- *оптимизация запроса*, гарантирующая выявление лучшей стратегии доступа (к каким фрагментам БД должен быть обеспечен доступ для выполнения данного запроса, как должно выполняться преобразование данных и как при этом они должны синхронизироваться);
- *отображение*, позволяющее определить местоположение данных в локальных и удаленных фрагментах;
- *интерфейс ввода/вывода*, обеспечивающий считывание/запись данных в постоянном месте хранения;
- *форматирование*, подготавливающее данные для представления их конечному пользователю или для передачи в прикладные программы;
- *безопасность*, т. е. обеспечение конфиденциальности данных как в локальных, так и в удаленных БД;
- *резервное копирование*, которое гарантирует доступность и восстанавливаемость базы данных в случае аварии;
- *управление параллельным выполнением*, обеспечивающее одновременный доступ к данным и гарантирующее целостность данных во всех фрагментах базы данных в данной СУРБД;
- *управление транзакциями*, обеспечивающее переход данных из одного устойчивого состояния в другое. Сюда включаются синхронизация локальных и удаленных транзакций, а также транзакции между несколькими распределенными сегментами.

На рис. 10.3 представлена схема управления взаимодействием между конечными пользователями и базой данных в централизованной СУБД. Эту схему можно использовать в качестве основы для сравнения централизованной СУБД и полностью распределенной СУБД (СУРБД).

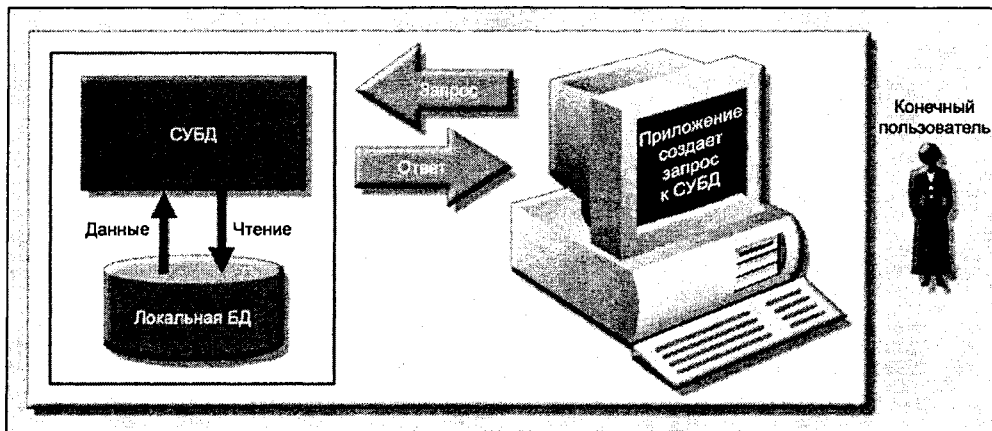


Рис. 10.3. Система управления централизованной базой данных

Из рис. 10.3 следует, что СУБД:

- ☐ получает запросы приложения (или конечного пользователя);
- ☐ проверяет достоверность, анализирует и разбивает запрос на составные части. В запрос могут включаться математические и/или логические операции, например "Выбрать всех клиентов с балансом больше \$1000". Запрос может затребовать данные только из одной таблицы или из нескольких таблиц;
- ☐ отображает логические компоненты запроса на физические данные;
- ☐ разбивает запрос на несколько операций дискового ввода/вывода;
- ☐ осуществляет поиск, определяет местоположение, считывает и проверяет данные;
- ☐ обеспечивает непротиворечивость, безопасность и целостность данных;
- ☐ проверяет соответствие данных заданным условиям (если они определены в запросе);
- ☐ представляет данные в нужной форме.

Все эти действия происходят как бы за сценой, они *прозрачны* для пользователя.

Система управления полностью распределенной базой данных должна выполнять все функции централизованной СУБД и к тому же обеспечивать функциональные возможности, необходимые для работы с распределенными данными и распределенной обработкой информации. Все функции СУРБД также должны быть *прозрачными* для пользователя. Прозрачные функции доступа к данным в СУРБД представлены на рис. 10.4.

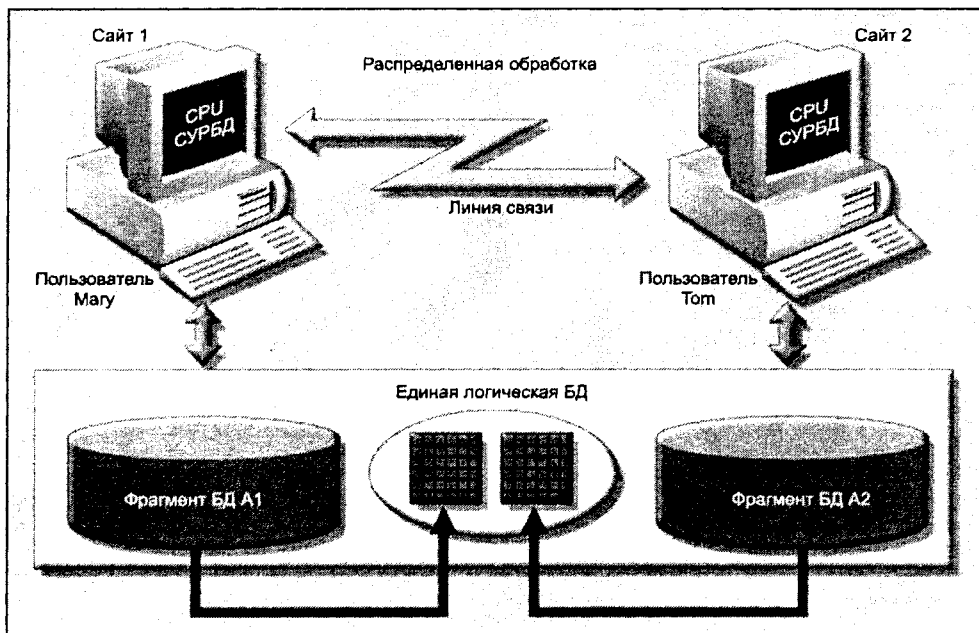


Рис. 10.4. Система управления полностью распределенной базы данных

Логически единая база данных на рис 10.4 состоит из двух фрагментов A1 и A2, расположенных на сайтах 1 и 2 соответственно. Пользователь Mary, как и пользователь Tom, могут запрашивать базу данных, как будто она является локальной БД. Оба пользователя "видят" только одну логическую БД и не знают имена фрагментов. На самом деле, конечным пользователям нет необходимости знать ни о том, что БД разделена на отдельные фрагменты, ни о том, где эти фрагменты расположены.

Разделение между обработкой данных и их хранением приводит к нескольким сценариям, в которых с помощью СУБД достигается некоторая степень распределения (обработки или данных, либо и того и другого). Для лучшего понимания различных типов сценариев распределения БД мы должны, прежде всего, определить компоненты системы распределенной базы данных.

10.4. Компоненты СУРБД

В состав СУРБД должны входить (по крайней мере) следующие компоненты:

- ❑ *компьютерные рабочие станции* (сайты или узлы), формирующие сетевую систему. Система распределенной базы данных должна быть независимой от оборудования;
- ❑ *компоненты сетевого оборудования и программного обеспечения* каждой рабочей станции. Сетевые компоненты позволяют всем сайтам взаимодействовать друг с другом и обмениваться данными. Поскольку эти компоненты (компьютеры, операционные системы, сети и т. д.), скорее всего, поставляются различными производителями, желательно, чтобы функции распределенной БД могли выполняться на различных платформах;
- ❑ *коммуникационные устройства*, которые переносят данные с одной рабочей станции на другую. СУРБД не должна зависеть от средств коммуникации, т. е. она должна поддерживать несколько типов коммуникационных устройств;
- ❑ *процессор транзакций (transaction processor, TP)*, представляющий собой программный компонент, находящийся на каждом компьютере, где выполняется запрос данных. Процессор транзакций получает и обрабатывает данные запроса приложения (удаленные и локальные). Процессор транзакций называют также *процессором приложений (application processor, AP)* или *менеджером транзакций (transaction manager, TM)*;
- ❑ *процессор данных (data processor, DP)*, представляющий собой программный компонент, расположенный на каждом компьютере, где хранятся и извлекаются данные, расположенные на данном сайте. Процессор данных также называют *менеджером данных (data manager, DM)*. Процессор данных может даже представлять собой централизованную СУБД.

Рис. 10.5 иллюстрирует расположение и взаимодействие всех компонентов. Связь между TP и DP, показанная на рис. 10.5, позволяет установить сквозные специфические правила или *протоколы*, используемые в СУРБД.

Протоколы определяют, как система распределенной базы данных:

- ❑ организует интерфейс с сетью для передачи данных и команд между процессорами данных (DP) и процессорами транзакций (TP);

- синхронизирует все данные, полученные от DP (сторона TP), и маршрутизирует полученные данные на соответствующие TP (сторона DP);
- обеспечивает функции общего управления БД в распределенной системе (безопасность, управление параллельным выполнением, создание резервных копий и восстановление).

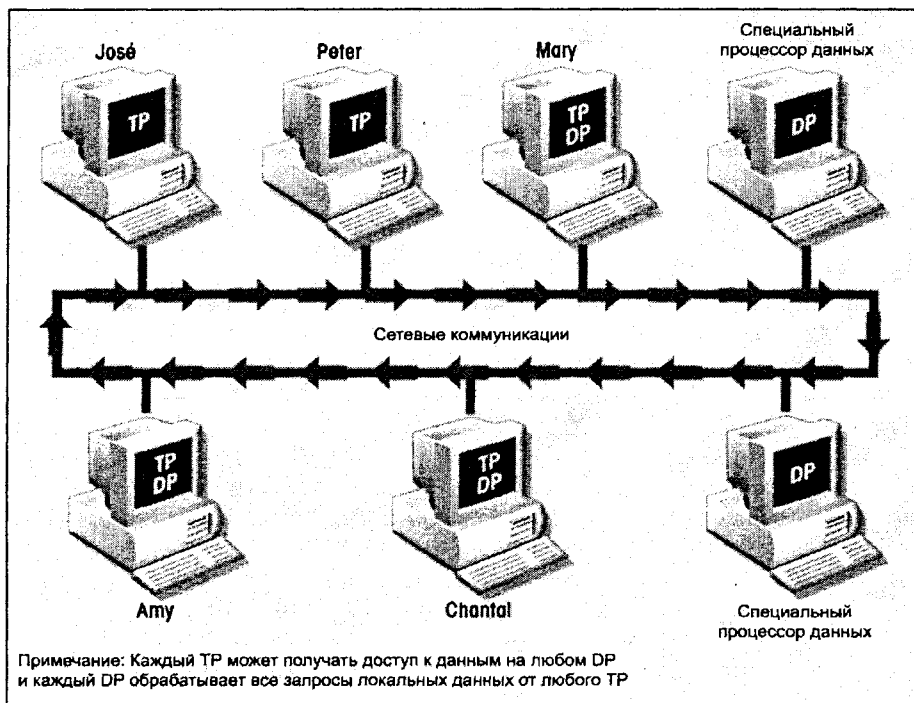


Рис. 10.5. Компоненты системы распределенной базы данных

Процессор данных (DP) и процессор транзакций (TP) можно добавить в систему, не воздействуя на ее другие компоненты. Процессоры TP и DP могут размещаться на одном и том же компьютере, позволяя конечным пользователям получать доступ к локальным и удаленным данным, не заботясь об их местонахождении. Теоретически DP может представлять собой независимую централизованную СУБД с соответствующим интерфейсом для поддержки удаленного доступа к другим независимым СУБД в сети.

10.5. Уровни распределения данных и их обработки

Можно классифицировать современные базы данных по способам поддержки ими распределения данных и их обработки. Например, СУБД может хранить данные на

единственном сайте (централизованная БД) или на нескольких сайтах (распределенная БД), а также может поддерживать обработку данных на одном сайте или на нескольких сайтах. В табл. 10.1 представлена простая классификация баз данных в соответствии с распределением данных и процессов их обработки. Типы процессов обработки мы обсудим в последующих разделах этой главы.

Таблица 10.1. Системы баз данных: уровни распределения данных и процессов их обработки

	Размещение данных на одном сайте	Размещение данных на нескольких сайтах
Обработка данных на одном сайте	Основная СУБД (на мэйн-фреймах)	Невозможна (требуется несколько процессов на разных сайтах)
Обработка данных на нескольких сайтах	Файл-сервер, клиент/серверная СУБД (СУБД в локальной сети)	Полностью распределенная клиент/серверная СУРБД

10.5.1. Обработка и размещение данных на одном сайте

При таком сценарии вся обработка данных выполняется одним процессором (CPU) или главным компьютером (мэйнфреймом, мини-компьютером или персональным компьютером — хост-компьютером), а все данные хранятся на локальном диске хост-компьютера. Обработка не может выполняться на стороне компьютера конечного пользователя. Такой сценарий обычен для большинства СУБД на мэйнфреймах или мини-компьютерах. СУБД размещена на хост-компьютере, к которому можно получать доступ с подключенных к нему терминальных устройств (рис. 10.6). Такая схема также типична для первого поколения однопользовательских баз данных на микрокомпьютерах.

Из рис. 10.6 следует, что функции процессора транзакций (TP) и процессора данных (DP) встроены в СУБД, размещенную на одном компьютере. СУБД обычно работает под управлением многозадачной операционной системы с разделением времени, позволяющей нескольким процессам выполняться на основном процессоре (host CPU) "одновременно", получая доступ к единому процессору данных. Все операции, связанные с хранением и обработкой данных, выполняются одним центральным процессором.

10.5.2. Обработка данных на нескольких сайтах, размещение данных на одном сайте (MPSD)

Если обработка данных ведется на нескольких сайтах (узлах), а данные хранятся на одном сайте (multiple-site processing, single-site data, MPSD), то несколько процессов обработки данных выполняются на разных компьютерах, которые получают совместный доступ к хранилищу информации.

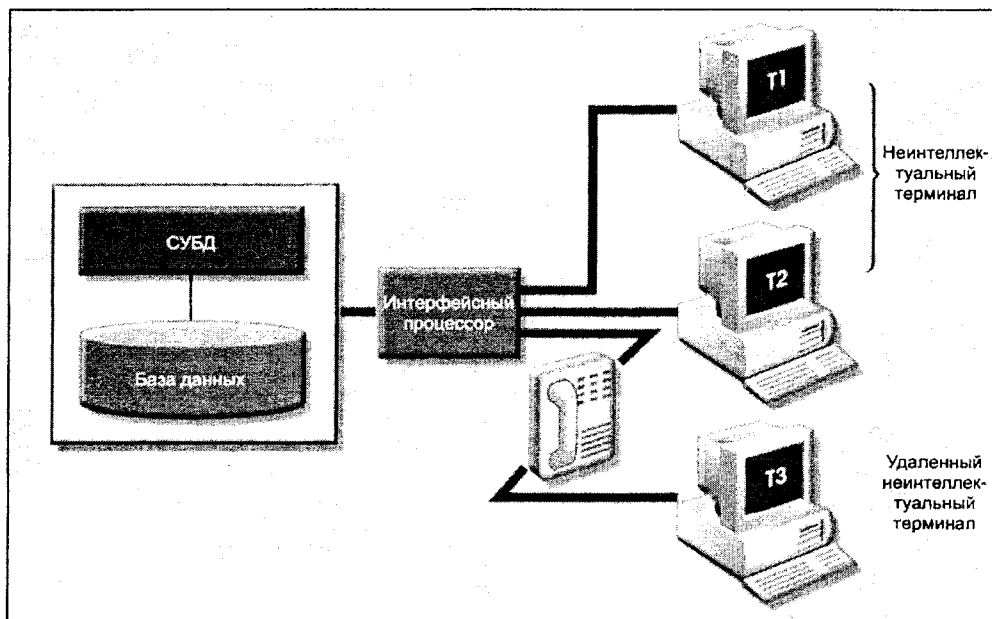


Рис. 10.6. Нераспределенная (централизованная) СУБД

Обычно такой сценарий требует наличия сетевого файл-сервера, на котором выполняются определенные приложения, доступные по локальной сети. Множество многопользовательских бухгалтерских программ, работающих в локальной сети персональных компьютеров, действуют именно по такому сценарию (рис. 10.7).

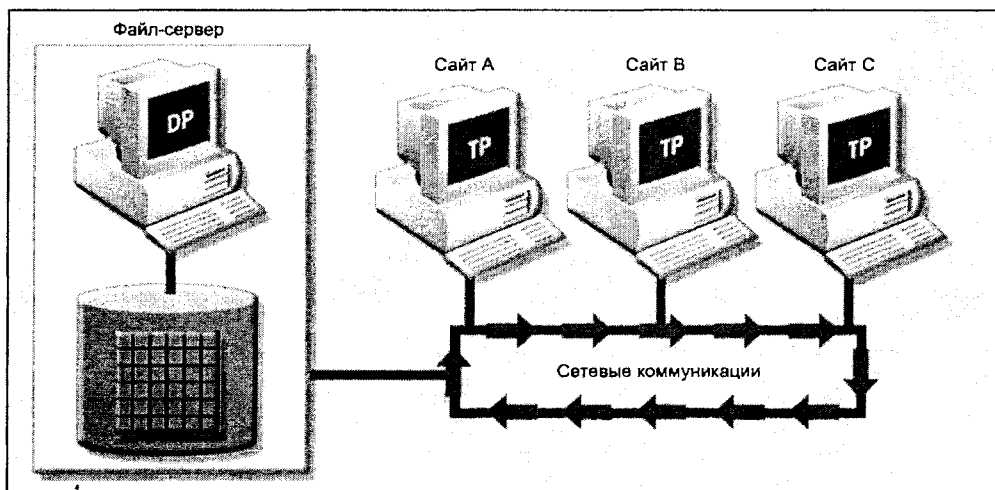


Рис. 10.7. Обработка данных на нескольких сайтах, данные на одном сайте

На рис. 10.7 необходимо обратить внимание на следующее:

- процессор транзакций на каждой рабочей станции действует только как редиректор (программа переназначения), направляющий всю информацию сетевых запросов на файл-сервер;
- конечный пользователь видит файл-сервер просто как еще один жесткий диск. Поскольку ввод/вывод хранимых данных обрабатывается только файл-сервером, в этом случае возможности распределенной обработки очень ограничены;
- чтобы получить доступ к данным, конечный пользователь должен напрямую ссылаться на файл-сервер. Все действия по блокировке файлов выполняются на компьютере конечного пользователя;
- функции выборки, поиска и обновления данных выполняются на рабочих станциях, при этом требуется, чтобы весь файл целиком передавался по сети на рабочую станцию для обработки. Это требование приводит к увеличению сетевого трафика, уменьшает время отклика и увеличивает стоимость коммуникаций.

Последнее положение легко продемонстрировать. Предположим, что на файл-сервере хранится таблица CUSTOMER (клиент), содержащая 10 000 строк информации, в 50 из которых сумма баланса больше \$1000. Если на сайте А создать такой запрос:

```
SELECT *  
FROM CUSTOMER  
WHERE CUS_BALANCE > 1000
```

то все 10 000 записей таблицы CUSTOMER будут переданы по сети для расчетов на сайт А.

Один из вариантов сценария обработки данных на нескольких сайтах и размещения данных на одном сайте называется *архитектурой клиент/сервер*. Эта схема, которую мы будем подробно обсуждать в гл. 12, в целом похожа на схему с файл-сервером, за исключением того, что *вся обработка данных выполняется на сайте сервера, что позволяет уменьшить сетевой трафик*. Несмотря на то что и в файл-серверной схеме, и в архитектуре клиент/сервер обработка выполняется на нескольких сайтах, распределение обработки имеет место только в схеме клиент/сервер. Обратите внимание, что в схеме с сетевым файл-сервером требуется, чтобы база данных размещалась на одном сайте. В отличие от этого в архитектуре клиент/сервер возможна поддержка распределения данных по нескольким сайтам.

10.5.3. Обработка и размещение данных на нескольких сайтах (MPMD)

Этот сценарий относится к системе управления полностью распределенной базой данных с поддержкой нескольких процессоров данных и процессоров транзакций на нескольких сайтах (multiple-site processing, multiple-site data, MPMD). В зависимости от уровня поддержки различных типов централизованных СУБД, СУРБД подразделяются на гомогенные и гетерогенные.

Гомогенные системы распределенных баз данных объединяют по сети только один тип централизованных СУБД. В этом случае на разных мэйнфреймах, мини-компьютерах

и микрокомпьютерах будут выполняться одинаковые СУБД. В отличие от этого *гетерогенные системы распределенных баз данных* объединяют по сети различные типы централизованных СУБД (рис. 10.8). Полностью гетерогенная СУРБД поддерживает различные СУБД, которые, возможно, даже используют различные модели данных (реляционные, иерархические или сетевые) и выполняются в разных системах, например, на мэйнфреймах, мини-компьютерах и микрокомпьютерах.





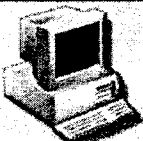
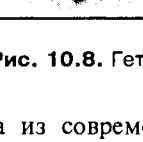
	Платформа	СУБД	Операционная система	Сетевой протокол
	IBM 3090	DB2	MVS	APPC LU 6.2
	DEC/VAX	VAX rdb	MVS	DECnet
	IBM AS/400	SQL/400	OS/400	3270
	RISC-компьютер	Informix	UNIX	TCP/IP
	Pentium CPU	Oracle	Windows 2000	NetBIOS

Рис. 10.8. Гетерогенная система управления распределенной базой данных

Ни одна из современных СУРБД не обеспечивает полную поддержку сценария, представленного на рис. 10.8, и не предоставляет для этого соответствующие инфраструктуры. Некоторые реализации СУРБД обеспечивают поддержку нескольких аппаратных платформ, операционных систем и сетей и позволяют выполнять доступ к данным другой СУБД. Однако в подобных СУРБД все еще имеется множество ограничений. Например:

- ☐ удаленный доступ предоставляется на уровне "только чтение" и не обеспечивает возможность записи данных;

- ограничивается количество удаленных таблиц, доступных в одной транзакции;
- ограничивается число отдельных баз данных, к которым можно получить доступ;
- ограничиваются модели баз данных, к которым можно получать доступ. То есть, например, доступ может осуществляться только к реляционным базам данных, но не к сетевым или иерархическим.

Представленный список ни в коем случае нельзя считать окончательным. Технология СУРБД продолжает быстро развиваться, и все время открываются новые возможности. Управление данными на нескольких сайтах приводит к возникновению множества проблем, которые необходимо выявлять и понимать. Поэтому далее мы рассмотрим лишь несколько ключевых свойств систем управления распределенными базами данных.

10.6. Прозрачные свойства распределенной базы данных

Система распределенной базы данных обладает целым рядом функциональных характеристик, которые можно назвать прозрачными (невидимыми пользователю). *Прозрачные свойства СУРБД* обладают общей особенностью, позволяя конечному пользователю считать себя монопольным пользователем системы. Другими словами, пользователь полагает, что он работает с централизованной СУБД, и при этом все сложности работы с распределенной БД скрыты от него (прозрачны).

К прозрачным свойствам СУРБД относятся:

- *прозрачность распределения (distribution transparency)*, позволяющая считать распределенную БД единой логической базой данных. Если СУРБД обеспечивает прозрачность распределения, то пользователя совершенно не интересует:
 - что данные разделяются по нескольким сайтам;
 - что данные дублируются на нескольких сайтах;
 - где размещены данные.
- *прозрачность транзакций (transaction transparency)*, которая позволяет транзакции обновлять данные на нескольких сетевых сайтах. Свойство прозрачности транзакции гарантирует, что данная транзакция будет либо выполнена полностью, либо отменена, что должно обеспечить целостность данных;
- *прозрачность ошибок (failure transparency)*, которая гарантирует, что система будет продолжать выполнение операций в случае неисправности какого-либо узла (сайта). Функции, не выполненные по причине сбоя, будут завершены на других узлах;
- *прозрачность производительности (performance transparency)*, которая позволяет системе функционировать как централизованной СУБД. Качество работы системы не должно ухудшаться из-за того, что она работает на различных сетевых платформах. Прозрачность производительности также гарантирует, что система будет находить наиболее эффективный и выгодный путь доступа к удаленным данным;

- *прозрачность гетерогенности (heterogeneity transparency)*, которая позволяет объединять несколько различных локальных СУБД (реляционных, сетевых и иерархических) в единую или глобальную схему. СУРБД отвечает за перевод запросов данных из глобальной в локальную схему СУБД.

В последующих разделах мы подробно рассмотрим свойства прозрачности распределения, прозрачности транзакций и прозрачности производительности.

10.7. Прозрачность распределения

Прозрачность распределения позволяет управлять физически разнесенной базой данных так, как если бы она была централизованной БД. Уровень прозрачности, обеспечиваемый СУРБД зависит от системы. Различают три уровня прозрачности:

- *прозрачность фрагментации*. Это наивысший уровень прозрачности. Конечный пользователь или программист может ничего не знать о разделении базы данных. Поэтому при доступе к данным не задаются ни имя фрагмента, ни его местоположение;
- *прозрачность местоположения* имеет место, когда конечный пользователь или программист должен задавать имя фрагмента БД, но местоположение фрагмента задавать не нужно;
- *прозрачность локального отображения*. При этом конечный пользователь или программист должен определять как имя фрагмента БД, так и его местоположение.

Свойства прозрачности сведены в табл. 10.2.

Таблица 10.2. Свойства прозрачности

Если в SQL-операторе требуется указывать:			
имя фрагмента	местоположение	СУРБД поддерживает	Уровень прозрачности распределения
Да	Да	Прозрачность локального отображения	Низкий
Да	Нет	Прозрачность местоположения	Средний
Нет	Нет	Прозрачность фрагментации	Высокий

Можно спросить, почему в табл. 10.2 не упомянута ситуация, при которой в столбце "имя фрагмента" указано "Нет", а в столбце "местоположение" — "Да". Причина проста: невозможна ситуация, при которой существует местоположение, не соответствующее какому-нибудь фрагменту (если не нужно задавать фрагмент, то, очевидно, нет нужды задавать и его местоположение).

Для иллюстрации использования различных уровней прозрачности предположим, что имеется таблица EMPLOYEE (сотрудник), в которой есть атрибуты EMP_NAME, EMP_DOB, EMP_ADDRESS, EMP_DEPARTMENT и EMP_SALARY. Данные атрибута EMPLOYEE распределены по трем местам: Нью-Йорк, Атланта и

Майами. Таблица разделяется на три части; т. е. данные о сотрудниках в Нью-Йорке хранятся во фрагменте E1, о сотрудниках Атланты — во фрагменте E2, а о сотрудниках Майами — во фрагменте E3 (рис. 10.9).

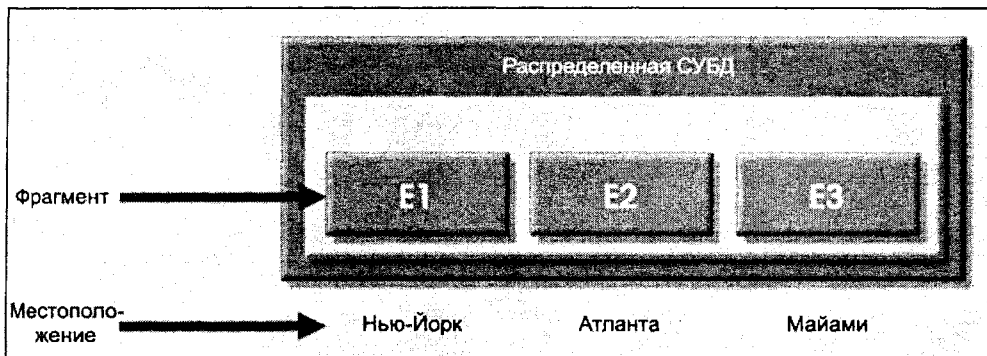


Рис. 10.9. Расположение фрагментов

Теперь предположим, что конечный пользователь хочет получить список всех сотрудников с датой рождения до 1 января 1940 года. Чтобы обратить внимание именно на проблемы прозрачности, предположим, что таблица EMPLOYEE фрагментирована и каждый фрагмент уникален (условие уникальности фрагмента указывает на то, что все без исключения строки уникальны, независимо от того, в каком фрагменте они находятся). Наконец, предположим, что ни одна часть базы данных не дублируется на каком-либо другом сайте сети.

В зависимости от уровня прозрачности распределения мы рассмотрим три случая.

Случай 1.

База данных поддерживает прозрачность фрагментации

Запрос соответствует формату запроса к нераспределенной БД, т. е. в нем не определяются имя фрагмента и его местоположение. Запрос выборки выглядит так:

```
SELECT *  
FROM EMPLOYEE  
WHERE EMP_DOB < '01-JAN-1940';
```

Случай 2.

База данных поддерживает прозрачность местоположения

В запросе необходимо определять имя фрагмента, местоположение фрагмента не задается:

```
SELECT *  
FROM E1  
WHERE EMP_DOB < '01-JAN-1940';
```

```
UNION
SELECT *
  FROM E2
  WHERE EMP_DOB < '01-JAN-1940';
```

```
UNION
SELECT *
  FROM E3
  WHERE EMP_DOB < '01-JAN-1940';
```

Случай 3.

База данных поддерживает прозрачность локального отображения

В запросе необходимо задавать и имя фрагмента, и местоположение. На псевдо-SQL запрос можно записать в таком виде:

```
SELECT *
  FROM E1 NODE NY
  WHERE EMP_DOB < '01-JAN-1940';
```

```
UNION
SELECT *
  FROM E2 NODE ATL
  WHERE EMP_DOB < '01-JAN-1940';
```

```
UNION
SELECT *
  FROM E3 NODE MIA
  WHERE EMP_DOB < '01-JAN-1940';
```

Примечание

Слово **NODE** указывает местоположение фрагмента БД, используется только для иллюстрации и не является частью синтаксиса SQL.

В только что приведенном формате запроса видно, как влияет прозрачность распределения на способ взаимодействия конечных пользователей и программистов с БД.

Прозрачность распределения обеспечивается *словарем распределенных данных* (*distributed data dictionary, DDD*) или *каталогом распределенных данных* (*distributed data catalog, DDC*). DDC содержит описание всей базы данных с точки зрения администратора БД. Описание базы данных называется *схемой глобального распределения* (*distributed global schema*) и представляет собой глобальную схему базы данных, используемую локальными процессорами транзакций (TP) для трансляции запросов пользователей в подзапросы (удаленные запросы), которые будут обрабатываться

различными процессорами данных (DP). Каталог DDC сам по себе является распределенным и дублируется на узлах сети. Поэтому необходимо постоянно обслуживать каталог DDC путем обновления всех сайтов.

Помните, что в некоторых реализациях современных СУРБД устанавливаются ограничения на уровень прозрачности. Например, СУРБД может допускать распределение по сайтам базы данных, но не отдельной таблицы. Такое условие означает, что СУБД поддерживает прозрачность местоположения, но не поддерживает прозрачность фрагментации.

10.8. Прозрачность транзакций

Прозрачность транзакций является свойством СУРБД, которое гарантирует, что все транзакции БД будут обеспечивать целостность и непротиворечивость распределенной базы данных. Необходимо помнить, что транзакции СУРБД могут обновлять данные, хранящиеся на различных компьютерах, объединенных в сеть. Свойство прозрачности транзакций гарантирует, что транзакция будет завершена только в том случае, если на всех сайтах базы данных, вовлеченных в транзакцию, будут завершены все части этой транзакции.

В системах распределенных баз данных для управления транзакциями и обеспечения целостности и непротиворечивости базы данных используется сложный механизм. Для того чтобы понять, как происходит управление транзакциями, необходимо знать основные концепции удаленных запросов, удаленных транзакций, распределенных транзакций и распределенных запросов.

10.8.1. Распределенные запросы и распределенные транзакции

Является транзакция распределенной или нет, в любом случае она формируется на базе одного или нескольких запросов. Основное отличие нераспределенных транзакций от распределенных состоит в том, что последние могут обновлять или запрашивать данные на нескольких удаленных сайтах сети¹. Для более наглядной иллюстрации концепций распределенных транзакций сначала установим различие между удаленной и распределенной транзакциями с помощью форматов транзакций BEGIN WORK и COMMIT WORK. Чтобы не указывать местоположение данных, предположим, что поддерживается прозрачность местоположения.

Удаленный запрос (remote request), представленный на рис. 10.10, позволяет получить доступ к данным, которые будут обрабатываться одним удаленным процессором базы данных. Другими словами, SQL-оператор (или запрос) может ссылаться на данные, расположенные только на одном удаленном сайте.

Точно так же *удаленная транзакция (remote transaction)*, составленная из нескольких запросов, может получать доступ к данным, размещенным только на одном сайте. Удаленная транзакция представлена на рис. 10.11.

¹ Для получения дополнительной информации об этих концепциях см. David McGoveran и Colin White, "Clarifying Client-Server", *DBMS* 3(12), ноябрь 1990, стр. 78—89.

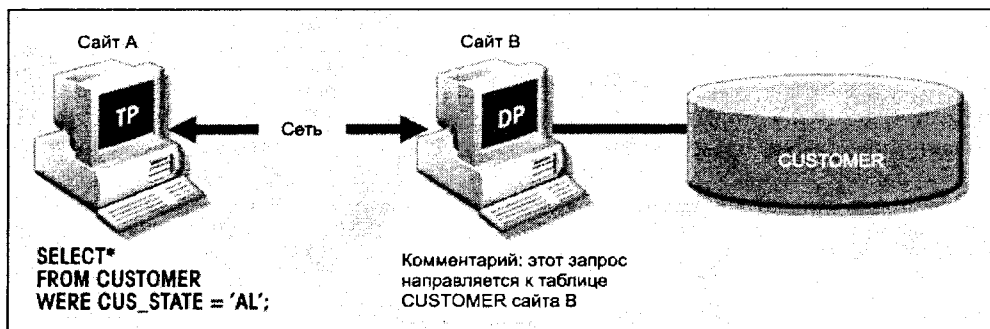


Рис. 10.10. Удаленный запрос

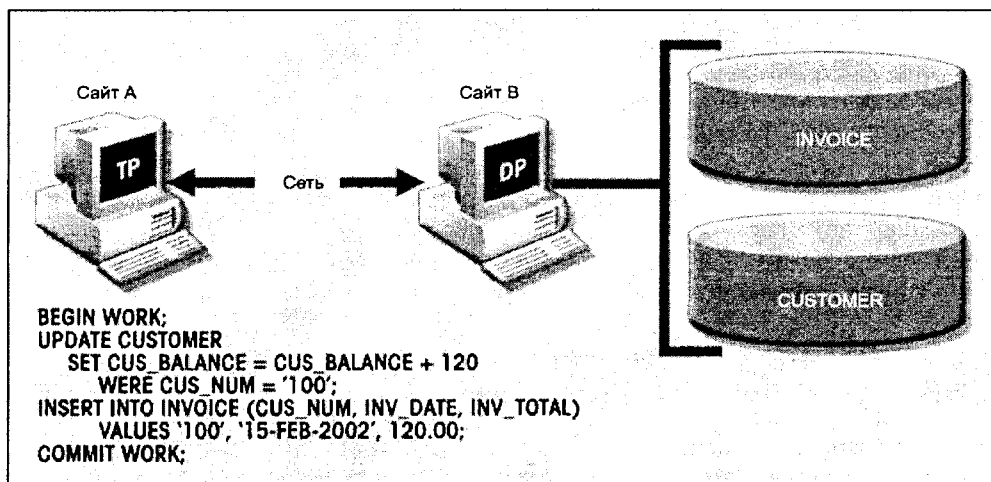


Рис. 10.11. Удаленная транзакция

На рис. 10.11 обратите внимание на следующие особенности удаленной транзакции:

- ☐ транзакция обновляет таблицы CUSTOMER и INVOICE;
- ☐ обе таблицы размещены на Сайте В;
- ☐ транзакция может ссылаться только на один процессор данных (DP);
- ☐ каждый оператор SQL (или запрос) может ссылаться в данный момент времени только на один (один и тот же) удаленный процессор данных (DP), и вся транзакция может ссылаться только на один и выполняться только одним удаленным DP.

Распределенная транзакция (distributed transaction) позволяет ссылаться на несколько различных (локальных или удаленных) сайтов DP. Хотя каждый простой запрос может ссылаться только на один удаленный сайт DP, транзакция в целом может ссылаться на несколько сайтов DP, поскольку каждый запрос может ссылаться на различные сайты. Процесс распределенной транзакции представлен на рис. 10.12.

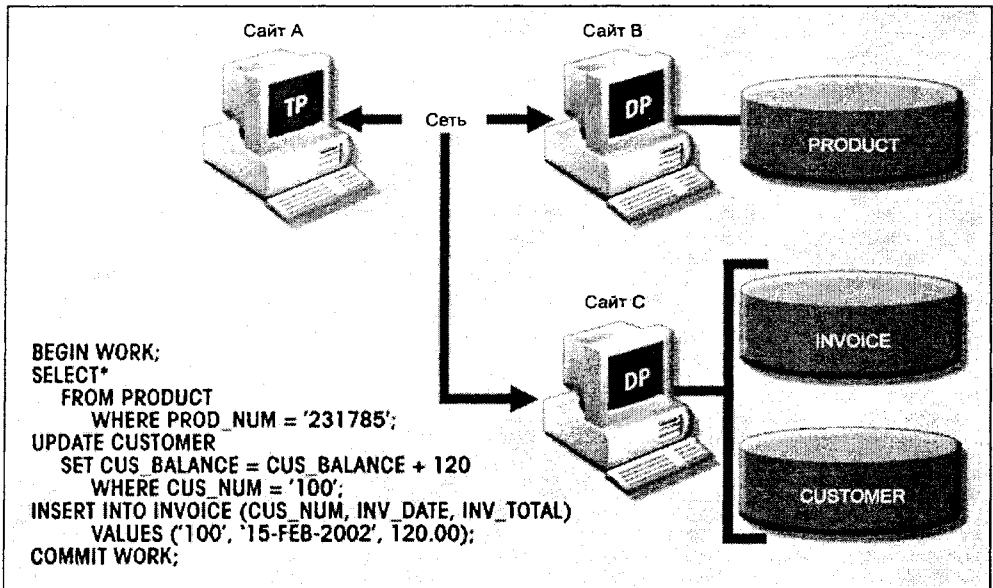


Рис. 10.12. Распределенная транзакция

Обратите внимание на следующие особенности на рис. 10.12:

- транзакция ссылается на два удаленных сайта (B и C);
- первый запрос (оператор SELECT) обрабатывается процессором данных на удаленном сайте Сайт B, а следующие запросы (UPDATE и INSERT) обрабатываются DP на удаленном сайте (C);
- каждый запрос в один момент времени может получить доступ только к одному удаленному сайту.

Третий пункт может создавать проблемы. Предположим, что таблица PRODUCT разделяется на два фрагмента PROD1 и PROD2, расположенные на сайтах B и C соответственно. При этом сценарии предыдущая распределенная транзакция выполняться не будет, поскольку запрос

```

SELECT *
  FROM PRODUCT
   WHERE PROD_NUM = '231785';

```

не сможет получить доступ к данным на более чем одном удаленном сайте. Поэтому в этом случае СУБД должна обеспечивать поддержку распределенных запросов.

Распределенный запрос (distributed request) позволяет получать данные от нескольких удаленных сайтов с DP. Поскольку каждый запрос может получить доступ к данным, расположенным более чем на одном сайте, транзакция может получить доступ к нескольким сайтам.

Возможность выполнять распределенный запрос предоставляет только полностью распределенная база данных, поскольку здесь мы можем:

- ☐ разбить базу данных на несколько фрагментов;
- ☐ ссылаться на один или более таких фрагментов из одного запроса, т. е. использовать прозрачность фрагментации.

Размещение и разбиение данных должно быть прозрачно для конечного пользователя. На рис. 10.3 представлен распределенный запрос. Здесь нужно обратить внимание на то, что транзакция для ссылки на две таблицы **CUSTOMER** и **INVOICE** использует единственный оператор **SELECT**. Эти две таблицы расположены на разных сайтах **B** и **C**.

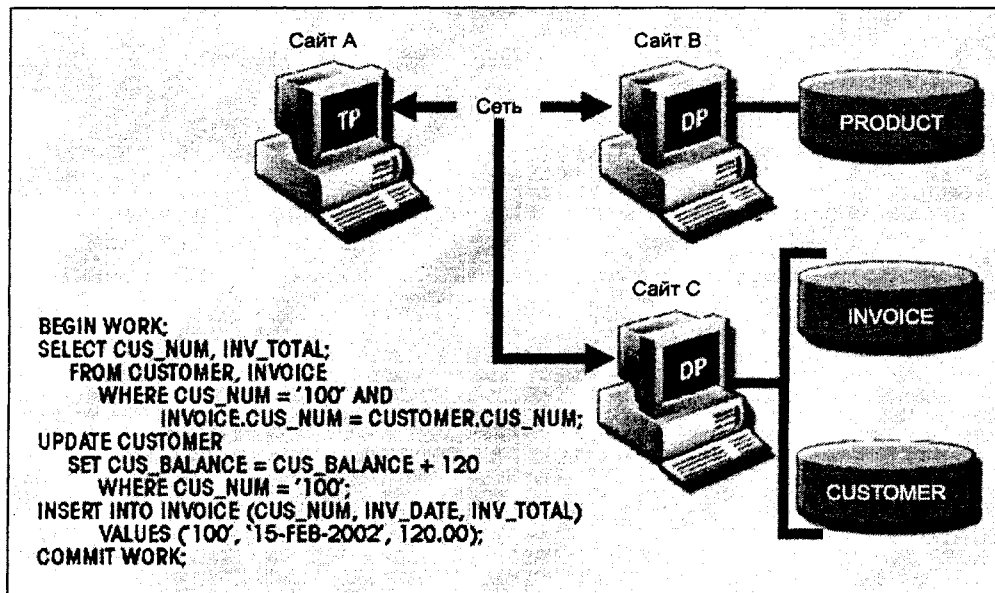


Рис. 10.13. Распределенный запрос

Возможность создания распределенного запроса позволяет в одном запросе обращаться к физически разделенной таблице. Предположим, что таблица **CUSTOMER** разделена на два фрагмента **C1** и **C2**, расположенные на сайтах **B** и **C** соответственно. Теперь допустим, что конечный пользователь хочет получить список всех клиентов, чей баланс превышает \$250. Этот запрос проиллюстрирован на рис. 10.14. Полная поддержка прозрачности фрагментов предоставляется только в СУРБД, которые поддерживают распределенные запросы.

Осмысление различных типов запросов к базе данных в распределенных системах поможет понять проблемы прозрачности транзакций. Свойство прозрачности транзакций гарантирует, что все транзакции можно рассматривать как централизованные, а также обеспечивает их сериализуемость (вспомните обсуждение в гл. 9), т. е.

при одновременном выполнении транзакций (независимо от того, являются они распределенными или нет) база данных будет переходить из одного устойчивого состояния в другое. Далее мы рассмотрим управление параллельным выполнением распределенных транзакций.

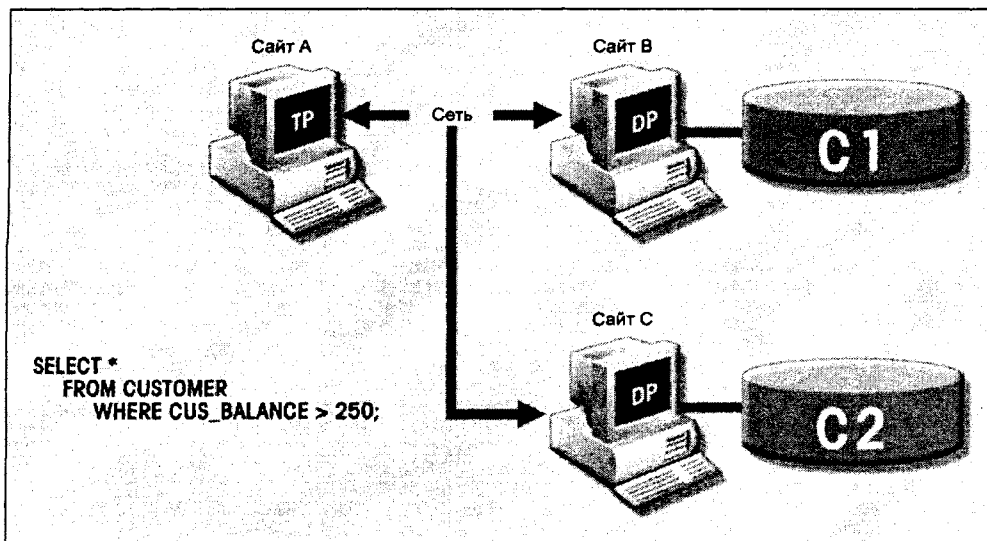


Рис. 10.14. Еще один распределенный запрос

10.8.2. Управление параллельным выполнением в распределенной среде

Управление параллельным выполнением становится особенно значимым в среде распределенной базы данных, поскольку многоместные (на нескольких сайтах) и многопроцессорные операции с большей вероятностью могут привести к противоречивости данных и тупикам, чем одноместные (выполняющиеся на одном сайте) системы. Например, компонент TP (процессор транзакций) СУРБД должен гарантировать, что все части транзакции на всех сайтах будут завершены до того, как последний оператор COMMIT завершит всю транзакцию в целом.

Предположим, что каждая операция транзакции подтверждалась локальным процессором данных (DP), но один из DP не смог записать результаты транзакции. Это может привести к проблемам (рис. 10.15): транзакция (транзакции) приведет к противоречивому состоянию БД и неизбежным проблемам с целостностью, поскольку мы не можем отменить уже записанные данные! Решение проблемы, представленной на рис. 10.15, состоит в использовании *протокола двухфазного подтверждения транзакции (two-phase commit protocol)*, который мы сейчас подробно обсудим.

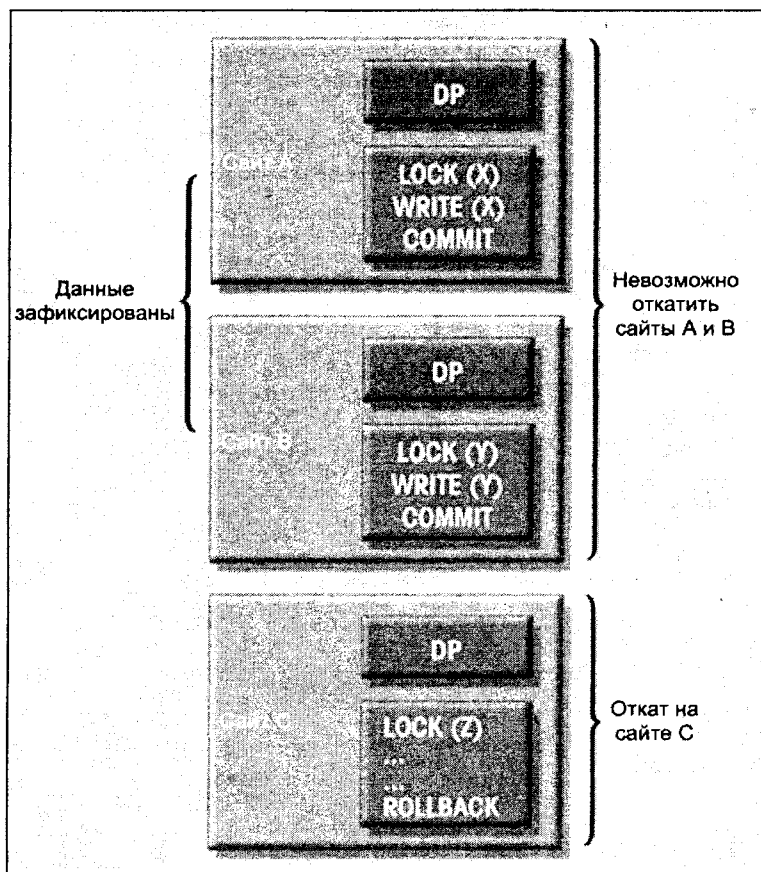


Рис. 10.15. Результат преждевременного завершения транзакции

10.8.3. Протокол двухфазного подтверждения транзакции

Централизованной базе данных необходим только один процессор данных (DP). Все операции с базой данных проводятся на одном сайте и последовательность операций сразу становится известна СУБД. И наоборот, распределенные базы данных позволяют транзакциям осуществлять доступ к данным на нескольких сайтах. Завершающий оператор COMMIT не должен выполняться до тех пор, пока каждый сайт не завершит свою часть транзакции. Протокол двухфазного подтверждения транзакции требует, чтобы каждая запись в журнале транзакций процессора данных выполнялась до фактического обновления фрагмента (см. разд. 9.6). Поэтому протокол двухфазного подтверждения транзакции требует применения протокола DO-UNDO-REDO (выполнить-отменить-повторить) и протокола упреждающей записи.

Протокол *DO-UNDO-REDO* используется процессором данных для отката транзакций назад (roll back) и/или отката транзакций вперед (roll forward) на основе записей в системном журнале транзакций. Протокол *DO-UNDO-REDO* устанавливает три типа операций:

1. *DO* выполняет операцию и записывает в журнал транзакций значения "перед" и "после".
2. *UNDO* отменяет операцию с помощью записей в журнале транзакций, сделанных операцией *DO*.
3. *REDO* вновь выполняет отмененную операцию с помощью записей в журнале, сделанных операцией *DO*.

Чтобы гарантировать, что операции *DO*, *UNDO* и *REDO* смогут обеспечить корректное выполнение операций при крахе системы, используется протокол упреждающей записи. Протокол *упреждающей записи (write-ahead protocol)* принуждает фиксировать в журнале запись данных для постоянного хранения перед фактическим выполнением этой операции.

Протокол двухфазного подтверждения транзакции определяет операции между двумя типами узлов: *узел-координатор (coordinator)* и один или более подчиненных *узлов-субординаторов (subordinates)*, или *когорту (cohort)*. Протокол реализуется в две фазы.

Фаза 1. Подготовка

1. Координатор посылает сообщение *PREPARE TO COMMIT* (подготовка к завершению) всем субординаторам.
2. Субординаторы получают сообщение, записывают информацию в журнал транзакций в соответствии с протоколом упреждающей записи и посылают координатору уведомление *YES/PREPARED TO COMMIT* (да/завершение подготовлено) или *NO/NOT PREPARED* (нет/завершение не готово).
3. Координатор убеждается, что все узлы готовы к завершению, или в противном случае отменяет действие.

Если все узлы сообщили, что они готовы к завершению (*PREPARED TO COMMIT*), транзакция переходит в фазу 2. Если один или более узлов отвечают, что они не готовы (*NO* или *NOT PREPARED*), координатор распространяет среди всех субординаторов сообщение *ABORT* (прекращение).

Фаза 2. Последний оператор COMMIT

1. Координатор оповещает всех субординаторов, рассылая сообщение *COMMIT*, и ожидает ответ.
2. Каждый субординатор, получив сообщение *COMMIT*, обновляет базу данных в соответствии с протоколом *DO*.
3. Субординаторы отвечают координатору сообщением *COMMITTED* (завершено) или *NOT COMMITTED* (не завершено).

Если один или более субординаторов не выполнили операцию завершения, координатор рассылает сообщение ABORT и тем самым инициирует операцию UNDO (отмену всех изменений).

Цель протокола двухфазного подтверждения транзакции состоит в обеспечении корректного завершения всеми узлами своих частей транзакции; в противном случае транзакция отменяется. Если один или более узлов не выполняют операцию завершения, то необходимая информация по восстановлению БД будет находиться в журнале транзакций и база данных может быть восстановлена с помощью протокола DO-UNDO-REDO (помните, что информация журнала обновляется на основе протокола упреждающей записи).

10.9. Прозрачность производительности и оптимизация запроса

Одна из основных функций базы данных состоит в обеспечении доступа к данным. Поскольку в централизованной БД все данные размещаются на одном сайте, СУБД должна оценивать каждый запрос к данным и находить наиболее эффективный способ получения доступа к локальным данным. В отличие от этого СУРБД позволяет разбивать базу данных на несколько фрагментов, в связи с чем транслировать запросы становится сложнее, поскольку СУРБД должна решать, к какому из фрагментов БД необходимо обеспечить доступ. Кроме того, данные могут дублироваться на нескольких сайтах. Дублирование (репликация) данных еще более усложняет проблему доступа к данным, поскольку в этом случае СУРБД должна решать, к какой из копий данных необходимо обеспечить доступ. Для устранения подобных проблем и обеспечения приемлемой производительности базы данных СУРБД использует технологию оптимизации запросов.

Целью программы оптимизации запросов является минимизация общих затрат, связанных с выполнением запроса. Затраты на выполнение запроса зависят от перечисленных ниже факторов.

- ☐ Затраты на время доступа (ввод/вывод) при получении доступа к физическим данным, хранящимся на диске.
- ☐ Затраты на коммуникацию, связанную с передачей данных между узлами в системе распределенной базы данных.
- ☐ Затраты на использование центрального процессора (CPU), связанные с накладными расходами на управление распределенными транзакциями.

Хотя затраты часто разделяют на коммуникационные затраты и затраты на обработку, на самом деле их очень трудно отделить друг от друга. Не все алгоритмы оптимизации используют одинаковые параметры и не все алгоритмы присваивают одинаковые веса этим параметрам. Например, некоторые алгоритмы минимизируют общее время, другие минимизируют время коммуникации, а третьи не принимают в расчет время работы центрального процессора, считая эти затраты незначительными по сравнению с другими расходами.

Чтобы оценить оптимизацию запроса, помните, что процессор транзакций (TP) должен получить данные от процессора данных (DP), синхронизировать их, скомпоновать ответ и представить его конечному пользователю или приложению. Хотя этот процесс является стандартным, необходимо учитывать, что отдельный запрос может выполняться на одном из нескольких сайтов. Время отклика удаленных сайтов не всегда легко предопределить, поскольку некоторые узлы могут выполнить свою часть запроса быстрее других.

Одно из наиболее важных свойств оптимизации запроса в системе распределенной базы данных состоит в том, что она должна обеспечить прозрачность распределения, а также прозрачность *реплики* (точной копии). Прозрачность распределения мы пояснили ранее в этой главе. *Прозрачность реплики (replica transparency)* связана с возможностью СУРБД скрывать существование нескольких копий данных от пользователя.

Большинство алгоритмов оптимизации запросов основаны на двух принципах:

- выбор оптимального порядка выполнения;
- выбор сайтов, к которым необходимо получить доступ для минимизации стоимости коммуникации.

На основе этих двух принципов алгоритм оптимизации можно оценить по режиму его работы или расчетному времени оптимизации.

Режим работы может быть ручным и автоматическим. Автоматическая оптимизация запроса означает, что СУРБД сама находит наиболее эффективный путь доступа без вмешательства пользователя. Ручная оптимизация запроса предполагает, что оптимизация и исполнение определяются конечным пользователем или программистом. Автоматическая оптимизация запроса, очевидно, более предпочтительна с точки зрения пользователя, однако при этом возрастают непроизводительные издержки СУРБД.

Алгоритмы оптимизации запроса можно классифицировать по времени выполнения оптимизации. По этому показателю алгоритмы подразделяются на *статические* и *динамические*.

- *Статическая оптимизация запроса* выполняется во время компиляции. Другими словами, наилучшая стратегия оптимизации избирается во время компиляции запроса СУБД. Такой подход применяется в том случае, если SQL-операторы встроены в процедурный язык программирования, например, COBOL или Pascal. Когда программа передается СУБД для компиляции, создается план доступа к базе данных. При выполнении программы СУБД использует этот план для доступа к базе данных.
- *Динамическая оптимизация запроса* производится на этапе его выполнения. Стратегия доступа к базе данных определяется при выполнении программы. Поэтому стратегия доступа динамически определяется СУБД на основе самой последней информации о базе данных. Хотя динамическая оптимизация запроса является более эффективной, затраты на нее определяются стоимостью выполнения всех ее процессов. Наилучшая стратегия определяется всякий раз при выполнении запроса, а это может происходить несколько раз в одной программе.

Наконец, технологии оптимизации запросов подразделяются в зависимости от типа информации, используемой для оптимизации запроса. Например, оптимизация запросов может быть основана на статистических алгоритмах или на алгоритмах, использующих определенные правила.

- ❑ *Статистические алгоритмы оптимизации запроса.* В этом случае используется статистическая информация о базе данных, представляющая собой такие характеристики БД, как ее размер, число записей, среднее время доступа, число обработанных запросов, число пользователей с правами доступа и т. д. Эта статистическая информация затем используется СУБД для определения наилучшей стратегии.
- ❑ Статистическая информация контролируется СУБД и создается в двух разных режимах: динамическом и ручном. В *динамическом режиме создания статистики* СУРБД автоматически оценивает и обновляет статистику после каждого доступа к данным. В *ручном режиме создания статистики* ее необходимо обновлять периодически при помощи специальных утилит, таких, например, как RUNSTAT, компании IBM, которая используется, например, в OS/2 Database Manager.
- ❑ *Алгоритм оптимизации запроса на основе правил* основан на наборе определенных пользователем правил, определяющих наилучшую стратегию запросов. Эти правила устанавливаются конечными пользователями или администратором базы данных и, как правило, носят очень общий характер.

10.10. Проектирование распределенной базы данных

Является база данных распределенной или нет, принципы проектирования и основные концепции, описанные в гл. 2, остаются теми же. Однако при проектировании распределенной БД возникают три новые проблемы.

- ❑ Как разбивать базу данных на фрагменты?
- ❑ Какие фрагменты необходимо дублировать (реплицировать или тиражировать)?
- ❑ Где расположить эти фрагменты и реплики?

Фрагментация данных и репликация данных относятся к первым двум указанным проблемам, а *размещение данных* — к третьей.

10.11. Фрагментация данных

Фрагментация данных допускает разбиение одного объекта на два или более сегмента или фрагмента. Объект может представлять собой пользовательскую базу данных, системную базу данных или таблицу. Каждый фрагмент может храниться на любом сайте компьютерной сети. Информация о фрагментации данных хранится в каталоге распределенных данных (distributed data catalog, DDC), к которому процессор транзакций (TP) может получить доступ при обработке запросов пользователя.

Обсуждающиеся здесь стратегии фрагментации данных действуют на уровне таблиц и заключаются в разбиении таблицы на логические фрагменты. Мы исследуем три типа стратегий фрагментации данных: *горизонтальная*, *вертикальная* и *смешанная*. (Отметим, что фрагментированную таблицу в любой момент можно снова объединить посредством комбинации операций объединения (union) и соединения (join).)

- *Горизонтальная фрагментация.* При этом таблица (отношение, relation) подразделяется на подмножества (фрагменты) кортежей (строк). Каждый фрагмент хранится на отдельном узле (сайте) и каждый фрагмент имеет уникальные строки. Однако все уникальные строки имеют одинаковые атрибуты (столбцы). Иначе говоря, каждый фрагмент эквивалентен оператору SELECT с модифицирующим выражением WHERE по единственному атрибуту.
- *Вертикальная фрагментация.* Такой тип фрагментации подразумевает разделение отношения (таблицы) на подмножества атрибутов (столбцов). Каждое подмножество (фрагмент) хранится на отдельном узле и каждый фрагмент имеет уникальные столбцы — за исключением ключевого столбца, который имеется во всех фрагментах. Это эквивалентно применению оператора PROJECT.
- *Смешанная фрагментация.* Эта фрагментация представляет собой комбинацию вертикальной и горизонтальной стратегий. Другими словами, таблица может разбиваться на несколько горизонтальных множеств (строк), каждая из которых разбивается на множество атрибутов (столбцов).

Для иллюстрации стратегий фрагментации мы используем таблицу CUSTOMER (клиент) компании XYZ, представленную на рис. 10.16. В таблице есть атрибуты CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE, CUS_LIMIT, CUS_BAL, CUS_RATING и CUS_DUE.

	CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
▶	10	Sinex, Inc.	12 Main St.	TN	\$3,500.00	\$2,700.00	3	\$1,245.00
	11	Martin Corp.	321 Sunset Blvd.	FL	\$6,000.00	\$1,200.00	1	\$0.00
	12	Mynux Corp.	910 Eagle St.	TN	\$4,000.00	\$3,500.00	3	\$3,400.00
	13	BTBC, Inc.	Rue du Monde	FL	\$6,000.00	\$5,890.00	3	\$1,090.00
	14	Victory, Inc.	123 Maple St.	FL	\$1,200.00	\$550.00	1	\$0.00
	15	NBCC Corp.	909 High Ave.	GA	\$2,900.00	\$350.00	2	\$50.00

Рис. 10.16. Пример таблицы CUSTOMER

10.11.1. Горизонтальная фрагментация

Предположим, что руководству компании XYZ необходима информация о клиентах по всем трем штатам, но каждому подразделению компании (TN — Теннесси, FL — Флорида и GA — Джорджия) необходима информация только по своим локальным клиентам. На основе этого было принято решение распределить данные по штатам. Поэтому для реализации структуры, представленной в табл. 10.3, выбрали горизонтальную фрагментацию.

Таблица 10.3. Горизонтальная фрагментация таблицы CUSTOMER по штатам

Фрагмент	Местоположение	Условие	Узел	Номера клиентов	Количество строк
CUST_H1	Теннесси	CUS_STATE = 'TN'	NAS	10, 12	2
CUST_H2	Джорджия	CUS_STATE = 'GA'	ATL	15	1
CUST_H3	Флорида	CUS_STATE = 'FL'	TAM	11, 13, 14	3

Каждый горизонтальный фрагмент может иметь свое число строк, но ДОЛЖЕН иметь те же атрибуты, что и остальные фрагменты. После фрагментации мы получим три таблицы, представленные на рис. 10.17.

Фрагмент: CUST_H1		Местонахождение: Теннесси				Узел NAS		
	CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
▶	10	Sinex, Inc.	12 Main St.	TN	\$3,500.00	\$2,700.00	3	\$1,245.00
		12 Mynux Corp.	910 Eagle St.	TN	\$4,000.00	\$3,500.00	3	\$3,400.00
Фрагмент: CUST_H2		Местонахождение: Джорджия				Узел ATL		
	CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
▶	15	NBCC Corp.	909 High Ave.	GA	\$2,000.00	\$350.00	2	\$50.00
Фрагмент: CUST_H3		Местонахождение: Флорида				Узел TAM		
	CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
▶	11	Martin Corp.	321 Sunset Blvd	FL	\$6,000.00	\$1,200.00	1	\$0.00
	13	BTBC, Inc.	Rue du Monde	FL	\$6,000.00	\$5,890.00	3	\$1,090.00
	14	Victory, Inc.	123 Maple St.	FL	\$1,200.00	\$550.00	1	\$0.00

Рис. 10.17. Фрагменты таблицы по местоположению

10.11.2. Вертикальная фрагментация

Мы можем разбить таблицу CUSTOMER на вертикальные фрагменты, представляющие собой наборы атрибутов. Например, предположим, что у компании есть два подразделения: отдел обслуживания и отдел приема платежей. Каждое подразделение расположено в отдельном здании и каждому подразделению необходима информация только по нескольким атрибутам таблицы CUSTOMER. Разделение по фрагментам для такого варианта представлено в табл. 10.4.

Таблица 10.4. Вертикальная фрагментация таблицы CUSTOMER

Фрагмент	Местоположение	Узел	Атрибуты
CUST_V1	Помещение обслуживания	SVC	CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE

Таблица 10.4 (окончание)

Фрагмент	Местоположение	Узел	Атрибуты
CUST_V2	Помещение приема платежей	ARC	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE

Каждый вертикальный фрагмент имеет одинаковое количество строк, но включает в себя различные атрибуты, зависящие от ключевого столбца. Результаты вертикальной фрагментации представлены на рис. 10.18. Обратите внимание, что ключевой атрибут (CUS_NUM) является общим и для фрагмента CUST_V1, и для фрагмента CUST_V2.

Фрагмент: CUST_V1		Местонахождение: Обслуживание (здание)			Узел SVC
	CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE	
▶	10	Sinex, Inc.	12 Main St.	TN	
	11	Martin Corp.	321 Sunset Blvd.	FL	
	12	Mynux Corp.	910 Eagle St.	TN	
	13	BTBC, Inc.	Rue du Monde	FL	
	14	Victory, Inc.	123 Maple St.	FL	
	15	NBCC Corp.	909 High Ave.	GA	

Фрагмент: CUST_V1		Местонахождение: Прием платежей (здание)			Узел ARC
	CUS_NUM	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE
▶	10	\$3,500.00	\$2,700.00	3	\$1,245.00
	11	\$6,000.00	\$1,200.00	1	\$0.00
	12	\$4,000.00	\$3,500.00	3	\$3,400.00
	13	\$6,000.00	\$5,890.00	3	\$1,090.00
	14	\$1,200.00	\$550.00	1	\$0.00
	15	\$2,000.00	\$350.00	2	\$50.00

Рис. 10.18. Содержание таблиц при вертикальной фрагментации

10.11.3. Смешанная фрагментация

Структура компании XYZ требует, чтобы данные таблицы CUSTOMER были фрагментированы горизонтально, отражая деление компании на местные филиалы, и в то же время в рамках филиала данные необходимо фрагментировать вертикально, чтобы выделить подразделения (обслуживание и платежи). Другими словами, таблицу CUSTOMER необходимо фрагментировать по смешанной стратегии.

Смешанная фрагментация представляет собой двухступенчатую процедуру. Сначала мы проводим горизонтальную фрагментацию на каждом сайте на основе разбиения компании по филиалам (CUS_STATE). Горизонтальная фрагментация представляет

набор кортежей клиентов (горизонтальные фрагменты), расположенных на каждом сайте. Поскольку подразделения расположены в разных помещениях, мы используем вертикальную фрагментацию в рамках каждого горизонтального фрагмента для разбиения атрибутов, что обеспечит каждое подразделение необходимой ему информацией. Смешанная фрагментация приводит к результатам, представленным в табл. 10.5.

Таблица 10.5. Смешанная фрагментация таблицы CUSTOMER

Фрагмент	Местоположение	Условие горизонтальной фрагментации	Узел	Номера строк на сайте	Атрибуты для вертикальной фрагментации на каждом фрагменте
CUST_M1	Обслуживание в Теннесси	CUS_STATE = 'TN'	NAS-S	10, 12	CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE
CUST_M2	Прием платежей в Теннесси	CUS_STATE = 'TN'	NAS-C	10, 12	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE
CUST_M3	Обслуживание в Джорджии	CUS_STATE = 'GA'	ATL-S	15	CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE
CUST_M4	Прием платежей в Джорджии	CUS_STATE = 'GA'	ATL-C	15	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE
CUST_M5	Обслуживание во Флориде	CUS_STATE = 'FL'	TAM-S	11, 13, 14	CUS_NUM, CUS_NAME, CUS_ADDRESS, CUS_STATE
CUST_M6	Прием платежей во Флориде	CUS_STATE = 'FL'	TAM-C	11, 13, 14	CUS_NUM, CUS_LIMIT, CUS_BAL, CUS_RATING, CUS_DUE

Каждый фрагмент, представленный в табл. 10.5, содержит данные о клиентах по штату и, внутри каждого штата, — по локальным подразделениям, чтобы удовлетворить требования подразделений. Таблицы, соответствующие фрагментам, перечисленным в табл. 10.5, представлены на рис. 10.19.

Фрагмент: CUST_M1		Местонахождение: TN-Обслуживание		Узел NAS-S	
CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE		
10	Sinex, Inc.	12 Main St.	TN		
12	Myrnux Corp.	910 Eagle St.	TN		
Фрагмент: CUST_M2		Местонахождение: TN-Прием платежей		Узел NAS-C	
CUS_NUM	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE	
10	\$3,500.00	\$2,700.00	3	\$1,245.00	
12	\$4,000.00	\$3,500.00	3	\$3,400.00	
Фрагмент: CUST_M3		Местонахождение: GA-Обслуживание		Узел ATL-S	
CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE		
13	NBCC Corp.	909 High Ave.	GA		
Фрагмент: CUST_M4		Местонахождение: GA-Прием платежей		Узел ATL-C	
CUS_NUM	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE	
13	\$2,000.00	\$350.00	2	\$50.00	
Фрагмент: CUST_M5		Местонахождение: FL-Обслуживание		Узел TAM-S	
CUS_NUM	CUS_NAME	CUS_ADDRESS	CUS_STATE		
11	Martin Corp.	321 Sunset Blvd.	FL		
13	BTBC, Inc.	Rue du Monde	FL		
14	Victory, Inc.	123 Maple St.	FL		
Фрагмент: CUST_M6		Местонахождение: FL-Прием платежей		Узел TAM-C	
CUS_NUM	CUS_LIMIT	CUS_BAL	CUS_RATING	CUS_DUE	
11	\$6,000.00	\$1,200.00	1	\$0.00	
13	\$6,000.00	\$5,890.00	3	\$1,090.00	
14	\$1,200.00	\$550.00	1	\$0.00	

Рис. 10.19. Содержимое таблиц после смешанной фрагментации

10.12. Репликация данных

Репликация данных связана с хранением копий данных в сети компьютеров на нескольких сайтах, предназначенных для специальных операций с данными. Поскольку копии фрагментов повышают уровень доступности данных и уменьшают время отклика, репликация поможет уменьшить общие затраты на коммуникации при выполнении запросов.

Предположим, что база данных А разделена на два фрагмента А1 и А2. Внутри реплицированной и распределенной базы данных возможен сценарий, представленный на рис. 10.20: фрагмент А1 хранится на сайтах S1 и S2, в то время как фрагмент А2 хранится на сайтах S2 и S3.

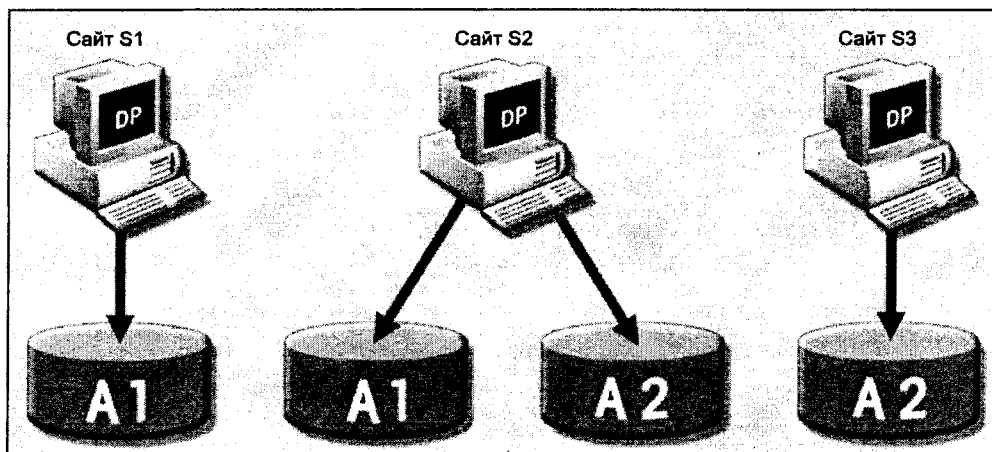


Рис. 10.20. Репликация данных

Реплицированные данные подчиняются правилу взаимной непротиворечивости. *Правило взаимной непротиворечивости (mutual consistency rule)* требует, чтобы все копии фрагментов данных были идентичны. Поэтому для обеспечения непротиворечивости данных в репликах СУРБД должна гарантировать, что обновление БД выполняется на всех сайтах, где есть реплики данных.

Хотя репликация обладает некоторыми преимуществами, она влечет за собой и некоторые дополнительные накладные расходы, поскольку система должна обслуживать каждую копию данных. Для иллюстрации появления таких накладных расходов рассмотрим процессы, которые СУРБД должна выполнить при работе с базой данных.

- ❑ Если БД фрагментирована, то СУРБД для получения доступа к соответствующему фрагменту должна разбивать запрос на подзапросы.
- ❑ Если БД реплицирована, то СУРБД должна принимать решение, к какой копии необходимо обеспечить доступ. Операция чтения (Read) выбирает *ближайшую* копию, которая годится для данной транзакции. Операция записи (Write) в соответствии с правилом взаимной непротиворечивости должна выбирать и обновлять *все* копии данных.
- ❑ Процессор транзакций (TP) посылает данные на обработку на каждый выбранный процессор данных (DP).
- ❑ DP получает данные, обрабатывает каждый запрос и отправляет данные обратно на TP.
- ❑ TP объединяет все ответы DP.

Проблема еще больше усложнится, если мы примем во внимание такие дополнительные факторы, как топологию сети и ее пропускную способность.

Возможны три варианта репликации БД: база данных может быть *полностью реплицированной*, *частично реплицированной* и *нереплицированной*.

- ❑ *Полностью реплицированная база данных* хранит на множестве сайтов несколько копий *каждого* фрагмента БД. В этом случае все фрагменты БД реплицированы.

Полностью реплицированная БД может оказаться неудобной в использовании из-за больших накладных расходов.

- *Частично реплицированная база данных* хранит на множестве сайтов несколько копий *некоторых* фрагментов БД. Большинство СУРБД допускают работу именно с частично реплицированными БД.
- *Нереплицированная база данных* хранит каждый фрагмент БД на отдельном сайте. В этом случае дублированные фрагменты БД отсутствуют.

На репликацию БД влияет несколько факторов:

- размер БД;
- частота использования БД;
- затраты (эффективность, непроизводительные издержки программного обеспечения и управления), связанные с синхронизацией транзакций и их частей при обеспечении должной отказоустойчивости, связанной с репликацией данных.

Если частота обращения к удаленным данным очень высока, а база данных большая, то репликация данных может уменьшить затраты на обработку запросов. Информация о репликации данных хранится в каталоге распределения данных (DDC), содержимое которого TP использует, чтобы принять решение — к какой копии фрагмента БД необходимо обеспечить доступ. Репликация данных позволяет восстановить утерянные данные.

10.13. Распределение данных

Распределение данных (data allocation) представляет собой процесс принятия решения о месте хранения данных. Выделяют следующие типы стратегии распределения данных:

- *централизованное распределение данных* — вся база данных хранится на одном сайте;
- *секционированное распределение данных*, при котором база данных разбивается на несколько разъединенных частей (фрагментов) и хранится на нескольких сайтах;
- *реплицированное распределение данных* предполагает хранение одной или более копий фрагментов БД на нескольких сайтах.

Распределение данных по компьютерной сети достигается с помощью сегментирования данных, репликации данных или с помощью комбинации этих методов. Распределение данных тесно связано с методами фрагментации БД. По большей части распределение данных связано с решением одной проблемы: какие данные необходимо разместить и где.

Алгоритмы распределения данных должны учитывать различные факторы, а именно:

- производительность и доступность данных;
- размер, число строк и число отношений, которые данная сущность имеет с другими сущностями;
- типы транзакций, применяемые в БД, атрибуты, доступ к которым осуществляется в этих транзакциях, и т. д.

Некоторые алгоритмы используют внешние данные, например, топологию сети или пропускную способность сети. Оптимального или универсального алгоритма пока нет, хотя в настоящее время их реализовано очень много.

10.14. Сравнение СУРБД и архитектуры "клиент/сервер"

Поскольку в настоящее время установилась тенденция к развитию распределенных баз данных, многие поставщики БД применяют обозначение *клиент/сервер* для указания возможности распределения данных. Однако это обозначение не всегда точно отражает характеристики архитектуры "клиент/сервер".

Архитектура "клиент/сервер" связана со способом взаимодействия компьютеров, формирующих систему. Основными составляющими элементами архитектуры "клиент/сервер" являются *пользователь* ресурсов, или клиент, и *поставщик* ресурсов, или сервер. Архитектуру "клиент/сервер" можно использовать при реализации СУБД, в которой клиентом является процессор транзакций (ТР), а сервером — процессор данных (ДР).

Взаимодействие клиента и сервера в СУРБД тщательно планируется. Клиент (ТР) взаимодействует с конечным пользователем и посылает запрос на сервер (ДР). Сервер получает, планирует и выполняет запрос, *выбирая только те записи, которые необходимы клиенту*. Затем сервер посылает информацию клиенту, но только когда клиент их затребует.

Приложения "клиент/сервер" обладают рядом достоинств:

- ☐ решения "клиент/сервер" дешевле, чем альтернативные микрокомпьютерные решения или системы на базе мэйнфреймов;
- ☐ решения "клиент/сервер" предоставляют конечному пользователю графический интерфейс микрокомпьютеров, что увеличивает функциональные возможности и упрощает работу;
- ☐ опытных пользователей персонального компьютера гораздо больше, чем тех, кто имеет опыт работы с мэйнфреймом;
- ☐ с помощью персонального компьютера легко организовать рабочее место;
- ☐ на рынке персональных компьютеров много доступных средств анализа данных и создания запросов, предназначенных для облегчения взаимодействия с множеством СУБД;
- ☐ при переносе разработки приложений с мэйнфреймов на мощные персональные компьютеры есть значительный выигрыш в стоимости.

Клиент/серверные приложения, к сожалению, имеют и целый ряд недостатков:

- ☐ архитектура "клиент/сервер" порождает более сложную инфраструктуру, в которой зачастую трудно управлять различными платформами (локальными сетями, операционными системами и т. д.);
- ☐ увеличение числа пользователей и обрабатываемых сайтов часто вызывает проблемы безопасности;

□ инфраструктура "клиент/сервер" позволяет обеспечить доступ к данным гораздо большему числу пользователей. Это требует от пользователя достаточно полного знания компьютеров и программного обеспечения. Необходимость обучения увеличивает стоимость обслуживания такой инфраструктуры.

Более подробно с концепциями, компонентами и управлением в архитектуре "клиент/сервер" мы познаком вас в гл. 12.

10.15. Двенадцать правил Дейта для распределенных баз данных

Ни одно исследование распределенных баз данных не может претендовать на полноту без цитирования двенадцати правил К. Дейта (C. J. Date) для распределенных баз данных². Правила Дейта описывают полностью распределенные базы данных, и хотя ни одна из современных баз данных не соответствует *всем* этим правилам, они, тем не менее, играют очень важную роль при разработке распределенных БД. Вот эти правила.

1. *Независимость локального сайта.* Каждый локальный сайт может действовать как независимая, автономная централизованная СУБД. Каждый сайт отвечает за безопасность, управление параллельным выполнением, резервное копирование и восстановление данных.
2. *Независимость от центрального сайта.* Ни один сайт в сети не зависит от центрального или какого-либо другого сайта. Все сайты имеют равные возможности.
3. *Независимость от сбоев.* Функционирование системы не зависит от сбоя на каком-либо узле. Система продолжает выполнение операций даже при неисправности узла или при расширении сети.
4. *Прозрачность местоположения.* Пользователь не обязан знать местоположение данных, чтобы осуществлять их поиск.
5. *Прозрачность фрагментации.* Пользователь видит единую логическую базу данных. Фрагментация базы данных прозрачна для пользователя. Пользователю нет необходимости знать имена фрагментов БД для получения доступа к ним.
6. *Прозрачность репликации.* Пользователь видит единую логическую базу данных. СУРБД предоставляет доступ к фрагментам данных прозрачно (невидимо) для пользователя. СУРБД управляет всеми фрагментами так, что пользователь этого не замечает.
7. *Распределенная обработка запросов.* Распределенная обработка запросов может выполняться на нескольких сайтах процессоров данных (DP). Оптимизацию запросов СУРБД выполняет опять-таки прозрачно для пользователя.
8. *Распределенная обработка транзакций.* Транзакции могут обновлять данные на нескольких различных сайтах. Выполнение транзакции происходит прозрачно для пользователя на нескольких сайтах DP.

² См. Date, C. J. "Twelve Rules for Distributed Database", *Computer Word*, 8 июня 1987, 2(23), стр. 77—81

9. *Независимость от оборудования.* Система должна выполняться на любой аппаратной платформе.
10. *Независимость от операционной системы.* Система должна выполняться в любой операционной системе.
11. *Независимость от сети.* Система должна работать на любой сетевой платформе.
12. *Независимость от базы данных.* Система должна поддерживать любую базу данных.

Резюме

В распределенных базах данных логически связанные данные хранятся на двух или более физически независимых сайтах, связанных компьютерной сетью. База данных разделяется на фрагменты, причем разбиение может быть горизонтальным (наборы строк) или вертикальным (наборы атрибутов). Каждый фрагмент может размещаться на различных узлах сети (сайтах).

Распределенная обработка состоит в разделении логической обработки базы данных между двумя или более узлами сети. Распределенные базы данных требуют распределенной обработки. Система управления распределенной базой данных (СУРБД) управляет обработкой и хранением логически связанных данных во взаимосвязанных компьютерных системах.

К основным компонентам СУРБД относятся процессор транзакций (ТР) и процессор данных (ДР). Процессор транзакций представляет собой программный компонент, размещенный на каждом компьютерном узле сети, который запрашивает информацию. Процессор данных представляет собой программный компонент, расположенный на каждом компьютере, где хранятся данные и осуществляется их поиск.

Современные системы баз данных классифицируются по уровню распределения данных и процессов обработки. При этом выделяются три основные категории систем базы данных: (1) обработка и размещение данных на одном сайте, (2) обработка данных на нескольких сайтах и размещение их на одном сайте, (3) обработка и размещение данных на нескольких сайтах.

Гомогенные системы распределенных баз данных объединяют в сети только один тип СУБД. Гетерогенные системы распределенных баз данных объединяют в сети несколько различных типов СУБД.

СУРБД обладают набором прозрачных для пользователя свойств: распределения, транзакций, сбоев, гетерогенности и производительности. Все прозрачные свойства преследуют одинаковую цель: чтобы работа с распределенной базой данных выглядела так же, как с централизованной БД, т. е. конечный пользователь видит данные как составную часть единой логической базы данных и не должен задумываться о сложности реализации такой системы.

Прозрачность распределения позволяет управлять физически рассеянной базой данных так, как будто бы она была централизованной БД. Существуют три уровня прозрачности распределения: фрагментации, местоположения и локального отображения.

Прозрачность транзакций гарантирует, что выполнение распределенных транзакций обеспечит целостность и непротиворечивость распределенной БД.

Прозрачность сбоев гарантирует непрерывность функционирования СУРБД, если на одном или более узлов происходит сбой.

Прозрачность производительности гарантирует, что система будет работать как централизованная СУБД, и что общая производительность системы не зависит от распределения данных по нескольким удаленным сайтам.

Прозрачность гетерогенности гарантирует, что СУБД будет объединять несколько различных типов локальных СУБД в единую общую, или глобальную, схему базы данных.

Транзакция состоит из одного или более запросов к БД. Нераспределенная транзакция обновляет или запрашивает данные на единственном сайте. Распределенная транзакция может обновлять и запрашивать данные с нескольких сайтов.

В сети распределенных баз данных необходимо управление параллельным выполнением. Для обеспечения завершения всех фрагментов транзакции используется протокол двухфазного подтверждения транзакции.

Распределенная СУБД (СУРБД) оценивает каждый запрос с тем, чтобы определить оптимальный путь доступа в распределенной базе данных. СУРБД должна оптимизировать запрос с целью уменьшения затрат на доступ, коммуникации и загрузку центрального процессора (CPU), связанных с выполнением запроса.

При проектировании распределенных БД необходимо принимать в расчет фрагментацию и репликацию данных. Проектировщик должен решать, каким образом необходимо разместить каждый фрагмент или реплику с тем, чтобы уменьшить время отклика и гарантировать доступность данных для конечного пользователя.

База данных может быть реплицирована (тиражирована) по нескольким сайтам в компьютерной сети. Репликация фрагментов БД ставит своей целью повышение доступности данных и уменьшение времени доступа к ним. БД может быть полностью реплицированной, частично реплицированной или нереплицированной. Для определения местоположения фрагментов БД или реплик разрабатываются стратегии распределения данных.

Поставщики баз данных часто обозначают свое программное обеспечение как клиент/серверный продукт. Обозначение "архитектура «клиент/сервер»" связано со способом, которым два компьютера взаимодействуют через компьютерную сеть, формируя систему. Клиент/серверные системы приобрели широкую популярность в последние несколько лет в основном благодаря увеличению мощности мини-компьютеров, росту локальных сетей и сравнительно невысокой стоимости таких систем по сравнению с подобными решениями на базе мэйнфреймов.

Основные термины

Автоматическая оптимизация запросов — automatic query optimization

Алгоритм оптимизации запроса, основанный на правилах — rule-based query optimization algorithm

Алгоритм оптимизации, основанный на статистике — statistically based query optimization algorithm

Архитектура "клиент/сервер" — client/server architecture

Динамическая оптимизация запроса — dynamic query optimization

Каталог распределенных данных — distributed data catalog (DDC)

Координатор — coordinator

Менеджер данных — data manager (DM)

Менеджер транзакций — transaction manager (TM)

Нереплицированная база данных — unreplicated database

Полностью реплицированная база данных — fully replicated database

Правило взаимной непротиворечивости — mutual consistency rule

Прозрачность локального отображения — local mapping transparency

Прозрачность местоположения — location transparency

Прозрачность реплики — replica transparency

Прозрачность транзакций — transaction transparency

Прозрачность фрагментации — fragmentation transparency

Прозрачные свойства СУРБД

- прозрачность гетерогенности — heterogeneity transparency
- прозрачность производительности — performance transparency
- прозрачность распределения — distribution transparency
- прозрачность сбоев — failure transparency
- прозрачность транзакций — transaction transparency

Протокол DO-UNDO-REDO — DO-UNDO-REDO protocol

Протокол двухфазного подтверждения транзакции — two-phase commit protocol

Протокол упреждающей записи — write-ahead protocol

Процессор данных — data processor (DP)

Процессор приложения — application processor (AP)

Процессор транзакций — transaction processor (TP)

Распределение данных

- реплицированное — centralized data allocation
- секционированное — partitioned data allocation
- централизованное — centralized data allocation

Распределенная база данных — distributed database

Распределенная обработка — distributed processing

Распределенная транзакция — distributed transaction

Распределенный запрос — distributed request

Режим динамической генерации статистики — dynamic statistical generation mode

Режим ручной генерации статистики — manual statistical generation mode

Репликация данных — data replication

Ручная оптимизация запроса — manual query optimization

Система гетерогенной распределенной базы данных — heterogeneous distributed database system

Система гомогенной распределенной базы данных — homogeneous distributed database system

Система управления полностью распределенной гетерогенной базой данных — fully heterogeneous distributed database management system

Система управления распределенной базой данных (СУРБД) — distributed database management system (DDBMS)

Словарь распределенных данных — distributed data dictionary (DDD)

Статическая оптимизация запроса — static query optimization

Субординатор — subordinator

Схема глобального распределения — distributed global schema

Удаленная транзакция — remote transaction

Удаленный запрос — remote request

Уникальность фрагмента — unique fragment

Фрагментация данных

- вертикальная — vertical data fragmentation
- горизонтальная — horizontal data fragmentation
- смешанная — mixed data fragmentation

Фрагменты базы данных — database fragments

Частично реплицированная база данных — partially replicated database

Вопросы

1. Опишите развитие систем управления баз данных от СУБД к СУРБД.
2. Перечислите и опишите некоторые факторы, повлиявшие на эволюцию СУРБД.
3. В чем состоят преимущества СУРБД?
4. Перечислите недостатки СУРБД?
5. Поясните различие между распределенной базой данных и распределенной обработкой данных.
6. Что такое система управления полностью распределенной базой данных?
7. Перечислите основные компоненты СУРБД.
8. Расскажите о прозрачных свойствах СУРБД.
9. Опишите и поясните различные типы прозрачности распределения.
10. Опишите различные типы запросов к базе данных и транзакций БД.
11. Объясните необходимость протокола двухфазного подтверждения транзакции. Опишите обе фазы.

12. Какова цель функций оптимизации запроса?
13. С каким прозрачным свойством связана оптимизация запроса?
14. Перечислите типы алгоритмов оптимизации запроса.
15. Опишите три стратегии фрагментации данных. Приведите примеры.
16. Что такое репликация данных, и каковы три стратегии репликации?
17. Поясните различие между файл-серверной и клиент/серверной архитектурой.

Задачи

1. Следующая задача основана на СУРБД, представленной на рис. 10.21.

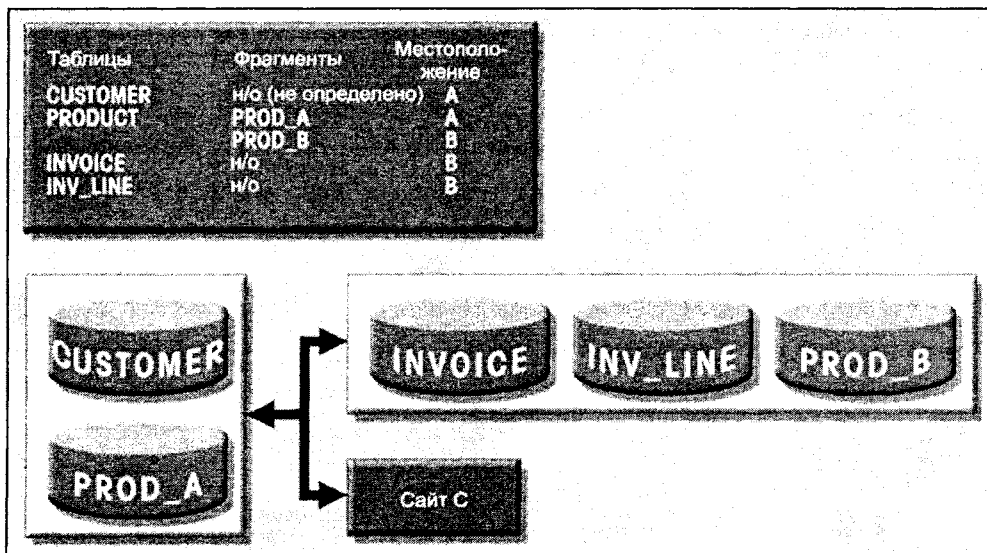


Рис. 10.21. СУРБД для задачи 1

Определите минимум типов операций, которые должна поддерживать база данных (удаленный запрос, удаленная транзакция, распределенная транзакция или распределенный запрос) для того, чтобы выполнить следующие операции:

На Сайте С

- `SELECT *`
`FROM CUSTOMER;`
- `SELECT *`
`FROM INVOICE`
`WHERE INV_TOT > 1000;`

- ```
SELECT *
FROM PRODUCT
WHERE PROD_QOH < 10;
```
- ```
BEGIN WORK;  
UPDATE CUSTOMER  
SET CUS_BAL = CUS_BAL + 100  
WHERE CUS_NUM = '10936';  
INSERT INTO INVOICE(INV_NUM,CUS_NUM,INV_DATE,INV_TOTAL)  
VALUES ('986391', '10936', '15-FEB-2002', 100);  
INSERT INTO LINE(INV_NUM, PROD_NUM, LINE_PRICE)  
VALUES ('986391', '1023', 100);  
UPDATE PRODUCT  
SET PROD_QOH = PROD_QOH -1  
WHERE PROD_NUM = '1023';  
COMMIT WORK;
```
- ```
BEGIN WORK;
INSERT CUSTOMER(CUS_NUM,CUS_NAME,CUS_ADDRESS, CUS_BAL)
VALUES ('34210', 'Victor Ephanor', '123 Main St.', 0.00);
INSERT INTO INVOICE(INV_NUM,CUS_NUM,INV_DATE,INV_TOTAL)
VALUES ('986434', '34210', '10-AUG-1999', 2.00);
COMMIT WORK;
```

### На Сайте А

- ```
SELECT CUS_NUM,CUS_NAME,INV_TOTAL  
FROM CUSTOMER,INVOICE  
WHERE CUSTOMER.CUS_NUM = INVOICE.CUS_NUM;
```
- ```
SELECT *
FROM INVOICE
WHERE INV_TOTAL > 1000;
```
- ```
SELECT *  
FROM PRODUCT  
WHERE PROD_QOH < 10;
```

На сайте В

- ```
SELECT *
FROM CUSTOMER;
```

- ```
SELECT CUS_NAME, INV_TOTAL
FROM CUSTOMER, INVOICE
WHERE INV_TOTAL > 1000 AND
CUSTOMER.CUS_NUM = INVOICE.CUS_NUM;
```
- ```
SELECT *
FROM PRODUCT
WHERE PROD_QOH < 10;
```

2. Следующая структура данных и ограничений создана для издательства журнала.

- Издательство публикует один региональный журнал во Флориде (FL), Южной Каролине (SC), Джорджии (GA) и Теннесси (TN).
- Издательство имеет 300 000 клиентов (подписчиков), распределенных по четырем штатам, перечисленным выше.
- Первого числа каждого месяца печатается ежегодная подписка INVOICE и рассылается клиентам, которым необходимо возобновить подписку. Сущность INVOICE содержит атрибут REGION, указывающий штат (FL, SC, GA TN), в котором находится клиент:

CUSTOMER (CUS\_NUM, CUS\_NAME, CUS\_ADDRESS, CUS\_CITY, CUS\_ZIP,  
CUS\_SUBSDATE)

INVOICE (INV\_NUM, INV\_REGION, CUS\_NUM, INV\_DATE, INV\_TOTAL)

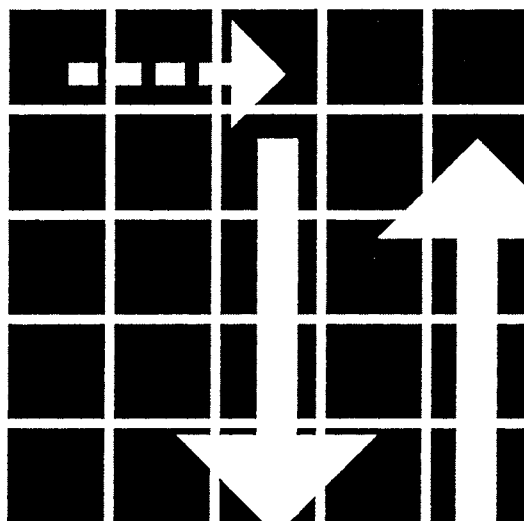
Руководство компании, узнав о проблемах, связанных с централизованным управлением, приняло решение о децентрализации управления подпиской на издание по четырем региональным отделениям. Каждый сайт подписки будет обрабатывать данные своих клиентов и их счета для создания годового отчета и ответа на нерегламентированные запросы типа:

- составить список всех клиентов по регионам;
- составить список всех новых клиентов по регионам;
- составить отчет по всем счетам клиентов по регионам.

Как исходя из этих требований нужно разбить базу данных?

3. Исходя из сценария и требований задачи 2, дайте ответ на следующие вопросы.

- Какие рекомендации вы можете дать по типам и свойствам системы базы данных?
- Какой тип фрагментации данных необходим для каждой таблицы?
- Какой критерий должен использоваться при разбиении каждой базы данных?
- Спроектируйте фрагменты базы данных. Представьте пример с именами узлов, фрагментов, атрибутов и данными.
- Какой тип операций с распределенной БД должен поддерживаться на каждом удаленном сайте?
- Какой тип операций должен поддерживаться на основном сайте (в штаб-квартире компании)?



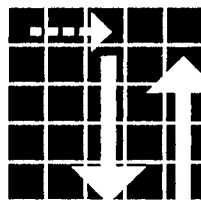
**ЧАСТЬ V**

**НОВЫЕ  
РАЗРАБОТКИ**





## Глава 11



# Объектно-ориентированные базы данных

В этой главе мы будем изучать:

- основные концепции объектно-ориентированных систем (ОО-систем);
- влияние объектно-ориентированных концепций (ОО-концепций) на моделирование данных и проектирование БД;
- использование объектно-ориентированных свойств в реляционных схемах и ER-моделях;
- основные свойства системы управления объектно-ориентированной базой данных (ОСУБД);
- достоинства и недостатки ОСУБД.

## Обзор

Управление данными и разработка приложений стали гораздо сложнее, чем могли себе это представить создатели первых иерархических, сетевых и реляционных СУБД. Современные базы данных могут обрабатывать сложные данные как на уровне управления данными, так и на уровне разработки приложений (последняя возможность появилась сравнительно недавно). В конце 1980-х — начале 1990-х годов требования к информации настолько усложнились, что данные стало очень трудно обрабатывать с помощью существующих технологий баз данных. Изменившийся состав данных, которые необходимо было моделировать (графика, видео, аудио, картография, диаграммы, отпечатки пальцев и голос, а также сложная цифровая и текстовая информация) потребовали реорганизации существующих систем баз данных. Все это стало причиной возникновения новых технологий баз данных, основанных на объектно-ориентированных концепциях программирования, и появления у реляционных баз данных новых функциональных возможностей, позволяющих обеспечить обработку сложных данных.

Привлекательность объектно-ориентированных технологий основана на мощных технологиях программирования и моделирования и расширенных возможностях обработки данных. Поскольку вклад ОО-технологий в развитие технологии обработки данных очень высок, необходимо изучить основные свойства ОО-систем и их влияние на моделирование и проектирование данных.

Мы также рассмотрим основные свойства нового поколения реляционных баз данных, отвечающих требованиям обработки сложных типов данных. Такие базы данных называют расширенными реляционными, или объектно-реляционными базами данных. Мы обсудим общие свойства ОО-систем, не углубляясь в изучение какой-то отдельной реализации ООСУБД.

## 11.1. Преимущества объектно-ориентированного подхода

Объектно-ориентированная методология моделирования и разработки основана на объектно-ориентированных концепциях. Точнее, ориентированность на объекты предполагает использование принципов проектирования и разработки, основанных на автономных компьютерных структурах, которые называются *объектами*. Каждый объект представляет некую сущность реального мира, открытую для взаимодействия и обладающую возможностью связи с другими объектами.

Из этого определения следует, что при использовании объектов модульность практически неизбежна. Объектно-ориентированные концепции широко применяются во многих компьютерных сферах, особенно в сложном программировании и проектировании. В табл. 11.1 представлены сведения о месте объектно-ориентированного подхода в некоторых компьютерных дисциплинах.

**Таблица 11.1.** Объектно-ориентированный подход в компьютерном мире

| Область компьютерных дисциплин           | Результат объектно-ориентированного подхода                                                                                                                                                                                       |
|------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Языки программирования                   | Уменьшение количества строк кода                                                                                                                                                                                                  |
|                                          | Сокращение времени разработки                                                                                                                                                                                                     |
|                                          | Возможность многократного использования кода                                                                                                                                                                                      |
|                                          | Упрощение обслуживания кода                                                                                                                                                                                                       |
|                                          | Увеличение производительности труда программистов                                                                                                                                                                                 |
| Графический интерфейс пользователя (GUI) | Расширенные возможности по созданию удобного интерфейса                                                                                                                                                                           |
|                                          | Дружелюбность конечному пользователю                                                                                                                                                                                              |
|                                          | Упрощение разработки стандартов                                                                                                                                                                                                   |
| Базы данных                              | Поддержка абстрактных типов данных                                                                                                                                                                                                |
|                                          | Поддержка сложных объектов                                                                                                                                                                                                        |
|                                          | Поддержка мультимедийных типов данных                                                                                                                                                                                             |
|                                          | Поддержка манипуляций со сложными типами данных в специальных приложениях для работы с графикой, изображениями, картами, экономическими моделями, телекоммуникациями, глобальными приложениями, медицинскими приложениями и т. д. |

Таблица 11.1 (окончание)

| Область компьютерных дисциплин | Результат объектно-ориентированного подхода                                                                                                                                                                                     |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Проектирование                 | Улучшенная поддержка семантических моделей<br>Более точное отображение реального мира                                                                                                                                           |
| Операционные системы           | Расширение возможностей переноса системы путем создания слоев абстракции для обработки проблем, связанных с оборудованием, и облегчение расширения системы с помощью наследования и других объектно-ориентированных конструкций |

## 11.2. История развития объектно-ориентированного подхода

Идеи объектно-ориентированного подхода берут свое начало в объектно-ориентированном программировании (ООП), предложенном в качестве альтернативы традиционным методам программирования. В среде ООП программист создает или использует объекты — автономные, многократно используемые модули, содержащие данные, а также процедуры, работающие с этими данными.

Понятие объектов впервые возникло в таких языках программирования, как Ada, Algol, LISP и SIMULA. Эти языки программирования были промежуточной ступенью до появления языков с более выраженными ОО-концепциями. Наиболее популярными объектно-ориентированными языками программирования (object-oriented programming language, OOPL) можно считать Smalltalk, C++ и Java. Язык Java используется при создании Web-приложений, выполняемых в среде Интернет и не зависящих от конкретной операционной системы.

Разработка объектно-ориентированных языков программирования преследовала следующие цели:

- ☐ обеспечить удобную программную среду разработки;
- ☐ предоставить программистам мощные инструментальные средства программного моделирования для разработки приложений;
- ☐ сократить время разработки за счет уменьшения общего количества строк кода;
- ☐ увеличить производительность труда программистов, обеспечив возможность многократного использования кода.

Объектно-ориентированное программирование изменило не только способ написания программ, но и само поведение этих программ. При объектно-ориентированном представлении мира каждый объект может манипулировать данными, которые являются частью этого объекта. Кроме того, каждый объект может посылать сообщения для изменения данных в других объектах.

Следовательно, объектно-ориентированная инфраструктура обладает следующими свойствами:

- набор данных больше не является пассивным;
- данные и процедуры, будучи связанными друг с другом, образуют объект;
- объект может воздействовать на самого себя.

Объект может взаимодействовать с другими объектами, тем самым образуя систему объектов. Поскольку такие объекты содержат в себе данные и код, можно достаточно просто создать модульную систему с возможностью ее многократного использования. Именно это свойство ОО-системы кажется совершенно естественным тем пользователям, у кого нет большого опыта программирования, но сбивает с толку опытных программистов, привыкших разделять данные и процедуры. Не удивительно, что идеи ОО получили быстрое развитие именно с появлением персональных компьютеров (ПК), поскольку на ПК обычно работают обычные пользователи, а не программисты и специалисты по системным работам.

Идеи ООП оказали влияние на большинство компьютерных приложений, включая базы данных. Поскольку база данных проектируется с целью сбора данных о бизнес-системе, ее можно рассматривать как набор взаимодействующих объектов. Каждый объект обладает определенными свойствами (атрибутами) и способами взаимодействия с другими объектами (методы). Обладая такой структурой, ОО-системы становятся весьма привлекательными для применения их в проектировании и разработке баз данных. Несмотря на то что данная книга предназначена в большей степени для проектировщиков баз данных, а не для программистов, мы, тем не менее, уделим объектно-ориентированному подходу должное внимание.

## 11.3. Общие принципы объектно-ориентированного подхода

Хотя концепции ОО уходят корнями в языки программирования, и программисты, читающие эту книгу, сразу распознают основные элементы программирования, для уяснения основных положений ОО нет необходимости обладать какими-то познаниями в сфере программирования.

### 11.3.1. Объекты: компоненты и свойства

В ОО-системах все, с чем мы имеем дело, является объектом; будь это студент, счет-фактура, самолет, сотрудник, служба, меню, отчет и т. д. Некоторые объекты материальны, некоторые нет. Мы можем определить *объект* как абстрактное представление сущности реального мира, имеющее уникальный идентификатор, встроенные свойства и возможность взаимодействовать с другими объектами, а также воздействовать на самого себя.

Обратите внимание на отличие *объекта* от *сущности*. У сущности есть компоненты данных и связи, но сущность не обладает возможностью манипулирования данными. Позже мы рассмотрим некоторые другие различия между сущностью и объектом.

Важнейшей характеристикой объекта является его *уникальная идентификация*. Чтобы особо подчеркнуть этот факт, рассмотрим объект реального мира, представленный на рис. 11.1. Обратите внимание, что студент (объект) с именем J. D. Wilson имеет уникальную (биологическую) идентификацию и этим отличается от объектов M. R. Gonzalez или V. K. Spelling. Также отметим, что хотя объекты имеют общие основные свойства, например, имя, номер социального страхования, адрес и дату рождения, тем не менее, каждый объект существует во времени и пространстве независимо от других объектов.

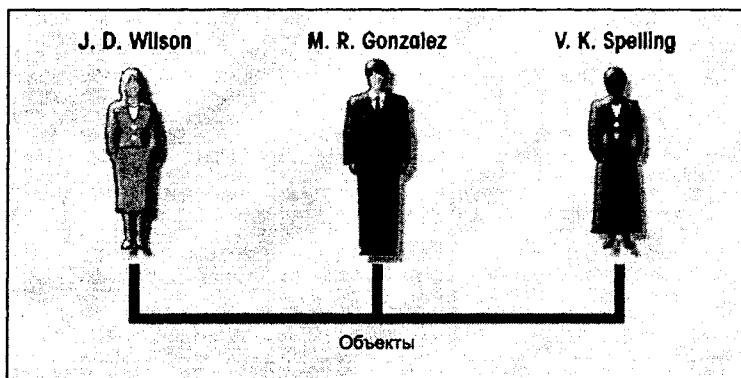


Рис. 11.1. Студенты как объекты реального мира

### 11.3.2. Идентификация объекта

Идентификация объекта выражается *идентификатором объекта (object ID, OID)*, уникальным для данного объекта. Идентификатор (OID) назначается системой в момент создания объекта и не может быть изменен ни при каких условиях.

Не надо путать первичный ключ реляционной модели и идентификатор объекта (OID). В отличие от OID, основу первичного ключа составляют значения, определенные пользователем для данного атрибута, и он может быть изменен в любое время. Идентификатор OID назначается системой, не зависит от значений атрибута и его нельзя изменить. OID можно удалить только в том случае, если удаляется сам объект, и данный OID никогда не может быть использован повторно.

### 11.3.3. Атрибуты (переменные экземпляра)

Объекты описываются их *атрибутами*, которые в объектно-ориентированной среде называются *переменными экземпляра (instance variables)*. Например, у студента John D. Smith могут быть атрибуты (переменные экземпляра), представленные в табл. 11.2. Каждый атрибут имеет уникальное имя и связанный с ним тип данных. В табл. 11.2 имена атрибутов это SOCIAL\_SECURITY\_NUMBER (номер социального страхования), FIRST\_NAME (имя), MIDDLE\_INITIAL (отчество или инициал), LAST\_NAME (фамилия) и т. д. Традиционные типы данных, также называемые

базовыми типами данных (*base data types*) или договорные типы данных (*conventional data types*), используются в большинстве языков программирования и включают в себя типы *real* (вещественный), *integer* (целый), *string* (строковый) и т. д.

Для атрибутов определены домены. *Домен* это логическая группа, представляющая собой набор возможных значений данного атрибута. Например, возможные значения атрибута *SEMESTER\_GPA* (средняя оценка за семестр, см. табл. 11.2) могут быть представлены числом типа *real* из набора базовых типов. Но это не означает, что для *GPA* (средняя оценка) допустимо любое вещественное число. Нужно иметь в виду, что типы данных определяют базовые домены, т. е. тип *real* (вещественный) представляет собой все вещественные числа, тип *integer* — все целые числа, тип *date* — все возможные даты, тип *string* — любые комбинации символов и т. д. Однако домены базовых типов данных представляют собой лишь основу, используемую для создания более ограниченных именованных доменов на более высоком логическом уровне. Например, для точного определения домена атрибута *GPA* мы должны создать домен с именем "GPA". Каждый домен имеет имя и описание, включая базовый тип данных, размер, формат и ограничения на значения домена. Поэтому мы можем определить домен "GPA" как "любое положительное число между 0,00 и 4,00 с двумя знаками после запятой". В данном случае у нас имеется домен с именем "GPA", базовым типом данных *real*, ограничивающим правилом "любое положительное число между 0,00 и 4,00" и форматом "два знака после запятой". Домен "GPA" предоставляет значения для атрибутов *SEMESTER\_GPA* и *OVERALL\_GPA*. Домены могут также определяться как список возможных значений, разделенных запятыми. Например, домен "GENDER" (пол) может быть определен как "Male, Female" (мужской, женский) или "M,F".

**Таблица 11.2. Атрибуты объекта**

| Имя атрибута           | Значение атрибута                                                                                                                                     |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| SOCIAL_SECURITY_NUMBER | 414-48-0944                                                                                                                                           |
| FIRST_NAME             | John                                                                                                                                                  |
| MIDDLE_NAME            | D                                                                                                                                                     |
| LAST_NAME              | Smith                                                                                                                                                 |
| DATE_OF_BIRTH          | 11/23/1966                                                                                                                                            |
| MAJOR*                 | Accounting                                                                                                                                            |
| SEMESTER_GPA           | 2,89                                                                                                                                                  |
| OVERALL_GPA            | 3,01                                                                                                                                                  |
| COURSES_TAKEN          | ENG201;MATH243;HIST201;ACCT211;ECON210;EC<br>ON212;ACCT212;CIS220;ENG202;MATH301;HIST20<br>2;CIS310;ACCT343;ACCT345;ENG242;MKTG301;FI<br>N331;ACCT355 |
| ADVISOR                | Dr. W. R. Learned                                                                                                                                     |

\* Атрибут, который ссылается на один или более других объектов.

Очень важно отметить, что реляционная модель базы данных также поддерживает домены. На самом деле, К. Дейт, один из "отцов" реляционной модели, полагает, что домены помогут обеспечить в реляционных базах данных поддержку абстрактных типов данных, дающую те же функциональные возможности, что и объектно-ориентированные базы данных<sup>1</sup>.

Так же как и в ER-модели, атрибут объекта может быть *однозначным* (иметь одно значение) или *многозначным* (иметь несколько значений). Следовательно, атрибут объекта может получать из своего домена одно или несколько значений. Например, атрибут SOCIAL\_SECURITY\_NUMBER получает из домена только одно значение, поскольку данный студент имеет только один номер социального страхования. А такие атрибуты, как Language (язык) или Hobby (увлечение), могут иметь несколько значений, поскольку студент может говорить на нескольких языках и иметь много увлечений.

Атрибуты объекта могут ссылаться на один или более других объектов. Например, атрибут MAJOR (специализация) связан с объектом Department (факультет), атрибут COURSER\_TAKEN (группы обучения) связан со списком (или набором) объектов Course (группы), а атрибут ADVISOR (куратор) связан с объектом Professor (преподаватель). На уровне реализации для ссылки на объект используется его OID, что позволяет реализовать связи между двумя и более объектами. В табл. 11.2 атрибут MAJOR содержит OID объекта Department (English), а атрибут ADVISOR содержит OID объекта Professor (Dr. W. R. Learned). Атрибут COURSES\_TAKEN содержит OID объекта, который содержит список объектов Course. Объект, содержащий в себе список других объектов, называется *объект-набор* (*collection object*).

### Примечание

Обратите внимание на разницу между реляционной и объектно-ориентированной моделями. В реляционной модели атрибут таблицы может содержать единственное значение, которое можно использовать для соединения (JOIN) строк из различных таблиц. В ОО-модели нет необходимости в операторе JOIN для связи объектов друг с другом.

## 11.3.4. Состояние объекта

*Состояние объекта* (*object state*) представляет собой набор значений атрибутов объекта в данный момент времени. При изменении состояния объекта его OID остается неизменным. Если нужно изменить состояние объекта, то мы должны изменить значения атрибутов объекта. Для этого надо послать объекту *сообщение*, которое будет инициировать *метод*.

## 11.3.5. Сообщения и методы

*Метод* (*method*) представляет собой код, выполняющий определенные операции над данными, принадлежащими объекту. Методы защищают данные от прямого и неавторизованного доступа из других объектов. Чтобы уяснить роль сообщений и мето-

<sup>1</sup> См. C. J. Date and Hugh Darwen, Foundation for Object/Relational Databases: The Third Manifesto, Addison-Wesley, 1998.

дов, представьте объект в виде ореха. Ядро ореха можно сравнить со структурой данных объекта, а скорлупу — с его методами (рис. 11.2).

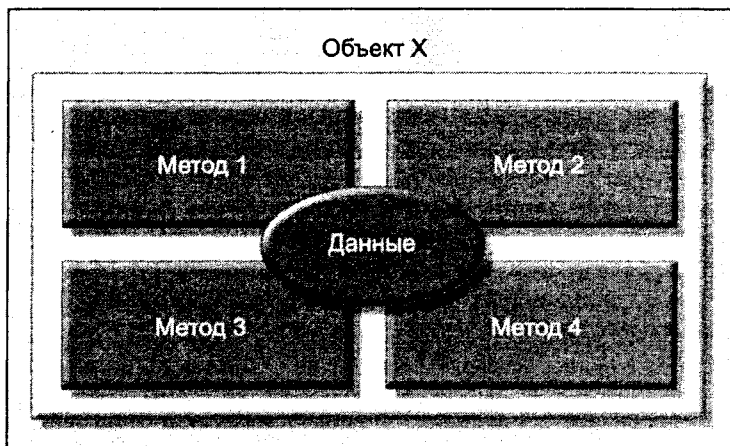


Рис. 11.2. Представление объекта

Все операции, выполняемые объектом, реализуются с помощью методов. Методы используются для изменения значений атрибутов объекта или для получения значений указанных атрибутов. Методы представляют собой реальные действия, такие, например, как изменение специализации студента, добавление студента в группу или распечатка имени студента и его адреса. В сущности *методы эквивалентны процедурам в традиционных языках программирования*. В терминах ОО методы определяют поведение объекта.

Для каждого метода определяются *имя* (name) и *тело* (body). Тело состоит из компьютерных инструкций на некотором языке программирования, описывающих некоторое реальное действие. Например, используя атрибуты объекта, приведенные в табл. 11.2, мы можем определить метод Avegra, возвращающий среднюю (average) оценку (GPA) студента на основе атрибутов SEMESTR\_GPA и OVERALL\_GPA объекта. Метод с именем Avegra выполняет следующие преобразования (рис. 11.2, а).

Если студент имеет в текущем семестре оценку GPA = 3,2 с нагрузкой 15 часов в семестр, а предыдущая оценка GPA студента была 3,0 при нагрузке 60 часов, то возвращаемое методом Avegra значение будет таким:

$$(3,2 \times 15 + 3,0 \times 60) / (15 + 60) = (48 + 180) / 75 = 3,04$$

В этом примере обратите внимание, что метод может получать доступ к переменным экземпляра (атрибутам) объекта, для которого этот метод определен.

Для инициализации метода объекту посылается сообщение. При отправлении *сообщения* (message) задаются объект-адресат, имя метода и все необходимые параметры. Отправитель сообщения (другой объект) не может обратиться напрямую к внутренней структуре объекта. Запрет доступа к внутренней структуре объекта гарантирует целостность состояния объекта и скрывает детали внутренней реализации объекта.



Скрытие внутренних деталей объекта (атрибутов и методов) называется *инкапсуляцией* (*encapsulation*).

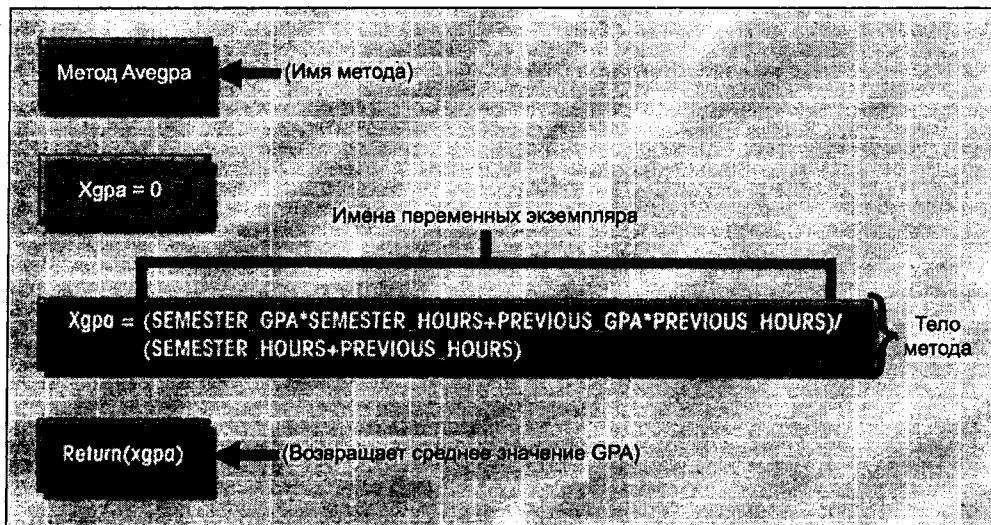


Рис. 11.2, а. Действие метода Avegpa

Объект может также посылать сообщения для изменения состояния или опроса текущего состояния объекта. (Опрос состояния — *interrogate* — означает выяснение у целевого объекта текущих значений экземпляров объекта.) Для выполнения этих задач тело метода может содержать ссылки на методы других объектов (отсылка сообщений другим объектам), как это показано на рис. 11.3.

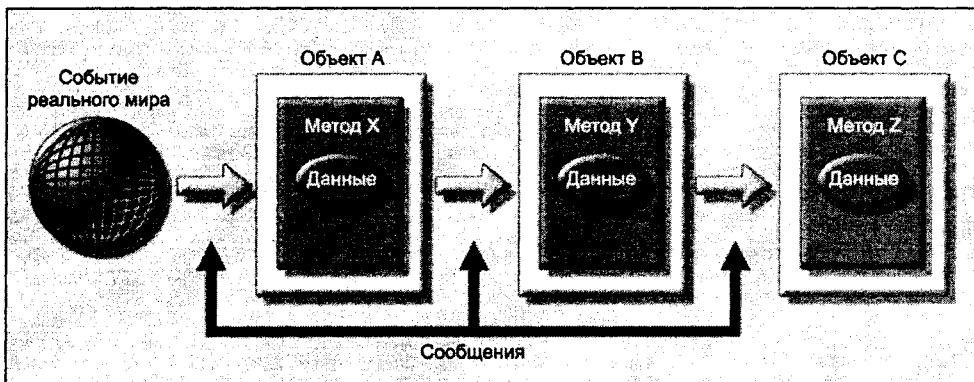


Рис. 11.3. Обмен сообщениями между объектами

### 11.3.6. Классы

В ОО-системах объекты классифицируются в соответствии с их схожестью и различием. Объекты, имеющие общие свойства, группируются в классы. Другими словами, *класс* представляет собой набор подобных объектов с разделяемыми структурой (атрибутами) и поведением (методами).

Класс содержит подробное описание структуры данных и реализации методов для объектов данного класса. Поэтому все объекты в классе используют одинаковую структуру и отвечают на одинаковые сообщения. Кроме того, класс действует как "буфер памяти" для схожих объектов. Каждый объект в классе представляет собой *экземпляр класса* (*class instance*) или *экземпляр объекта* (*object instance*) (рис. 11.4).

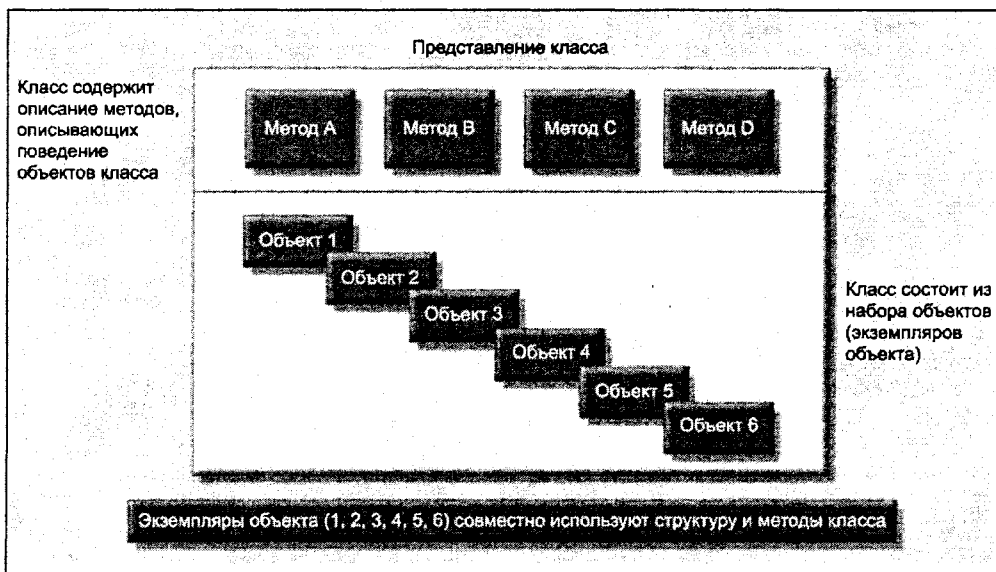


Рис. 11.4. Пример класса

Используя пример, представленный ранее в табл. 11.2, мы можем определить класс с именем *Student* для хранения объектов-студентов. Все объекты класса *Student*, представленные на рис. 11.5, используют одинаковую структуру (атрибуты) и отвечают на одинаковые сообщения (с помощью методов). Обратите внимание, что ранее мы уже определили метод *Average*, а методы *Enroll* (запись) и *Grade* (оценка), показанные на рис. 11.5, были добавлены позже. Каждый экземпляр класса представляет собой объект с уникальным *OID* и каждый объект "знает", к какому классу он принадлежит.

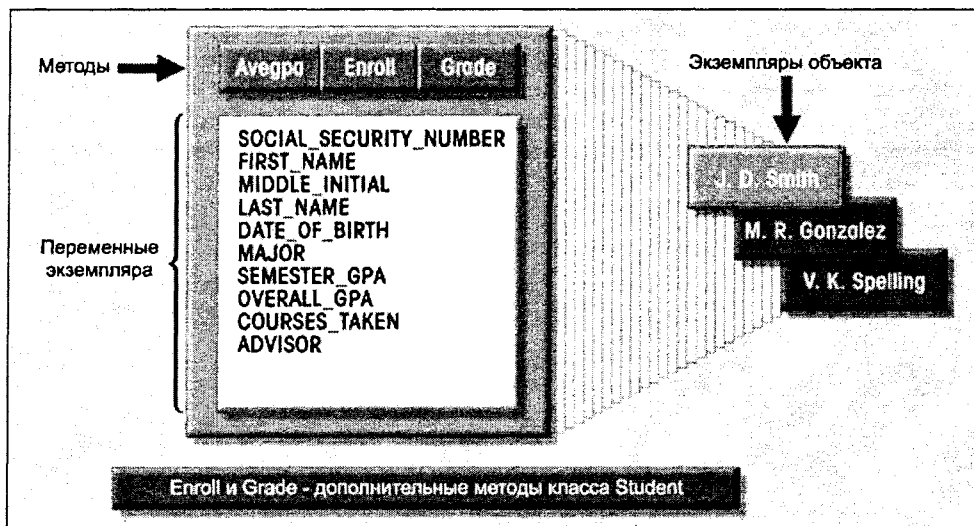


Рис. 11.5. Представление класса Student

### 11.3.7. Протокол

Набор сообщений класса, каждое с определенным именем, составляет *протокол* класса или объекта. Протокол представляет внешнюю (public) сторону объекта, т. е. он известен другим объектам, а также конечным пользователям. И наоборот, реализация структуры объекта и методов представляет внутренний (private) аспект объекта. Все эти рассуждения проиллюстрированы на рис. 11.6.

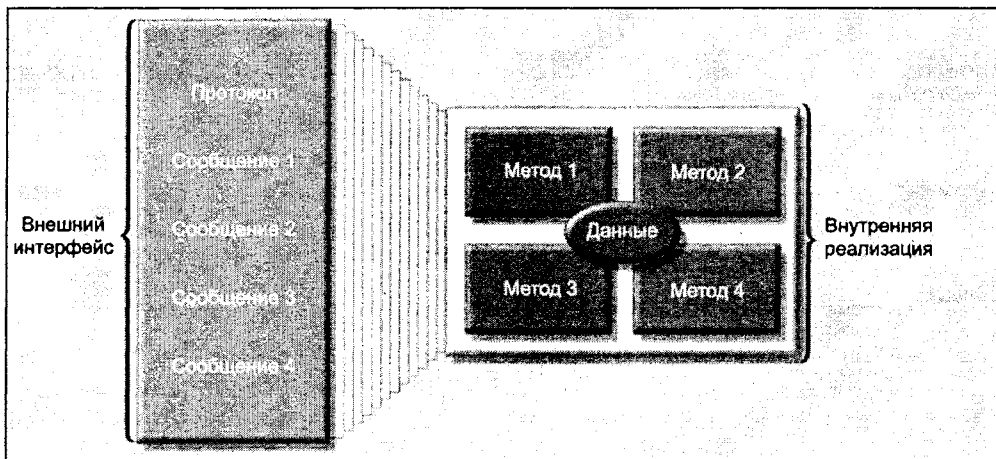


Рис. 11.6. Внешняя (public) и внутренняя (private) стороны объекта

Обычно сообщение посылается экземпляру объекта. Однако можно также послать сообщение классу, а не объекту. Когда получателем сообщения является класс, сообщение будет инициировать *метод класса*. Примером метода класса является метод new. Метод класса new создает новый экземпляр объекта (с уникальным OID) в классе-получателе. Поскольку объект еще не существует, сообщение new адресуется классу, а не объекту.

Все, о чем мы сейчас говорили, является основой понимания объектно-ориентированных концепций. На рис. 11.7 все сведения, представленные в этой главе, собраны в общую картину объектно-ориентированного подхода, поэтому вам необходимо как следует разобраться в этом рисунке, прежде чем двигаться дальше.

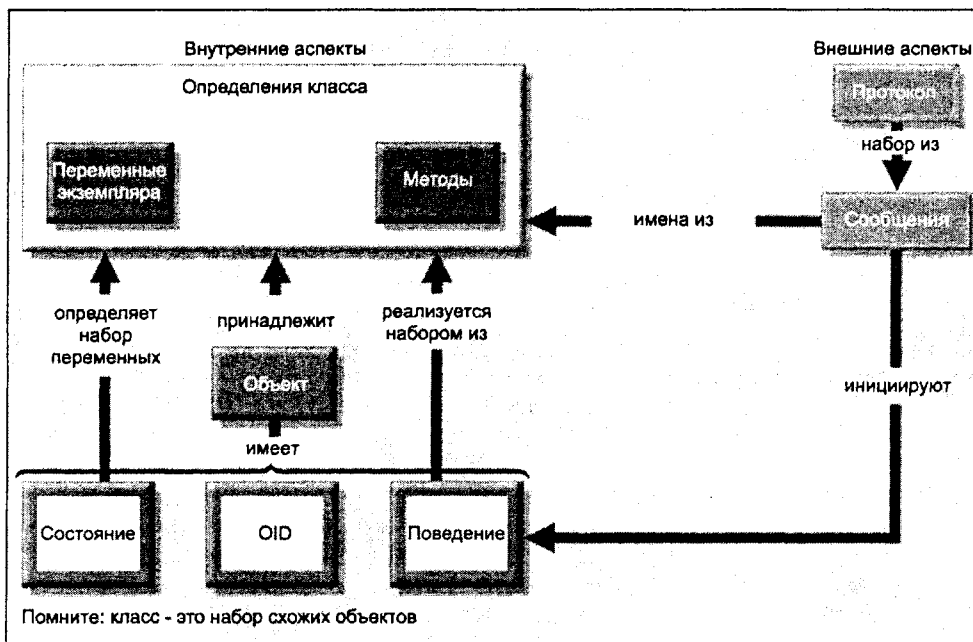


Рис. 11.7. Объектно-ориентированный подход: свойства объекта

### 11.3.8. Суперклассы, подклассы и наследование

Классы организуются в иерархию классов. *Иерархия классов* напоминает перевернутое кроной вниз дерево, в котором у каждого класса есть только один родительский класс. В случае, если классы имеют несколько родительских классов, иерархию классов называют *сеткой классов* (*class lattice*). Класс служит для распределения по группам объектов, имеющих одинаковые свойства. Например, класс "Автомобили" включает в себя большие седаны повышенной комфортности, а также небольшие

автомобили, а класс "Правительство" включает и федеральное правительство, и правительство штата, и правительство города. На рис. 11.8 показано, что класс "Музыкальные инструменты" включает в себя струнные и духовые инструменты.

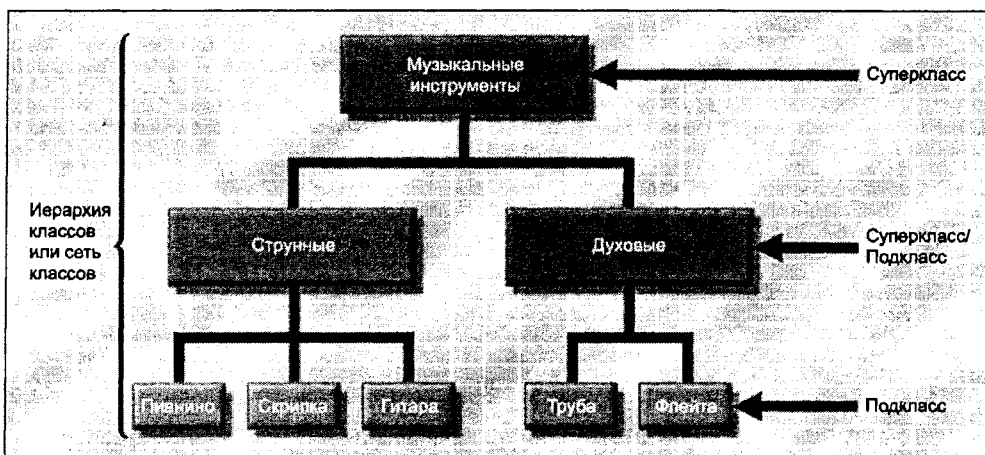


Рис. 11.8. Иерархия классов музыкальных инструментов

Обратите внимание на рис. 11.8, что классы "Пианино", "Скрипка" и "Гитара" являются *подклассами* класса "Струнные инструменты", который в свою очередь является подклассом класса "Музыкальные инструменты". Класс "Музыкальные инструменты" представляет собой *суперкласс* для класса "Струнные инструменты", который в свою очередь является суперклассом для классов "Пианино", "Скрипка" и "Гитара". Понятно, что суперкласс является более общим элементом классификации для своих подклассов, которые в свою очередь являются более специфическими компонентами общей классификации.

Иерархия классов обеспечивает мощную концепцию объектно-ориентированного подхода, которая называется наследованием. *Наследование (inheritance)* это возможность объекта внутри иерархии наследовать структуру данных и поведение (методы) классов, находящихся выше него. Например, класс "Пианино" на рис. 11.8 наследует структуру данных и поведение от суперклассов "Струнные инструменты" и "Музыкальные инструменты". Таким образом, класс "Пианино" наследует наличие струн и характер звучания от суперкласса "Струнные инструменты" и музыкальный строй от суперкласса "Музыкальные инструменты". Именно наследованием в объектно-ориентированных системах обеспечивается многократное использование кода.

В ОО-системах все объекты производятся от суперкласса Object или класса Root. Поэтому все классы совместно используют свойства и методы суперкласса Object. Наследование данных и методов происходит сверху вниз по иерархии классов. Существуют два варианта наследования: единичное (single) и множественное (multiple).

## Единичное наследование

*Единичное наследование* имеет место, когда над классом есть только один суперкласс (родительский). Такое положение на рис. 11.8 иллюстрируется классами "Струнные инструменты" и "Духовые инструменты". Большинство современных ОО-систем поддерживает единичное наследование. Когда система посылает сообщение экземпляру объекта, поиск соответствующего метода производится по всей иерархии в следующей последовательности.

1. Сканируется класс, которому принадлежит объект.
2. Если метод не найден, сканируется суперкласс.

Процесс сканирования повторяется до тех пор, пока не произойдет следующее:

1. Метод найден.
2. Достигнут верх иерархии классов, но метод не обнаружен. При этом система генерирует сообщение о том, что метод не найден.

Для иллюстрации процесса сканирования исследуем иерархию класса Employee (сотрудник), представленную на рис. 11.9. Если мы посылаем сообщение monthPay (ежемесячные выплаты) экземпляру объекта Pilot (пилот), будет выполняться метод monthPay объекта, определенный в его суперклассе Employee. Еще раз обратите внимание, что объектно-ориентированные системы обеспечивают многократное использование кода: код метода monthPay доступен и для подклассов Pilot и Mechanic.

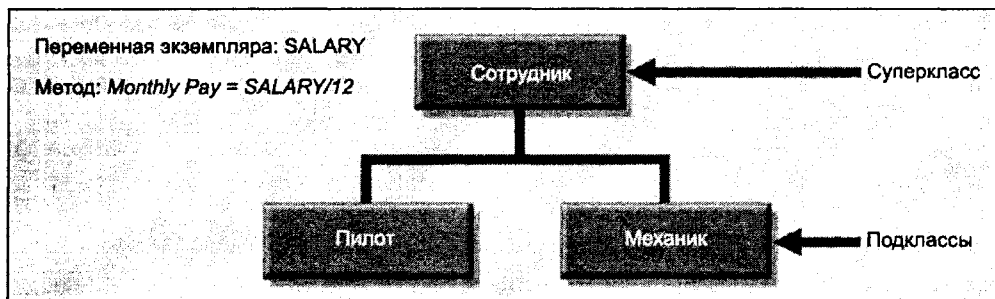


Рис. 11.9. Единичное наследование

## Множественное наследование

*Множественное наследование* имеет место, когда над классом имеется более одного родителя (суперкласса). Такое положение проиллюстрировано на рис. 11.10, где представлен пример множественного наследования (подкласс "Мотоциклы" наследует свойства как от суперкласса "Транспортное средство", так и от суперкласса "Велосипед"). От суперкласса "Транспортное средство" класс "Мотоциклы" наследует:

- ☐ свойства (топливо, поршневая группа и мощность);
- ☐ поведение (запустить двигатель, нажать педаль газа и т. д.).

От суперкласса "Велосипед" класс "Мотоцикл" наследует:

- ☐ свойства (два колеса и рулевая колонка);
- ☐ поведение (сесть верхом, повернуть рулевую колонку).

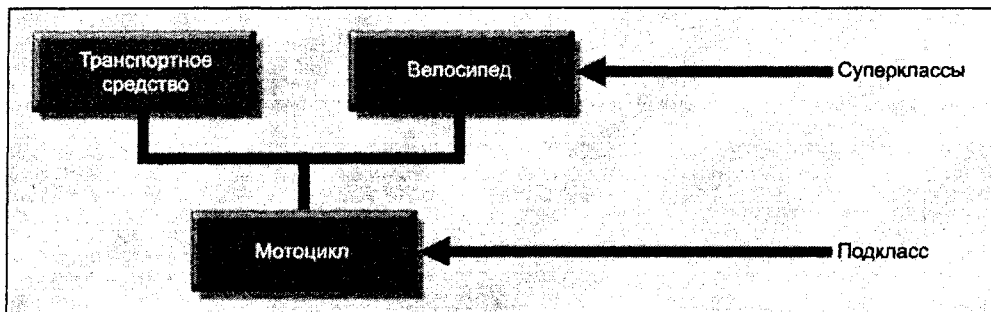


Рис. 11.10. Множественное наследование

|                                   |                       | Переменные экземпляра |                |
|-----------------------------------|-----------------------|-----------------------|----------------|
|                                   |                       | Имя                   | Значение       |
| Суперклассы<br>класса<br>Мотоцикл | Транспортное средство | MAXSPEED              | 100 миль в час |
|                                   | Велосипед             | MAXSPEED              | 35 миль в час  |

Рис. 11.11. Переменные экземпляров классов "Транспортное средство" и "Велосипед"

Назначать имя переменной или методу экземпляра при множественном наследовании иерархии класса следует с некоторой осторожностью. Например, если вы используете одинаковые имена для переменной или метода экземпляра в каждом из суперклассов, то ОО-система должна определить путь, для того чтобы решить, какой метод или атрибут необходимо использовать. Для иллюстрации предположим, что оба класса: и "Транспортное средство", и "Велосипед", представленные на рис. 11.11, используют переменную экземпляра MAXSPEED.

Какую версию переменной экземпляра MAXSPEED будет наследовать метод "Мотоцикл" в этом случае? Человек просто будет использовать для мотоцикла корректное значение 100 миль в час. Однако ОО-система не способна сделать такой выбор и поэтому может:

- ☐ вывести в специальном окне сообщение об ошибке, в котором разъясняется возникшая проблема;

- ☐ попросить пользователя ввести корректное значение или выполнить нужные действия;
- ☐ работать с некорректными результатами;
- ☐ использовать правила наследования, определенные пользователем для подкласса в сети классов. Такие правила наследования управляют наследованием методов и переменных экземпляра.

### 11.3.9. Переопределение методов и полиморфизм

Мы можем переопределить (override) метод суперкласса на уровне подкласса. Для иллюстрации переопределения метода используем иерархию класса Employee, представленную на рис. 11.12.

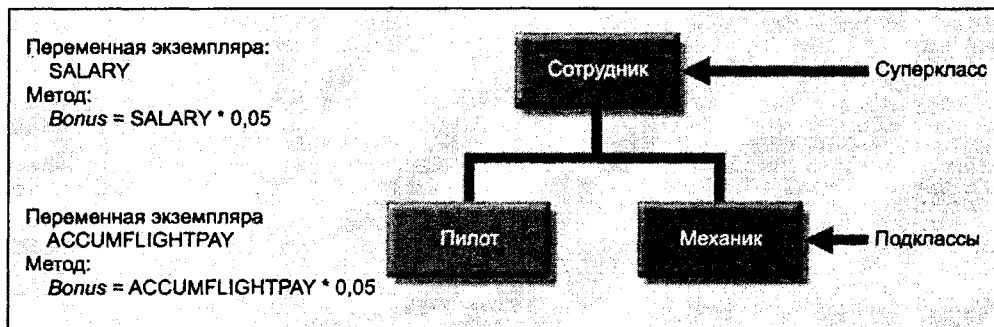


Рис. 11.12. Переопределение метода иерархии класса Employee

На рис. 11.12 следует обратить внимание, что для расчета премии к Рождеству для всех сотрудников мы определили метод Bonus (премия). Расчет премии зависит от специализации сотрудника. В данном случае каждый сотрудник, за исключением пилотов, получает премию к рождеству в размере 5% от годовой заработной платы. Пилоты получают Рождественскую премию на основе налетанных часов (ACCUMFLIGHTPAY), а не на основе годовой зарплаты. Определяя метод Bonus в подклассе Pilot (пилоты), мы переопределяем метод Bonus класса Employee для всех объектов, принадлежащих подклассу Pilot. При этом переопределение метода Bonus в подклассе Pilot не повлияет на расчет премии в подклассе Mechanic (механики). В отличие от переопределения методов, *полиморфизм* (polymorphism) позволяет различным объектам реагировать на одно и то же сообщение различными способами. Полиморфизм — очень важная функция ОО-систем, поскольку его существование позволяет объектам вести себя в соответствии с их специфическими свойствами. В терминах ОО полиморфизм означает:

- ☐ возможность использования одинаковых имен для методов, определенных в различных классах иерархии;
- ☐ пользователь может посылать одинаковые сообщения различным объектам, принадлежащим различным классам, и при этом получать корректный ответ.



Чтобы проиллюстрировать эффект полиморфизма, исследуем расширенную иерархию класса Employee, представленную на рис. 11.13. На базе этой иерархии система рассчитывает ежемесячные выплаты пилотам или механикам, посылая одно и то же сообщение monthPay объекту Pilot или Mechanic. Объект возвращает корректное значение выплаты, даже если в monthPay включается flyPay (оплата полета), свойственная объекту Pilot, и overtimePay (оплата сверхурочных), свойственная объекту Mechanic. Порядок расчета ежемесячной зарплаты для обоих подклассов (Pilot и Mechanic) один и тот же: годовой оклад делится на 12 месяцев.

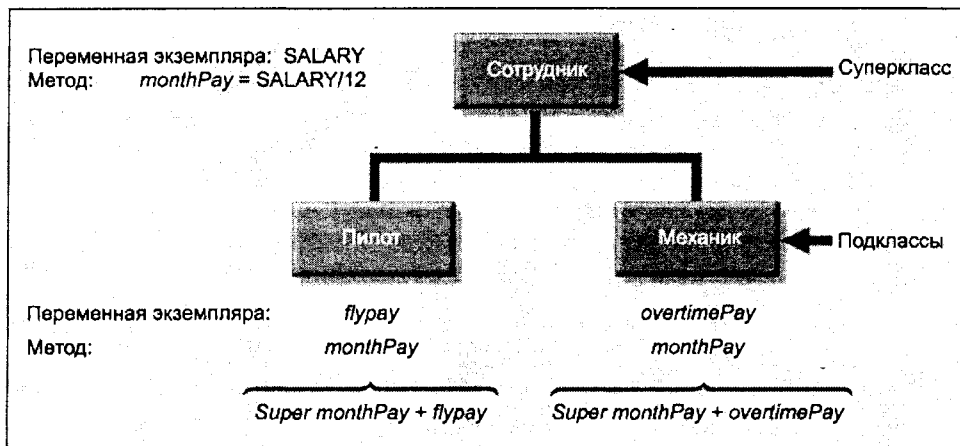


Рис. 11.13. Полиморфизм в иерархии класса Employee

### Примечание

На рис. 11.13 использован синтаксис Smalltalk: Выражение Super monthPay в методе monthPay класса Pilot указывает, что объект наследует метод monthPay суперкласса. Другие объектно-ориентированные языки программирования, например, C++, используют запись с точкой: Employee.monthPay.

На рис. 11.13 следует обратить внимание на следующие особенности полиморфизма:

- ❑ определение метода monthPay класса Pilot переопределяет и расширяет метод monthPay суперкласса Employee;
- ❑ метод monthPay, определенный в суперклассе Employee, повторно используется подклассами Pilot и Mechanic.

Полиморфизм усиливает переопределение метода, расширяя возможности многократного использования кода, что необходимо при модульном программировании и проектировании.

### 11.3.10. Абстрактные типы данных

Тип данных описывает множество объектов со схожими свойствами. Все традиционные языки программирования используют набор базовых типов данных (real, integer и string или character). Базовые типы данных подчиняются предопределенному набору операций. Например, базовый тип данных integer позволяет выполнять такие операции, как сложение, вычитание, умножение и деление.

В традиционные языки программирования включаются конструкторы типов, самым распространенным из которых является конструктор record (запись). Например, программист может определить запись типа CUSTOMER, определив для нее поля данных. Запись CUSTOMER будет представлять собой новый тип данных, в котором будет храниться информация о клиенте, и программист может напрямую получать доступ к этой структуре данных, ссылаясь на имена полей. Над записью можно выполнять такие операции, как WRITE (запись), READ (чтение) или DELETE (удаление). Для базовых типов данных определить новые операции нельзя.

Как и базовые типы данных, *абстрактные типы данных*, АТД (abstract data types) описывают множество схожих объектов. Однако есть определенные отличия абстрактного типа от традиционного типа данных:

- ☐ операции над АТД определяются пользователем;
- ☐ АТД не допускают непосредственного доступа к внутреннему представлению данных и реализации методов.

В некоторых ОО-системах, например, Smalltalk, базовые типы данных реализованы как абстрактные.

Для создания абстрактного типа данных необходимо обеспечить:

- ☐ имя типа;
- ☐ представление данных или переменные экземпляра объекта, принадлежащего АТД; каждая переменная экземпляра имеет тип данных, который может быть либо базовым типом, либо другим АТД;
- ☐ операции над абстрактными типами данных и ограничения реализуются с помощью методов.

Необходимо помнить, что определение абстрактного типа данных перестраивает определение класса. В некоторых ОО-системах для различения классов и типов при ссылке на структуры данных и методы класса используется ключевое слово type, а при ссылке на набор экземпляров объекта — ключевое слово class. Тип (type) — более статичное понятие, в то время как класс (class) связан в основном с временем выполнения.

Простой пример поможет понять это тонкое отличие ОО-класса от ОО-типа. Предположим мы приобрели шаблон для кружевных накидок на подушки. К шаблону прилагается описание его структуры, а также инструкция по его использованию. Этот шаблон и есть описание типа (type definition). Комплект сделанных с помощью шаблона реальных накидок, каждая из которых имеет уникальный номер (или OID), составляет класс (class).

Абстрактные типы данных совместно с наследованием позволяют создавать сложные объекты. *Сложный объект (complex object)* формируется путем комбинации других

объектов, находящихся в сложных взаимосвязях друг с другом. Пример сложного объекта можно найти в системах безопасности, где используются различные типы данных, например:

- стандартные (табличные) данные о сотруднике (имя, телефон, дата рождения и т. д.);
- битовая карта для хранения фотографии сотрудника;
- голосовые данные для хранения образца голоса сотрудника.

Возможность относительно просто работать с такой сложной средой данных повышает ставки ОО-систем на современном рынке баз данных.

### 11.3.11. Классификация объектов

Объект можно классифицировать в соответствии с его свойствами (простой, составной, сложный, смешанный и ассоциативный) или атрибутами.

В *простом объекте* (*simple object*) содержатся только однозначные атрибуты и нет атрибутов, которые ссылаются на другие объекты. Например, объект, который описывает текущий семестр, может быть определен следующими однозначными атрибутами: SEM\_ID (идентификатор семестра), SEM\_NAME (название семестра), SEM\_BEGIN\_DATE (дата начала семестра) и SEM\_END\_DATE (дата окончания семестра).

В *составном объекте* (*composite object*) содержится, по крайней мере, один многозначный атрибут, но нет атрибутов, ссылающихся на другие объекты. Например, объект Movie (фильм) может включать в себя следующие атрибуты: MOVIE\_ID (идентификатор фильма), MOVIE\_NAME (название фильма), MOVIE\_PRICE (стоимость фильма), MOVIE\_TYPE (жанр фильма) и MOVIE\_ACTORS (актеры, занятые в фильме). В данном случае атрибут MOVIE\_ACTORS является многозначным, описывающим нескольких исполнителей ролей в данной картине.

В *сложном объекте* (*compound object*) содержится, по крайней мере, один атрибут, который ссылается на другой объект. Примером может служить объект Student (студент) из табл. 11.2. В этом примере атрибут ADVISOR (куратор) ссылается на объект Professor (преподаватель).

В *смешанном объекте* (*hybrid object*) содержатся повторяющиеся группы атрибутов, из которых по крайней мере один ссылается на другой объект. Типичный пример смешанного объекта это счет-фактура, представленный на рис. 2.29. В этом случае счет содержит несколько товаров, а каждый товар представлен количеством проданных единиц и стоимостью единицы. Счет содержит повторяющиеся группы атрибутов (PROD\_CODE, LINE\_UNITS и LINE\_PRICE) для каждого проданного товара. Поэтому для представления этого объекта (счет) не требуется новый объект Inv\_line, как это представлено в ER-модели.

Наконец, *ассоциативный объект* (*associative object*) используется для представления связи между двумя или более объектами. В ассоциативном объекте могут содержаться его собственные атрибуты, представляющие специфические свойства связи. Хорошим примером ассоциативного объекта является Enroll (см. рис. 2.25). В этом случае объект Enroll связан с объектами Student и Class и включает в себя атрибут ENROLL\_GRADE, представляющий собой оценку студента в группе.

В реальных моделях данных вы обнаружите множество примеров простых и составных, смешанных и ассоциативных объектов. Мы подробно обсудим типы объектов далее в этой главе.

## 11.4. Свойства объектно-ориентированной модели данных

Объектно-ориентированные концепции, представленные в предыдущих разделах, представляют собой основные свойства *объектно-ориентированной модели данных* — ООМД (object-oriented data model, OODM), называемой также *объектной моделью данных*, ОМД (object data model, ODM). Объектная модель данных должна обладать, как минимум, следующими свойствами:

- ☐ поддерживать представление сложных объектов;
- ☐ обеспечивать расширение, т. е. должна иметься возможность определения новых типов данных, а также операций над ними;
- ☐ поддерживать инкапсуляцию, т. е. представление данных и реализация методов должны быть скрыты от внешних объектов;
- ☐ поддерживать наследование, т. е. любой объект может наследовать свойства (данные и методы) других объектов;
- ☐ обеспечивать идентификацию объекта (OID), описанную ранее в этой главе.

В учебных целях мы будем использовать определение и описание с помощью объектно-ориентированной модели данных (ООМД) компонентов, которые по возможности соответствуют компонентам ER-модели, описанным в гл. 3. Хотя все основные базовые компоненты ООМД нами уже определены, их краткая сводка может оказаться нелишней при дальнейшем изучении материала.

- ☐ В ООМД сущности реального мира моделируются *объектами*.
- ☐ Каждый объект состоит из *атрибутов* и набора *методов*.
- ☐ Каждый атрибут может *ссылаться* на другой объект или множество объектов.
- ☐ Атрибуты и реализации методов скрыты (*инкапсулированы*) от других объектов.
- ☐ Каждый объект идентифицируется *уникальным идентификатором объекта (OID)*, не зависящим от значений атрибутов этого объекта.
- ☐ Схожие объекты группируются в *класс*, который содержит описание данных (атрибуты или переменные экземпляров) и реализации методов.
- ☐ Класс описывает *тип* объекта.
- ☐ Классы организованы в *иерархию классов*.
- ☐ Каждый объект класса *наследует* все свойства своего суперкласса в иерархии классов.

Используя это сводное описание ОО-компонентов, сравните их с описанием ER-компонентов (табл. 11.3).

Таблица 11.3. Сравнение компонентов ОО- и ER-моделей

| Объектно-ориентированная модель данных | ER-модель            |
|----------------------------------------|----------------------|
| Тип                                    | Определение сущности |
| Объект                                 | Сущность             |
| Класс                                  | Набор сущностей      |
| Переменная экземпляра                  | Атрибут              |
| Не определено                          | Первичный ключ       |
| OID                                    | Не определено        |
| Метод                                  | Не определено        |
| Иерархия классов                       | ER-диаграмма         |

### 11.4.1. Схемы объектов: графическое представление объектов

Графически объект представляется в виде прямоугольника с именами переменных экземпляра. Вообще говоря, представление объекта соответствует всем объектам данного класса. Поэтому термины *объект* и *класс* на иллюстрациях часто взаимозаменяемы. Помня об этом, исследуем графическое представление класса Person, представленное на рис. 11.14. В этом случае для переменных экземпляров NAME (имя), ADDRESS (адрес), DOB (дата рождения) и SEX (пол) используются базовый тип данных string, а для переменной экземпляра AGE (возраст) используется базовый тип integer.

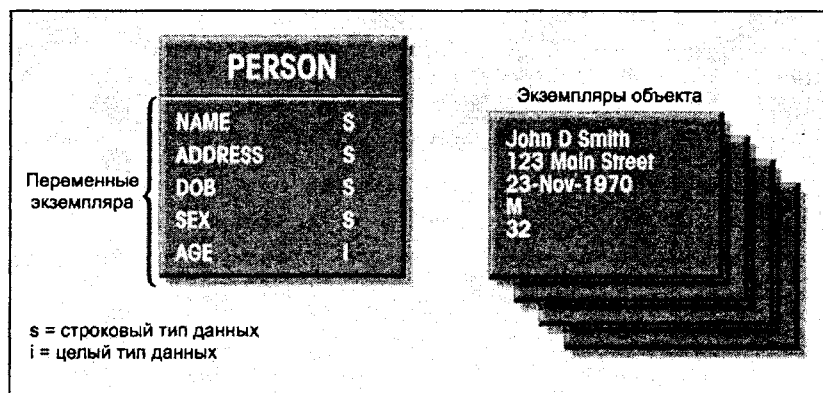


Рис. 11.14. Представление для всех объектов класса Person

Теперь рассмотрим состояние экземпляра объекта Person (рис. 11.15). Обратите внимание, что переменную экземпляра объекта AGE можно рассматривать как *про-*

*изводный (derived)* атрибут. Производные атрибуты могут быть реализованы с помощью методов. Например, для класса `Person` мы можем создать метод с названием `Age`. Этот метод возвращает разницу (в годах) между текущей датой и днем рождения (DOB) для данного экземпляра объекта. Поскольку методы могут создавать значения производных атрибутов, методы предоставляют дополнительное преимущество инкапсуляции и наследования.

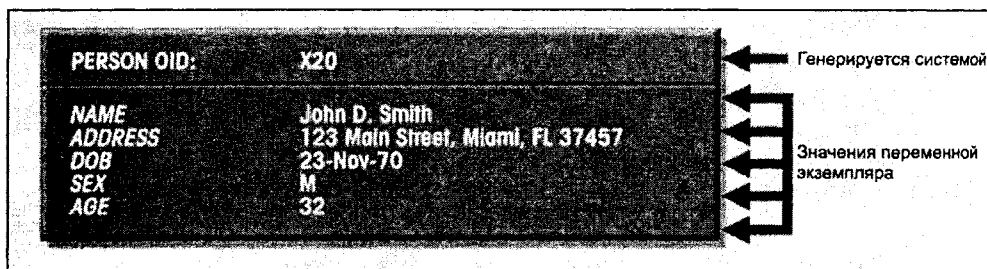


Рис. 11.15. Состояние экземпляра объекта `Person`

Необходимо помнить, что среда ОО позволяет создавать абстрактные типы данных (АТД) из основных типов. Например, `NAME`, `ADDRESS` и `DOB` являются сложными атрибутами, которые можно реализовать при помощи АТД. Для иллюстрации этого мы определили на рис. 11.16 `Name`, `Address` и `DOB` как абстрактные типы данных.

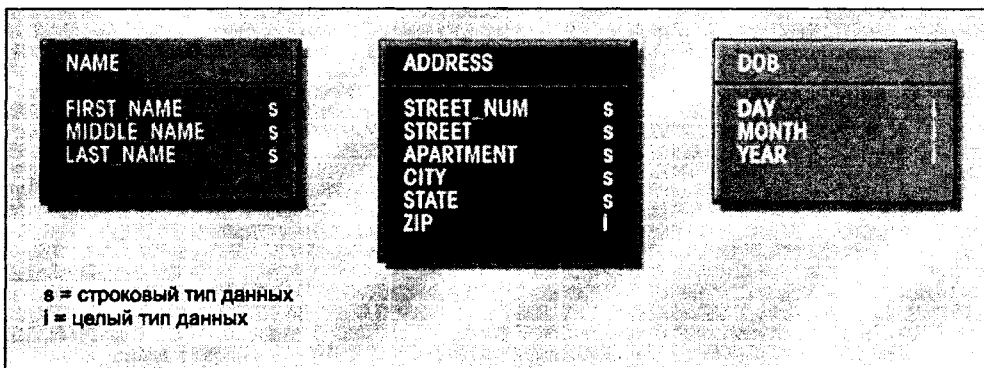


Рис. 11.16. Определение трех абстрактных типов данных

На рис. 11.16 обратите внимание, что класс `Person` теперь содержит атрибуты, которые указывают на объекты других классов или абстрактные типы данных. Новые типы данных для каждой переменной экземпляра класса `Person` представлены на рис. 11.17.

*Пространство объектов (object space)*, или *схема объектов (object schema)*, используется для представления состояния объекта в данный момент времени.

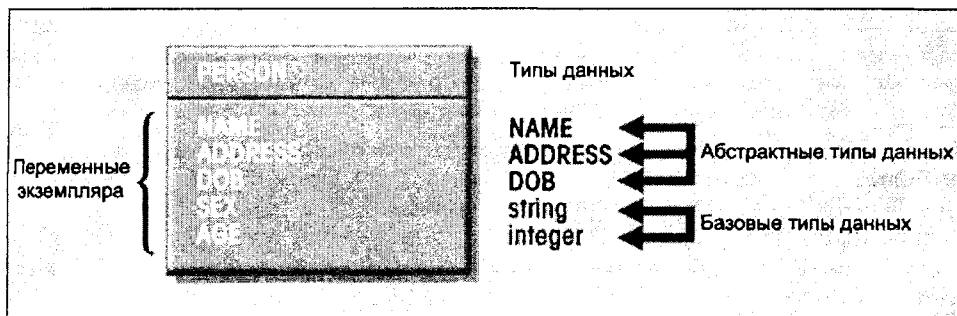


Рис. 11.17. Представление объектов для экземпляров класса Person с абстрактными типами данных

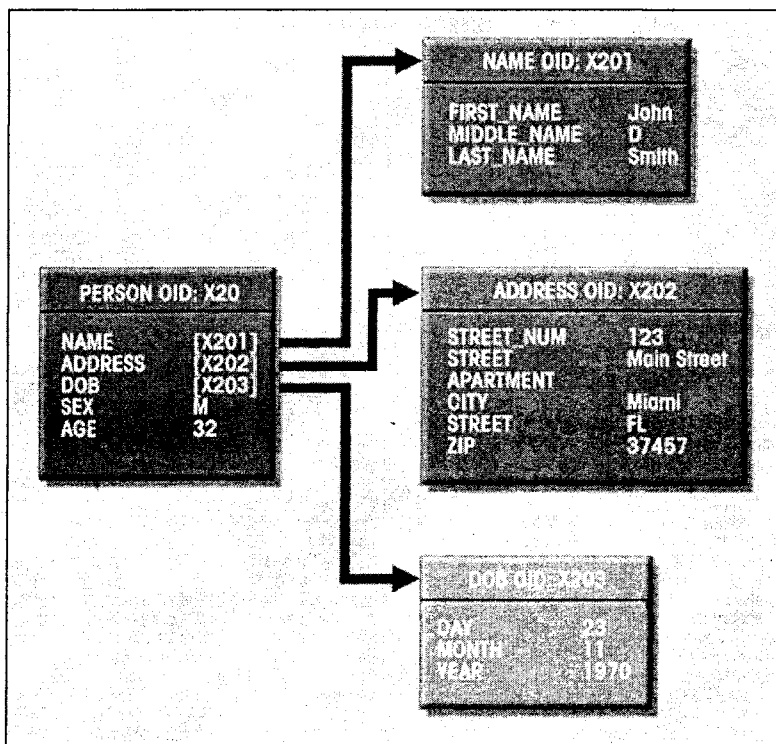


Рис. 11.18. Состояние объекта для экземпляра класса Person, использующего абстрактные типы данных

Состояние объекта для экземпляра класса Person представлено на рис. 11.18. Здесь обратите внимание на использование идентификаторов объектов (OID) для ссылки на другие объекты. Например, в атрибутах NAME, ADDRESS и DOB теперь содержится OID экземпляра соответствующего класса или абстрактного типа данных вме-

сто базового значения. Использование идентификатора OID для ссылки на объект позволяет избежать проблемы связности данных, которая может возникать в реляционных системах, если значение первичного ключа изменялось конечным пользователем во время изменения состояния объекта.

Для иллюстрации этого факта в дальнейшем мы будем использовать приложение учета арендной собственности, с помощью которого будем отслеживать арендуемые помещения и проживающих в них лиц. В данном случае два человека, живущих по одному адресу, будут, скорее всего, ссылаться на один и тот же экземпляр объекта Address (рис. 11.19). Это условие иногда называют *совместным использованием адресуемых объектов (referential object sharing)*. Изменение в экземпляре объекта Address будет отражено в обоих экземплярах Person.

На рис. 11.19 представлено состояние двух различных экземпляров объектов класса Person; оба экземпляра объекта ссылаются на один и тот же экземпляр объекта Address. Обратите внимание, что рис. 11.19 отображает четыре разных класса АТД: Person (два экземпляра), Name (два экземпляра), Address и DOB (два экземпляра).

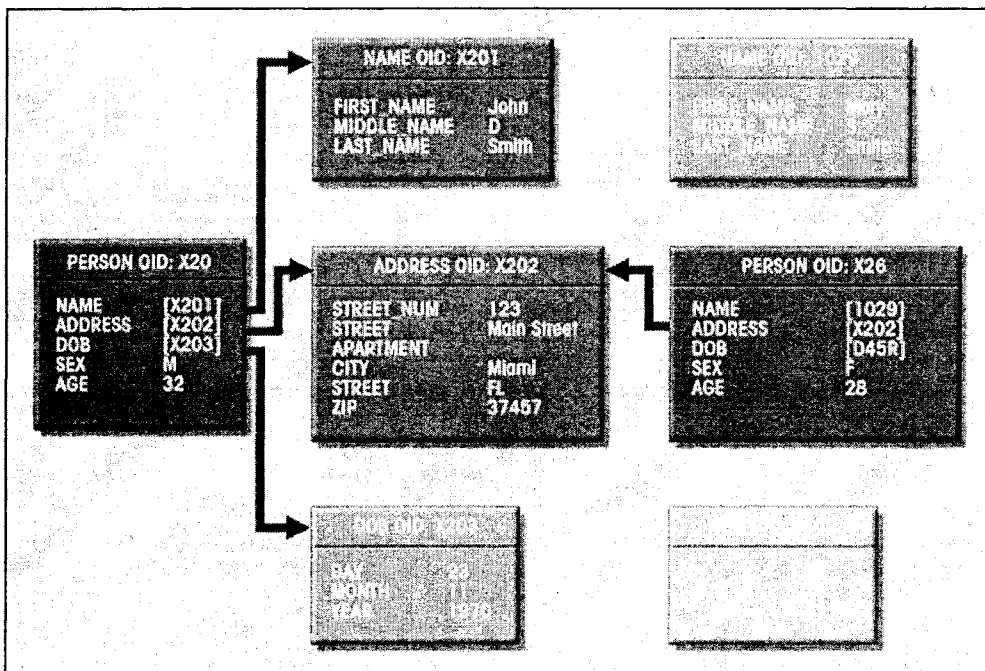


Рис. 11.19. Совместное использование объектов

## 11.4.2. Связи "класс-подкласс"

Вы помните, что класс наследует свойства своего суперкласса в иерархии? Это свойство приводит к использованию выражения "является" для описания связи между



классами в иерархии. То есть сотрудник *является* персоной и студент *является* персоной. Эта идея настолько важна, что требует более подробной иллюстрации иерархии класса (рис. 11.20).

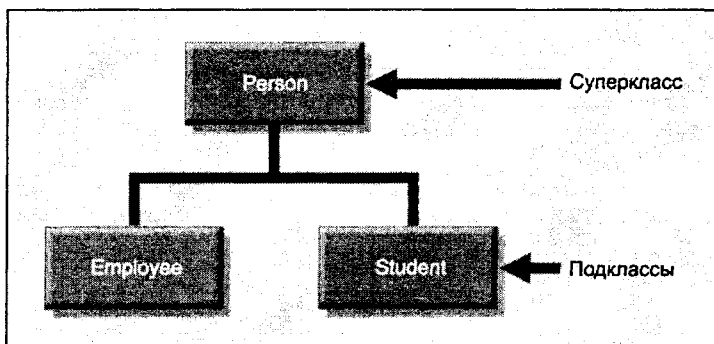


Рис. 11.20. Иерархия классов

В иерархии, представленной на рис. 11.20, объект Employee описывается семью атрибутами, представленными на рис. 11.21. Номер социального страхования (SS#) записывается как базовый тип string, а SALARY (зарплата) — как базовый тип integer. Имя, адрес, дата рождения, пол и возраст — все наследуется от суперкласса Person.

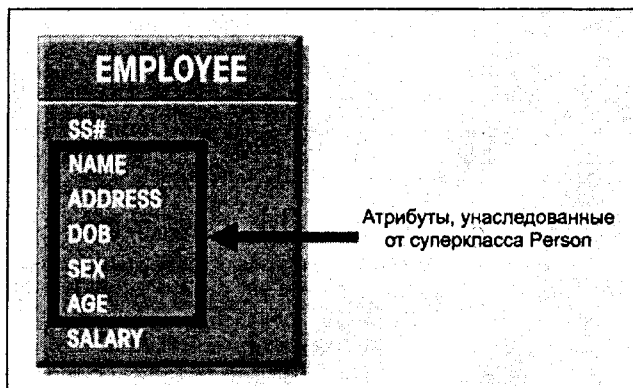


Рис. 11.21. Представление объекта Person

Этот пример основан на том факте, что ООМД поддерживает связь "класс-подкласс", для которой используются необходимые ограничения по целостности данных. Обратите внимание, что связь между суперклассом и подклассом относится к типу 1:M. То есть если мы имеем дело с единичным наследованием, то каждый суперкласс может иметь несколько подклассов, но каждый подкласс связан только с одним суперклассом.

### 11.4.3. Межобъектная связь "атрибут-класс"

Кроме поддержки связи "класс-подкласс", модель ООМД поддерживает связь "атрибут-класс". Эта связь (называемая также *межобъектной связью* — interobject relationship) создается в том случае, когда атрибуты объекта ссылаются на другой объект этого же или другого класса.

Межобъектная связь отличается от связи "класс-подкласс", которую мы только что изучили. Чтобы продемонстрировать эту разницу, исследуем иерархию класса для торгового предприятия EDLP (Every Day Low Prices, Ежедневное снижение цен), представленную на рис. 11.22.

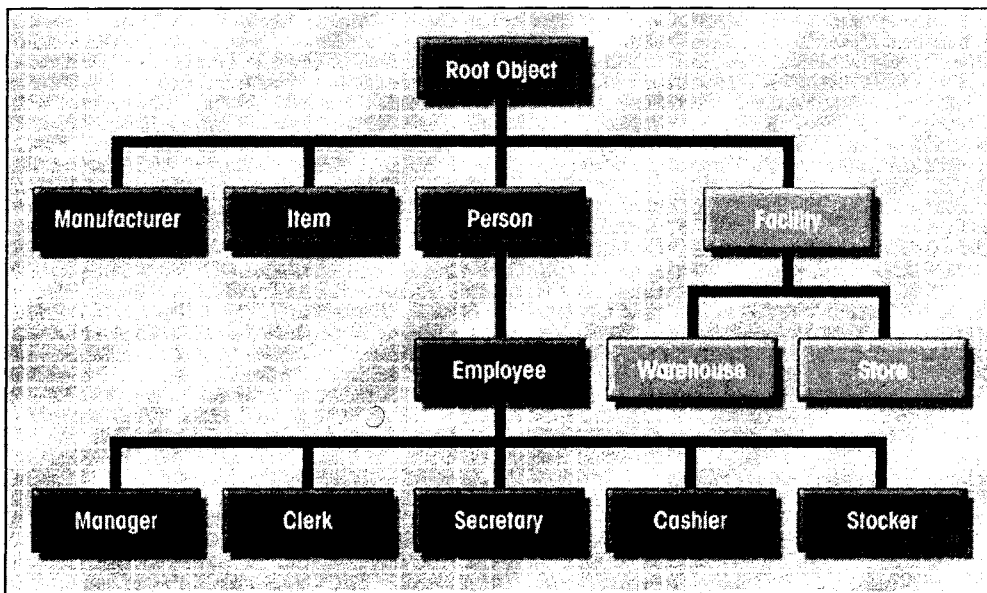


Рис. 11.22. Иерархия классов для торгового предприятия EDPL

Обратите внимание, что все классы основаны на суперклассе Root Object. В иерархии имеются классы Manufacturer (изготовитель), Item (предмет), Person (персона) и Facility (подразделение). Класс Facility содержит подклассы Warehouse (склады) и Store (магазины). Класс Person содержит подкласс Employee (сотрудники), в котором в свою очередь содержатся подклассы Manager (руководство), Clerk (чиновники), Secretary (секретари), Cashier (кассиры) и Stocker (кладовщики). Впоследствии мы будем использовать простую иерархию классов, представленную на рис. 11.22, для иллюстрации связей 1:M и M:N.

#### Представление связи 1:M

На основе данной иерархии классов (см. рис. 11.22) между Employee и Facility есть связь 1:M: каждый сотрудник работает в одном подразделении, но в каждом подраз-

делении работает несколько сотрудников. На рис. 11.23 показано, как можно представить такую связь.

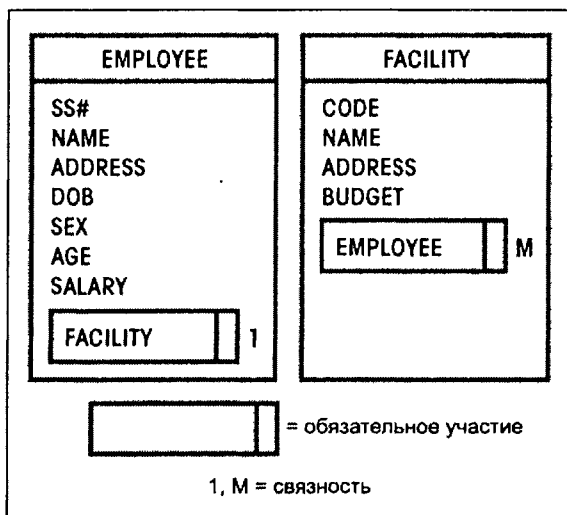


Рис. 11.23. Представление связи 1:M

Если внимательно посмотреть на связь между Employee и Facility, представленную на рис. 11.23, то можно заметить, что объект Facility включен в объект Employee и наоборот, объект Employee также включен в объект Facility. Для тщательного изучения связей мы будем использовать следующие графические средства:

- чтобы подчеркнуть связи, связанные классы заключаются в прямоугольники;
- прямоугольник с двойной линией на правой стороне указывает на обязательную связь;
- связность обозначается меткой у каждого прямоугольника. В данном случае следом за Facility в объекте Employee записана единица, "1", указывающая, что каждый сотрудник работает только в одном подразделении. "М" рядом с Employee в объекте Facility указывает на то, что в каждом подразделении работает много сотрудников.

Обратите внимание, что для представления обязательной сущности и обозначения связности связи (1:M) используется ER-нотация. Это сделано для согласования с ранее приведенными в книге диаграммами.

Вместо того чтобы включить прямоугольник объекта в класс, мы предпочитаем использовать имена, описывающие свойства класса, которые мы моделируем. Это особенно важно, когда два класса входят более чем в одну связь; в таких случаях лучше записывать имена атрибутов над прямоугольниками, изображающими класс, а в прямоугольнике указывать атрибут, с помощью которого можно ссылаться на этот класс. Например, мы можем представить две связи между Employee и Facility с помощью WORKS\_AT (работает в) и WORKERS (служащие), как показано на

рис. 11.24. Обратите внимание, что имеются две связи: связь 1:M, основанная на бизнес-правиле "в каждом подразделении работает множество сотрудников и каждый сотрудник работает только в одном подразделении" и связь 1:1, основанная на бизнес-правиле "каждым подразделением руководит только один сотрудник, и каждый руководитель управляет только одним подразделением".

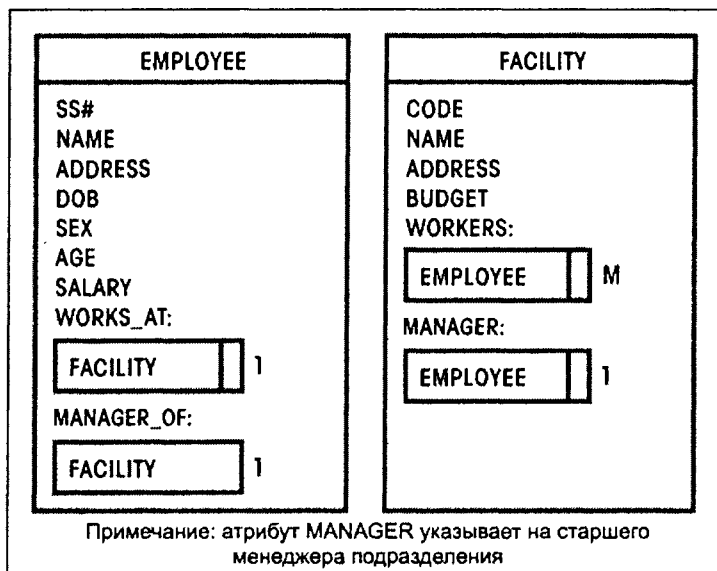


Рис. 11.24. Представление связей 1:1 и 1:M

Из рис. 11.24 следует, что связи представлены в обоих взаимодействующих классах. Это позволяет при необходимости инвертировать связь. Например, объект Facility внутри объекта Employee выражает отношение "руководит". В этом случае объект Facility необязателен и имеет максимальную связность 1. Объекты Employee и Facility представляют собой примеры сложных (compound) объектов. Другой тип связи (1:M) можно проиллюстрировать связью между сотрудниками и их иждивенцами. Чтобы установить такую связь, мы, прежде всего, создадим подкласс Dependent (иждивенцы) на основе суперкласса Person. Обратите внимание, что мы не можем создать подкласс Dependent на основе класса Employee и его суперкласса, поскольку эта иерархия класса представляет связь "является". Другими словами, каждый Manager (руководитель) является сотрудником (Employee), каждый Employee является Person (персоной), каждый иждивенец является Person (персоной) и каждая персона (Person) является объектом (Object) в нашем пространстве объектов — но каждый иждивенец (Dependent) не является сотрудником (Employee). На рис. 11.25 показано корректное представление связи между Employee и Dependent.

На рис. 11.25 обратите внимание, что объект Dependent необязателен для объекта Employee, и объект Dependent связан с объектом Employee связью 1:M. Однако объект Employee обязателен для объекта Dependent. В объектно-ориентированной модели

данных исчезает понятие слабой сущности, поскольку каждый объект идентифицируется своим уникальным *OID*.

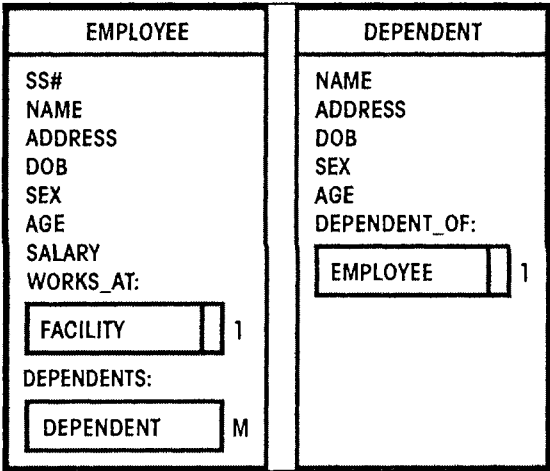


Рис. 11.25. Связь Employee-Dependent

Представление связи M:N

На основе иерархии классов упомянутого выше торгового предприятия EDLP тип связи M:N можно продемонстрировать на примере связи между Manufacturer и Item, как это показано на рис. 11.26. Здесь представлено условие, при котором каждый предмет (Item) может производиться многими производителями (Manufacturer), и каждый производитель (Manufacturer) может выпускать множество предметов (Item). Поэтому рис. 11.26 представляет собой концептуальное представление связи M:N между Item и Manufacturer. В этом представлении Item и Manufacturer являются сложными (compound) объектами.

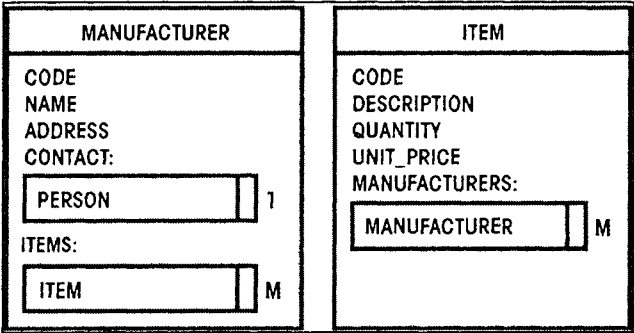


Рис. 11.26. Представление связи M:N

Также отметим, что на атрибут CONTACT (контакты) в классе Manufacturer на рис. 11.26 ссылается только один экземпляр класса Person. Здесь может возникнуть небольшое недоразумение: по всей видимости, у каждого контактного лица (персоны) имеется телефон, но мы не включили атрибут "номер телефона" в класс Person. В данном случае проектировщик может добавить этот атрибут, чтобы он был доступен всем подклассам класса Person.

## Представление связи M:N с классом пересечения

Предположим, что мы добавили к только что исследованному классу Item условие, позволяющее отслеживать дополнительную информацию по каждому предмету (Item). Например, представим связь между Item и Facility таким образом, что в каждом подразделении (Facility) может находиться несколько предметов (Item) и каждый предмет может находиться в нескольких подразделениях. Кроме того, нам необходимо отслеживать количество и местоположение (проход и ряд на складе, например) каждого предмета в каждом подразделении. Это условие представлено на рис. 11.27. Правая скобка (]) на рис. 11.27 указывает, что заключенные в нее атрибуты рассматриваются как логическая единица. Поэтому каждый экземпляр Item может содержать несколько вхождений Facility, каждое из которых сопровождается соответствующими значениями атрибутов AISLE (проход), ROW (ряд) и QTY\_ON\_HAND (имеющееся количество). Верно и обратное для каждого экземпляра Facility. Объекты Item и Facility в данном отношении являются смешанными (hybrid) с повторяющимися группами атрибутов. Отметим: семантика требует, что для определения прохода, ряда и количества каждого предмета необходимо в первую очередь обращаться к объектам Item и Facility.

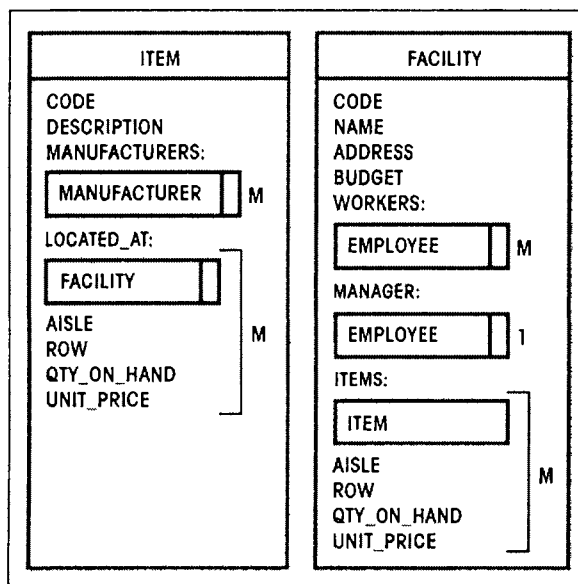


Рис. 11.27. Представление связи M:N с ассоциированными атрибутами

Чтобы это описание связи M:N больше соответствовало реляционному представлению, необходимо определить *класс пересечения* — *мост* (*intersection class*) для соединения Facility и Item и хранения связанных с ними атрибутов. В данном случае мы можем создать класс ассоциативного объекта Stocked\_Item (хранимые предметы) для размещения в нем экземпляров объектов Facility и Item и значений каждого из атрибутов AISLE, ROW и QTY\_ON\_HAND. Такой класс эквивалентен конструкции Interclass\_Connection семантической модели данных. На рис. 11.28 показано, как представляются экземпляры объектов Item, Facility и Stocked\_Item.

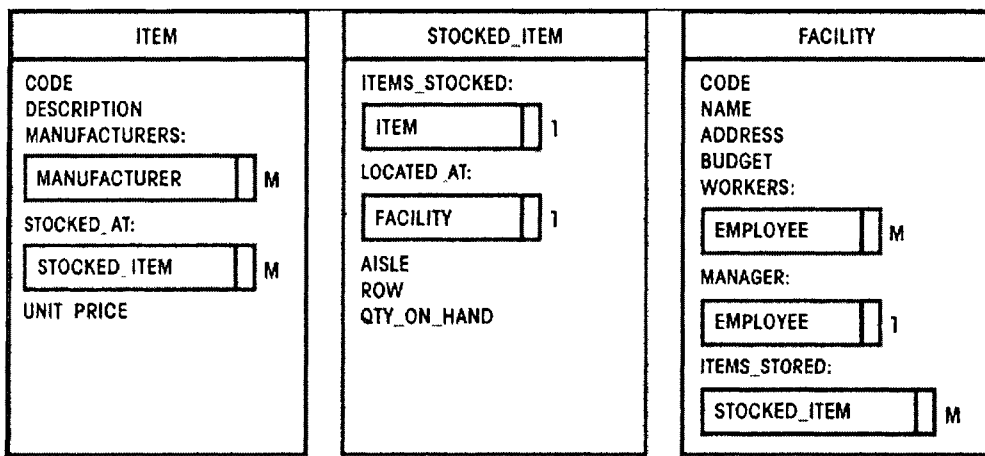


Рис. 11.28. Представление связи M:N с классом пересечения

Изучив представление основных ОО-отношений, мы можем представить пространство объектов в том виде, как это показано на рис. 11.29.

На рис. 11.29 много важной для проектирования информации; необходимо обратить особое внимание на следующее:

- ❑ экземпляр ассоциативного объекта Stocked\_Item содержит в себе ссылки на экземпляры каждого связанного с ним класса (Item и Facility). Класс пересечения Stocked\_Item необходим, только если нужно отслеживать дополнительную информацию, о которой мы говорили выше;
- ❑ в экземпляре объекта Item в этой схеме содержится набор экземпляров объекта Stocked\_Item, каждый из которых содержит экземпляр объекта Facility. Инверсия этого отношения также верна: экземпляр объекта Facility содержит набор экземпляров объекта Stocked\_Item, каждый из которых содержит экземпляр объекта Item. *Необходимо понимать, что эти две связи представляют два разных вида одной и той же схемы объекта;*
- ❑ при ссылках на объекты для получения доступа к объектам и включения их в пространство объекта используется OID;

- значения внутри квадратных скобок ( [ и ] ) представляют собой OID экземпляра объекта некоторого класса. Классы "набор из" представляют собой класс объектов, в которых каждый экземпляр объекта содержит набор объектов некоторого класса. Например, Z3402 и Z45621 представляют собой ссылки на объекты, которые состоят из набора Manufacturer (производители) и набора Stocked\_Item (храняемые предметы) соответственно;
- в реляционной модели таблица ITEM не содержит каких-либо данных, связанных с Manufacturer или Stocked\_Item в своей структуре. Здесь для представления информации (составной) о ITEM, Stocked\_Item и Facility мы должны выполнить реляционный оператор JOIN. В ООМД нет необходимости использовать оператор JOIN, чтобы комбинировать данные из разных объектов, поскольку объект ITEM *содержит* в себе ссылки на связанные с ним объекты; эти ссылки автоматически вводятся в пространство объекта при получении доступа к объекту ITEM.

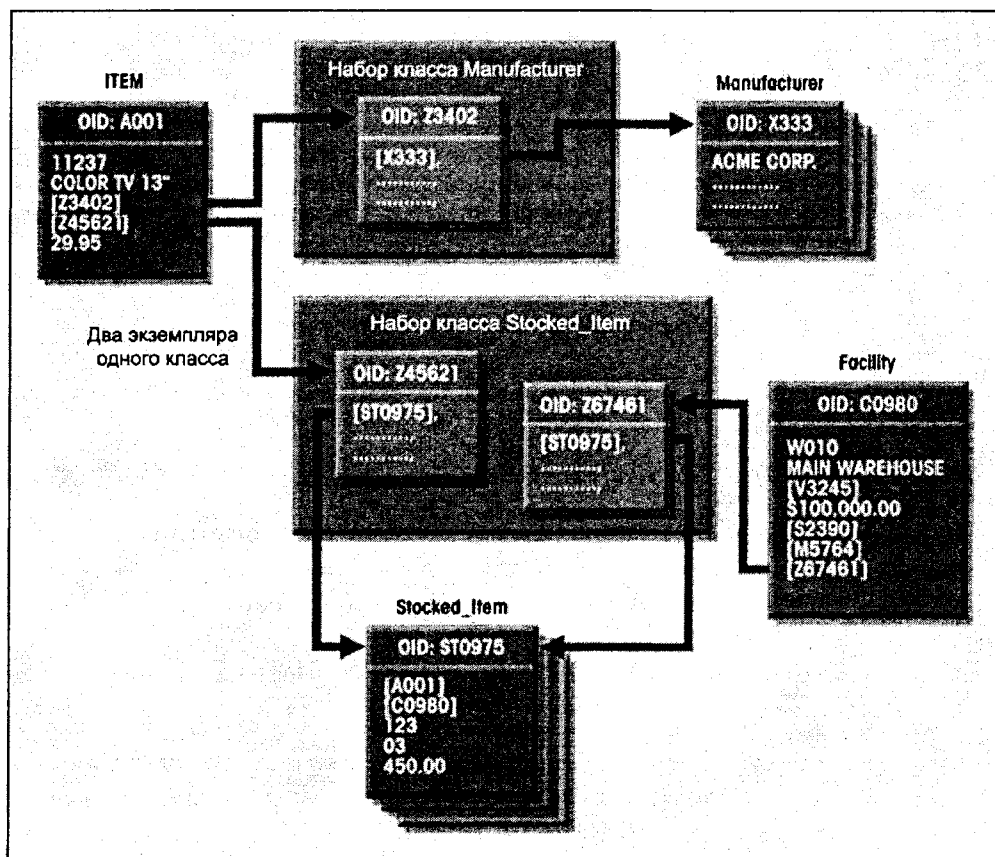


Рис. 11.29. Представление пространства объектов



#### 11.4.4. Позднее и раннее связывание: предназначение и использование

Возможность размещения в атрибуте других объектов, определяющих в разное время различные типы данных (или классы), — очень привлекательное свойство ООМД. При этом данный объект может содержать *числовое значение* данной переменной экземпляра, а следующий объект (того же класса) может содержать *строковое значение* этой же переменной экземпляра. Это свойство достигается с помощью позднего связывания. *Позднее связывание (late binding)* атрибута с типом данных неизвестно до момента выполнения (прогона). Поэтому два различных экземпляра объекта одного и того же класса могут содержать значения различных типов данных для одного и того же атрибута.

В противоположность ООМД, где имеется возможность позднего связывания, в традиционных СУБД требуется, чтобы базовый тип данных был определен для каждого атрибута при его создании. Предположим, что нам необходимо определить сущность INVENTORY (склад) со следующими атрибутами: ITEM\_TYPE (тип), DESCRIPTION (описание), VENDOR (поставщик), WEIGHT (вес) и PRICE (стоимость). В традиционных СУБД вы создаете таблицу с именем INVENTORY и присваиваете тип данных каждому атрибуту, как это показано на рис. 11.30.

Таблица: INVENTORY

| Атрибуты    | Стандартные (базовые) типы данных |
|-------------|-----------------------------------|
| ITEM_TYPE   | Numeric                           |
| DESCRIPTION | Character                         |
| VENDOR      | Numeric                           |
| WEIGHT      | Numeric                           |
| PRICE       | Numeric                           |

**Рис. 11.30.** Таблица INVENTORY  
с predeterminedными (базовыми) типами данных

Из предыдущих глав вам известно, что при работе с традиционными СУБД проектировщик должен определить тип данных для каждого атрибута *во время определения структуры таблицы*. Такой подход определения типа данных называется *ранним связыванием (early binding)*. Раннее связывание позволяет проверять тип данных каждого значения атрибута во время компиляции и определения таблиц. Например, атрибут ITEM\_TYPE на рис. 11.30 ограничен числовым значением. Точно так же

атрибут **VENDOR** может содержать только числовые значения, чтобы соответствовать первичному ключу некоторой строки в *таблице* **VENDOR**, которые тоже ограничиваются числовыми значениями.

Теперь взглянем на рис. 11.31, чтобы узнать, как в ООМД решается задача раннего связывания. Как и в традиционных СУБД, модель ООМД допускает определение типов данных на этапе проектирования. Однако в отличие от традиционных СУБД, в ООМД пользователю разрешается определять абстрактные типы данных (АТД). В данном примере, где представлено раннее связывание, абстрактные типы данных **Inv\_type**, **String\_of\_characters**, **Vendor**, **Weight** и **Money** ассоциированы с переменными экземпляра на этапе их определения. Поэтому проектировщик может определить необходимые операции для каждого типа данных. Например, тип данных **Weight** может иметь методы для вывода веса предмета в фунтах, килограммах и т. д. Точно так же тип данных **Money** может обладать методами, возвращающими стоимость в цифрах и символах для пересчета в доллары США (\$), евро (€) или английские фунты (£). Следует помнить, что абстрактные типы данных реализуются через классы.

**Класс: INVENTORY**

| Переменные экземпляра | Тип                         |
|-----------------------|-----------------------------|
| <b>ITEM_TYPE</b>      | <b>Inv_type</b>             |
| <b>DESCRIPTION</b>    | <b>String_of_characters</b> |
| <b>VENDOR</b>         | <b>Vendor</b>               |
| <b>WEIGHT</b>         | <b>Weight</b>               |
| <b>PRICE</b>          | <b>Money</b>                |

**Рис. 11.31.** Класс **INVENTORY** с ранним связыванием

В средах с поздним связыванием тип данных атрибута объекта до его использования неизвестен. Поэтому любой атрибут может иметь любой тип присваиваемого ему значения. С помощью представленных ранее базовых типов данных на рис. 11.32 показаны атрибуты (переменные экземпляра) **INV\_TYPE**, **DESCRIPTION**, **VENDOR**, **WEIGHT** и **PRICE** без предварительного определения типа данных. Поскольку для переменных экземпляра класса типы данных заранее не определяются, два разных объекта класса **INVENTORY** могут иметь разные типы данных атрибута. Например, атрибуту **INV\_TYPE** может быть присвоено символьное (строковое) значение в одном экземпляре и численное значение в другом. Позднее связывание также играет важную роль в полиморфизме и позволяет объектам выбирать реализацию метода, которую необходимо использовать во время выполнения.

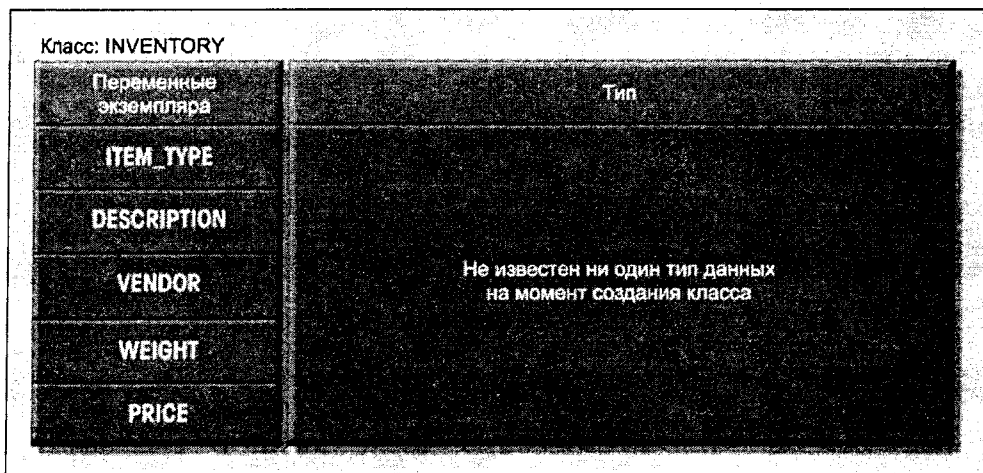


Рис. 11.32. Класс INVENTORY модели ООМД с поздним связыванием

### 11.4.5. Поддержка контроля версий

*Контроль версий (versioning)* представляет собой функциональную возможность модели ООМД, позволяющую отслеживать историю изменений состояния объекта. Контроль версий — весьма мощное и полезное свойство, особенно в среде автоматизированного проектирования (CAD, Computer Aided Design). Например, инженер с помощью системы CAD может загружать компоненты проекта на свою рабочую станцию, вносить некоторые изменения и видеть, как эти изменения влияют на взаимодействие компонентов. Если внесенные изменения не дают ожидаемого результата, инженер может отменить все действия и восстановить компоненты в их оригинальное состояние. Возможность контроля версий является одной из причин, по которой ООСУБД играют ведущую роль в системах CAD/CAM (системы автоматизированного проектирования и изготовления).

## 11.5. ООМД и прежние модели данных: сходство и различие

Несмотря на то что ООМД внешне очень похожа на реляционную модель или ER-модель, она по своей основе сильно отличается от них. Далее приводится подробная сводка всех основных свойств модели ООМД, представленных в этой главе.

### 11.5.1. Объект, сущность и кортеж

Концепция объекта в ООМД расширяет прежнее понятие *сущности (entity)* или *кортежа (tuple)* из других моделей данных. Хотя объект ООМД очень напоминает сущность и кортеж в ER-модели и реляционной схеме, все же объект ООМД обладает

важными дополнительными свойствами, такими как поведение, наследование и инкапсуляция. Эти замечательные свойства ООМД делают ОО-моделирование более приближенным к реальной действительности, чем ER-моделирование и реляционное моделирование. Фактически ER-моделирование и реляционные модели часто вынуждают проектировщика создавать новые искусственные сущности для того, чтобы выразить объекты реального мира. Например, в ER-модели счет-фактура обычно представляется двумя отдельными сущностями; вторая сущность (LINE — строка) обычно является слабой, поскольку зависит от первой (INVOICE — счет) и ее первичный ключ частично производится из сущности INVOICE (рис. 11.33).

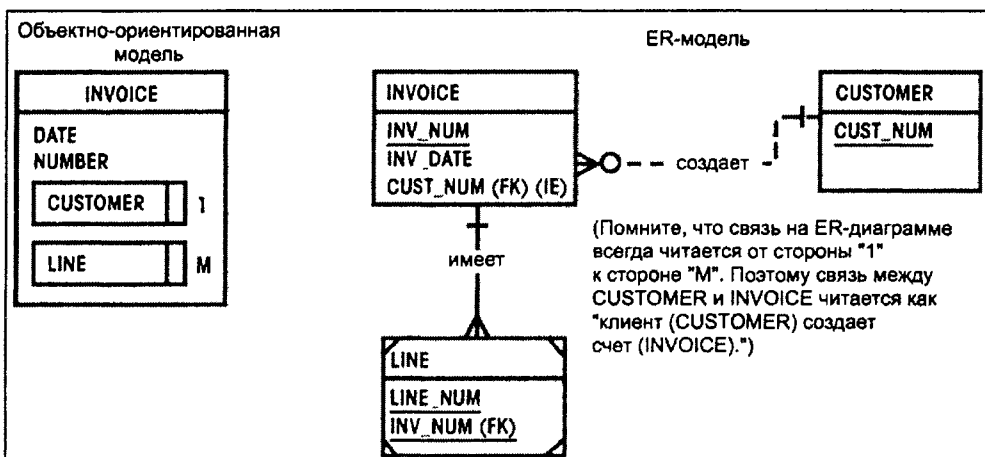


Рис. 11.33. Представление счета-фактуры в ER-модели

На рис. 11.33 видно, что при ER-подходе к моделированию реальной сущности "счет-фактура" требуется использование двух различных сущностей. Такая искусственная конструкция продиктована ограничениями на наследование в реляционной модели. Искусственное представление ER-модели приводит к дополнительным непроизводительным издержкам в основной системе. В отличие от этого объект INVOICE в ООМД напрямую моделируется как объект в пространстве объекта или в схеме объекта.

### 11.5.2. Класс, набор сущностей и таблица

Понятие *класс* можно увязать с понятиями *набор сущностей* и *таблица* в ER-модели и реляционной модели соответственно. Однако класс представляет собой более мощное понятие, позволяющее описывать не только структуру данных, но и поведение объектов класса. Класс также позволяет определить и реализовать в модели ООМД абстрактные типы данных (АТД). АТД являются мощным средством моделирования, поскольку они позволяют конечным пользователям создавать новые типы данных и использовать их как традиционные типы данных в среде БД. Таким образом, АТД усиливают семантическое содержимое моделируемых объектов.

### 11.5.3. Инкапсуляция и наследование

Абстрактные типы данных (АТД) предоставляют нам и две другие ОО-возможности, не поддерживаемые предыдущими моделями — инкапсуляцию и наследование. Классы организуются в иерархии классов. Объект, принадлежащий классу, наследует все свойства его суперкласса. Инкапсуляция означает, что представление данных и реализация методов скрыты от других объектов и от конечного пользователя. В модели ООМД только методы могут получать доступ к переменным экземпляра. В отличие от этого к компонентам данных или полям в традиционных системах можно получить прямой доступ из внешней среды.

В традиционных моделях отсутствуют методы, составляющие основу ООМД. Наиболее близкое понятие к *методам* представляют собой *триггеры* и *хранимые процедуры* в базах данных SQL. Однако поскольку в триггерах не используются преимущества инкапсуляции и наследования, свойственные методам ООМД, триггеры не могут обеспечить тех же функциональных возможностей.

### 11.5.4. Идентификатор объекта (OID)

Идентификатор объекта (OID) не поддерживается ни в ER-моделировании, ни в реляционных моделях. Хотя пользователи БД могут возразить, что Sequences в Oracle и AutoNumber в MS Access предоставляют возможность идентификации, однако это справедливо только в рамках уникальной идентификации элементов данных. К тому же в отличие от объектной модели, в которой связи явно не выражены, реляционная модель использует связи, основанные на значениях, например:

```
SELECT *
FROM INVOICE, INV_LINE
WHERE INVOICE.INV_ID = INV_LINE.INV_ID;
```

Иерархические модели и модели CODASIL поддерживают некоторые формы идентификации, которые можно считать похожими на OID, что является аргументом в пользу исследователей, утверждающих, что развитие ОО является шагом назад к старым системам указателей. Поэтому они считают, что ОО-системы возвращают нас к сложному моделированию и реализации, типичным для иерархических и сетевых моделей.

### 11.5.5. Связи

Основное свойство любой модели данных основано на представлении связей (relationships) между компонентами данных. В ООМД есть связи двух типов: ссылки между классами или наследование в иерархии классов. ER- и реляционные модели используют связи на основе значений. Это означает, что связи между сущностями устанавливаются через общие значения в одном или нескольких атрибутах сущности (концептуально это очень просто представить и отобразить!). В отличие от этого в ООМД для связей между объектами используется подход, основанный на идентификации (OID), поэтому связи не зависят от состояния объекта (это свойство упрощает работу с объектами БД и пользовательскими приложениями, но за это удобно приходится платить повышенной концептуальной сложностью).

## 11.5.6. Доступ

ER-модели и реляционные схемы в большой степени зависят от использования SQL для поиска и извлечения данных. SQL — это язык запросов, ориентированный на множества, основанный на формальных математических моделях. Поэтому в SQL используются ассоциативные методы доступа к данным для получения информации из БД на основе значений некоторых атрибутов. Например, для получения списка клиентов на основе суммарного значения их годовых покупок в SQL используется такое выражение:

```
SELECT *
FROM CUSTOMER
WHERE CUS_YTD_BUYS >= 5000;
```

Если значение параметра CUS\_YTD\_BUYS не определено, SQL "понимает", что условие означает "все значения" и сокращает оператор запроса следующим образом:

```
SELECT *
FROM CUSTOMER;
```

Поскольку семантика в ОО-модели данных более выраженная, в модели ООМД создается схема, в которой связи составляют часть структуры БД. Доступ к пространству структурированных объектов похож на доступ в режиме *последовательной обработки записей* (*record-at-time access*) в старых иерархических и сетевых моделях, особенно при использовании языков 3GL или даже объектно-ориентированных языков программирования, поддерживаемых ООСУБД. Модель ООМД приспособлена для поддержки как навигационного (последовательного) доступа, так и доступа, ориентированного на множественность значений. Навигационный доступ реализуется в ООМД напрямую с помощью идентификаторов OID, которые используются для навигации по структуре пространства объектов, разработанной программистом.

Ассоциативный доступ, ориентированный на множества, в ООМД осуществляется с помощью явно определенных методов. Поэтому проектировщик должен реализовать операции для манипуляции экземплярами объектов в схеме объекта. Реализация таких операций влияет на производительность БД и возможности управления данными. Это было главной проблемой при появлении объектной модели: недостаточная универсальность основных математических моделей для манипуляции данными. Отсутствие такого универсального доступа препятствует широкому распространению ООМД, поскольку каждая реализация требует создания собственной версии *объектно-ориентированного языка запросов* (OQL, Object Query Language). OQL — это язык запросов к базе данных, используемый в ООСУБД. Конечно, различные поставщики создают различные версии OQL, и это в свою очередь ограничивает возможность взаимодействия. Тем не менее, есть несколько групп (например, Object Management Group, OMG — рабочая группа по развитию стандартов объектного программирования), которые в настоящее время работают над разработкой стандартов объектно-ориентированных технологий. Группа OMG очень интенсивно работает над созданием стандарта OQL<sup>2</sup>.

<sup>2</sup> Во время написания этой книги OQL еще нельзя было считать стандартом ОО. Зайдите на сайт OMG ([www.omg.org](http://www.omg.org)), чтобы посмотреть самую свежую информацию о разработке OQL.

В OQL нет стандартного способа манипулирования множествами, а в стандартном SQL реляционной модели нет возможности манипулирования объектами объектно-ориентированной БД. Однако различие между OQL и SQL несколько уменьшилось после опубликования ANSI нового стандарта SQL в 1999 году. Этот стандарт SQL, известный как SQL3 или SQL-99, ведет к интеграции объектно-ориентированных свойств и реляционных баз данных. Для получения более подробных сведений об этом стандарте см. документ ANSI/ISO/IEC за номером 9075, ч. 1—5 на [www.ansi.org](http://www.ansi.org) (щелкните мышью на **nssn**, затем введите "SQL" в окне поиска).

В предыдущих разделах мы обсудили объектно-ориентированные концепции, унаследованные от объектно-ориентированных языков программирования. Мы использовали эти понятия для выявления свойств объектно-ориентированной модели данных (ООМД) и изучения ее графического представления. В следующем разделе мы сравним ООМД с предшествующими моделями данных и поясним, что одна из основных проблем ООМД — отсутствие в ней стандартного универсального метода доступа к данным. Хотя такого стандарта нет, есть соглашение о минимальных свойствах ООСУБД. Эти свойства мы также исследуем в следующем разделе.

## 11.6. Объектно-ориентированная система управления базой данных (ООСУБД)

В течение последних нескольких лет управление данными и среда приложений стали гораздо сложнее, чем могли себе это представить разработчики иерархических, сетевых или реляционных СУБД. Такие сложные структуры приложений могут проще всего обслуживаться *объектно-ориентированными СУБД (ООСУБД)*. ООСУБД представляет собой СУБД, в которой используются все преимущества обычных систем баз данных с мощными возможностями моделирования ООМД (рис. 11.34).

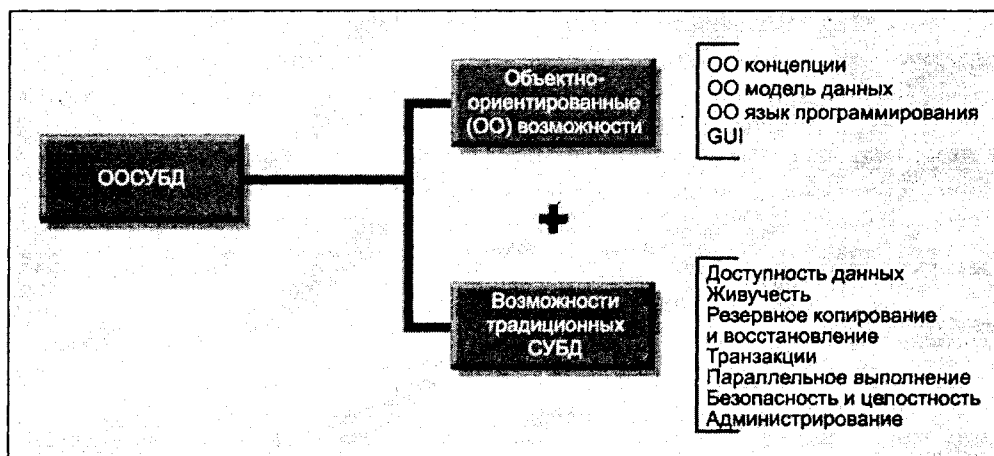


Рис. 11.34. Объектно-ориентированная СУБД

### Примечание

СУБД, основанная на объектной модели, может называться объектно-ориентированной СУБД (ООСУБД) или объектной СУБД (ОСУБД). Обозначение "ОО" и "ООСУБД" чаще использовалось на ранних стадиях развития объектно-ориентированного подхода, мы используем обозначение ООСУБД как наиболее предпочтительное для нас лично.

Некоторые продукты в области ООСУБД, такие как ObjectStore корпорации Object Design, Objectivity/DB корпорации Objectivity, Versant ODBMS корпорации Versant, ONTOS\*Integrator корпорации ONTOS и FastObjects компании Poet Software, используются при проектировании перечисленных ниже сложных систем.

- ☐ Медицинские приложения, управляющие цифровыми данными при рентгеновской съемке, магнито-резонансном и ультразвуковом сканировании вместе с текстовыми данными в медицинских исследованиях и при анализе истории болезни пациента.
- ☐ Финансовые приложения в управлении портфелем активов и рисковом управлении. Такие приложения отображают данные в реальном времени и основаны на многочисленных расчетах и агрегатных вычислениях над данными, извлекаемыми из сложных глобальных транзакций. Такие приложения могут управлять данными, расположенными по "временным рядам", как неким типом данных, определенным пользователем со своим внутренним представлением и методами.
- ☐ Телекоммуникационные приложения, например, приложения, управляющие конфигурацией коммуникационных сетей, которые автоматически наблюдают, отслеживают и переконфигурируют сеть на основе сотен параметров в реальном масштабе времени. Для поддержки своих телекоммуникационных приложений ООСУБД используют такие компании, как Ericsson, Ameritech и Bay Networks. Система Iridium Global Communications System компании Motorola управляет сложной сетью спутников и наземных станций также с помощью ООСУБД.
- ☐ В эксперименте Стэндфордского центра Linear Accelerator BaBar Physics с помощью ООСУБД ежедневно вводится 1 терабайт данных.
- ☐ В системах автоматизированного проектирования (САПР) и системах автоматизированного управления (САУ). В таких приложениях используются сложные отношения данных, а также множественные типы данных.
- ☐ В системах CASE (автоматизированная разработка программного обеспечения), создаваемых для управления большими объемами взаимосвязанных данных.
- ☐ Мультимедийные приложения, например, географические информационные системы (геоинформационные системы, ГИС), в которых используются видео, звук и высококачественная графика, требующие специальных возможностей по управлению данными (например, перекрывание, вхождение, указание, обводка и выделение).

Многие ООСУБД используют подмножество функциональностей объектно-ориентированных моделей. Поэтому создатели ООСУБД стараются выбрать возможности ОО-подхода, ориентированные на конкретную ООСУБД, например, поддержку раннего и позднего связывания типов данных и методов и поддержку единичного и множественного наследования. Каков бы ни был выбор, важнейшим фактором успеха реализации ООСУБД является наилучшее сочетание возможностей ОО и традиционных СУБД без принесения в жертву тех или иных средств.



### 11.6.1. Возможности ООСУБД

Как показано на рис. 11.34, ООСУБД представляет собой результат комбинирования объектно-ориентированных возможностей, таких как наследование класса, инкапсуляция и полиморфизм, с возможностями БД, такими как целостность, безопасность, непрерывность, управление транзакциями, управление параллельным выполнением, резервное копирование, восстановление, манипулирование данными и настройка системы. Публикация "Object-Oriented Database System Manifesto" (Atkinson и др., 1989)<sup>3</sup> — первая внятная попытка определить возможности ООСУБД — включает 13 обязательных свойств, а также необязательные свойства ООСУБД. Эти тринадцать правил представлены в табл. 11.4. Мы вкратце обсудим каждое из них.

**Таблица 11.4. Тринадцать правил для ООСУБД**

| <b>Правила, позволяющие считать систему объектно-ориентированной</b> |                                                                                                |
|----------------------------------------------------------------------|------------------------------------------------------------------------------------------------|
| Правило 1                                                            | Система должна поддерживать сложные объекты                                                    |
| Правило 2                                                            | Система должна поддерживать идентификацию объекта                                              |
| Правило 3                                                            | Объекты должны быть инкапсулированы                                                            |
| Правило 4                                                            | Система должна поддерживать типы или классы                                                    |
| Правило 5                                                            | Система должна поддерживать наследование                                                       |
| Правило 6                                                            | Система должна избегать раннего связывания                                                     |
| Правило 7                                                            | Система должна быть законченной в вычислительном отношении                                     |
| Правило 8                                                            | Система должна быть расширяемой                                                                |
| <b>Правила, позволяющие считать систему СУБД</b>                     |                                                                                                |
| Правило 9                                                            | Система должна помнить местонахождение данных                                                  |
| Правило 10                                                           | Система должна обеспечивать возможность управления большими базами данных                      |
| Правило 11                                                           | Система должна обслуживать нескольких пользователей                                            |
| Правило 12                                                           | Система должна обеспечивать возможность восстановления после аппаратного или программного сбоя |
| Правило 13                                                           | Запросы к данным должны быть простыми                                                          |

<sup>3</sup> Malcolm Atkinson и др., "The Object-Oriented Database System Manifesto". Этот официальный документ впервые был представлен на I Международной Конференции по Дедуктивным и Объектно-Ориентированным Базам Данных (First International Conference on Deductive and Object-Oriented Databases), Киото, Япония, 1989 год. Его можно найти на сайте группы Object Data Management Group [www.odmg.org](http://www.odmg.org). Выберите на этой странице *White Papers*, затем *Database* и *The OO Database System Manifesto*. Здесь вы также сможете найти серию официальных документов, посвященных текущим (1997—1999) стандартам и расширениям правил. Эти документы разработаны для подтверждения манифеста и никоим образом его не подменяют.

- ❑ *Правило 1. Система должна поддерживать сложные объекты.* В системе должна быть возможность конструирования сложных объектов из имеющихся в наличии. Примерами таких объектов могут быть множества, списки и кортежи, которые позволяют пользователю определять объединения объектов в качестве атрибутов.
- ❑ *Правило 2. Система должна поддерживать идентификацию объекта.* OID должен быть независимым от состояния объекта. Эта возможность позволяет сравнивать объекты в системе на двух уровнях: сравнение OID (идентичность объектов) и сравнение состояния объектов (равные или почти равные объекты).

### Примечание

Если объекты имеют различные OID, но их атрибуты совпадают, то эти объекты не считаются идентичными, но рассматриваются как почти равные ("shallow equal"). Точно так же, как близнецы хотя и очень похожи, но все же отличаются.

- ❑ *Правило 3. Объекты должны быть инкапсулированы.* Объекты имеют внешний интерфейс, но внутреннюю (скрытую) реализацию данных и методов. Возможность инкапсуляции гарантирует, что видима только внешняя сторона объекта, в то время как подробности реализации скрыты.
- ❑ *Правило 4. Система должна поддерживать типы или классы.* Это правило позволяет проектировщику выбирать для системы поддержку классов или типов. Типы в основном используются на этапе компиляции для проверки ошибок в типах при присвоении значений атрибутам. Классы используются для хранения и манипулирования схожими объектами во время выполнения. Иначе говоря, класс это более динамичное понятие, а тип — более статичное.
- ❑ *Правило 5. Система должна поддерживать наследование.* Объект должен наследовать свойства своего суперкласса в иерархии. Это свойство обеспечивает возможность многократного использования кода.
- ❑ *Правило 6. Система должна избегать раннего связывания.* Эта особенность позволяет использовать одинаковые названия для методов в разных классах. ОО-система решает, к какой из реализаций необходимо обеспечивать доступ во время выполнения, принимая во внимание класс, которому объект принадлежит. Это правило также известно как правило позднего или динамического связывания.
- ❑ *Правило 7. Система должна быть законченной в вычислительном отношении.* Базовые свойства языков программирования расширяются возможностями языка манипулирования данными (Data Manipulation Language, DML) базы данных, позволяя таким образом выразить на языке программирования любой тип операции.
- ❑ *Правило 8. Система должна быть расширяемой.* Последняя особенность, касающаяся ОО, связана с возможностью определения новых типов. Не существует различий в управлении типами данных, определенными пользователем и системой.
- ❑ *Правило 9. Система должна помнить местонахождение данных.* Традиционные СУБД постоянно хранят информацию на диске, т. е. СУБД обеспечивают устойчивость данных. ОО-системы обычно держат все пространство объектов в памяти; как только система выключается, все пространство объектов теряется. Многие исследователи ООСУБД стараются найти способ непрерывного хранения объектов во вторичной памяти (диски) и извлечения их оттуда.

- ❑ *Правило 10. Система должна обеспечивать возможность управления большими базами данных.* Обычные ОО-системы ограничивают пространство объектов размером доступной оперативной памяти. Например, Smalltalk не может обрабатывать объекты размером более 64 Кбайт. Поэтому важнейшая особенность ООСУБД состоит в оптимизации управления устройствами вторичного хранения информации с помощью буферов, индексов, кластеризации данных и выбора методов определения путей доступа.
- ❑ *Правило 11. Система должна обслуживать нескольких пользователей.* Традиционные СУБД прекрасно с этим справляются. ООСУБД должны обеспечивать такой же уровень параллельной работы, как и обычные системы.
- ❑ *Правило 12. Система должна обеспечивать возможность восстановления после аппаратного или программного сбоя.* ООСУБД должны предлагать тот же уровень защиты от неисправностей оборудования и программного обеспечения, что и традиционные СУБД, т. е. ООСУБД должны обеспечивать поддержку автоматического создания резервных копий и восстановления системы.
- ❑ *Правило 13. Запросы к данным должны быть простыми.* Эффективность запросов — одна из наиболее важных характеристик любой СУБД. Реляционные СУБД предоставляют стандартный способ выполнения запросов к БД с помощью SQL, а ООСУБД должны обеспечить такую же возможность с помощью языка OQL. Группа Object Management Group продвигает спецификацию OQL в качестве стандарта для доступа к данным в объектно-ориентированных базах данных (см. [www.omg.org](http://www.omg.org)).

К необязательным свойствам ООСУБД относятся следующие.

- ❑ *Поддержка множественного наследования.* Множественное наследование приводит к усложнению системы и требует управления конфликтующими свойствами между классами и подклассами.
- ❑ *Поддержка распределенных ООСУБД.* Стремление к интеграции системных приложений является мощным аргументом в пользу распределенных БД. Если необходимо обеспечить корректное взаимодействие ООСУБД с другими системами по сети, то база данных должна обеспечивать возможности распределения.
- ❑ *Поддержка контроля версий.* Контроль версий является новой характеристикой ООСУБД, которая особенно важна в приложениях САПР/САУ (CAD/CAM). Контроль версий позволяет вести историю преобразования объектов. Поэтому мы можем просматривать все состояния объекта, что позволит вернуть его в прежнее состояние и перейти в некоторое последующее.

## 11.7. Влияние объектно-ориентированного подхода на проектирование баз данных

При разработке и внедрении традиционных реляционных баз данных используются техника ER-моделирования, а также нормализация. В процессе проектирования основной упор делается на моделирование объектов реального мира с помощью простых табличных отношений, как правило, приведенных к 3НФ. К сожалению, как уже отмечалось, реляционные и ER-модели иногда не могут адекватно отражать

реальные объекты. Поэтому в ER-модели используются искусственные конструкции типа составных сущностей (мостов), усиливающих семантическое различие между реальным объектом и его представлением.

Вы, наверное, обратили внимание, что представленный в книге процесс проектирования базы данных, в основном, направлен на идентификацию элементов данных, а не на определение операций с данными. Фактически определение ограничений на данные и преобразование данных обычно рассматриваются на более поздних этапах проектирования БД и реализуются во внешнем программном коде. Иначе говоря, операции не являются частью модели БД.

При проектировании объектно-ориентированных БД вопрос противопоставления данных и процедур их обработки решается, и обеспечиваются как идентификация данных, так и определение процедур манипулирования данными. Однако если мы не используем на этапе реализации объектно-ориентированные языки, то даже включение объектно-ориентированных процедур не позволит считать такую систему "объектно-ориентированной".

Почему же традиционные системы допускают, а зачастую и требуют противопоставления данных и процедур? Ведь идея объектно-ориентированного проекта обдумывалась и в классической среде баз данных. Причина простая: проектировщики баз данных просто не имели инструментальных средств, которые могли бы связать воедино данные и процедуры.

Проект объектно-ориентированной БД (ООБД) представляет собой результат применения объектно-ориентированных понятий при проектировании БД. Проектирование ООБД заставляет нас считать данные и процедуры единой сущностью, что позволяет строить базу данных, в которой объекты рассматриваются как отдельные модули. Конкретнее, ОО-проект требует, чтобы в описание базы данных включались объекты, а также их представление данных, ограничения и операции. Такие особенности проектирования, очевидно, дают более полное и значимое описание базы данных, чем это было в традиционных проектах БД.

ОО-проектирование является итеративным и последовательным по своей природе. Проектировщик БД идентифицирует каждый объект реального мира и определяет его внутреннее представление данных, семантические ограничения и операции. Затем проектировщик группирует схожие объекты в классы и реализует ограничения и операции с помощью методов. Здесь проектировщик сталкивается с двумя основными проблемами:

- ❑ построение иерархии класса или сети класса (если допустимо множественное наследование) с помощью базовых типов данных и имеющихся классов. Решение этой задачи определит отношения между классами и подклассами;
- ❑ определение отношений внутри класса (связи атрибут-класс) с помощью базовых типов данных и абстрактных типов данных (АТД).

Важность этой задачи трудно переоценить, поскольку чем правильнее используются иерархия классов и соглашения об отношениях внутри класса, тем более будет гибкой и приближенной к реальности конечная модель.

Возможность повторного использования кода достигается нелегко. Одна из самых трудных задач проектирования ООБД состоит в создании иерархии классов, т. е. создание новых классов на базе существующих. В будущем администраторы БД

должны будут получать специальные навыки для надлежащего выполнения этой задачи и создания кода, описывающего поведение данных. Таким образом, администраторы БД, скорее всего, будут представлять собой программистов баз данных, которые должны определять поведение данных. Роль администраторов БД видимо изменится, когда на них будут возложены задачи программирования и реализации операций над данными.

Как администраторы БД, так и проектировщики сталкиваются и с другими проблемами: в отличие от реляционных или ER-систем в ООБД очень мало инструментальных средств, помогающих в проектировании и реализации базы данных. Проект ООБД можно реализовать в любой модели баз данных. Однако если такой проект предназначен для реализации в любой традиционной БД, он должен разрабатываться с особой тщательностью, поскольку традиционные БД не поддерживают абстрактные типы данных, ненормализованные данные, наследование, инкапсуляцию и другие возможности ОО-модели.

Как и в любой объектно-ориентированной технологии, недостаток стандартизации затрагивает и сферу проектирования ООБД. В настоящее время не существует ни стандартная методология процесса проектирования ООБД, ни критерии (как, например, правила нормализации в реляционной модели) для надлежащей оценки проекта. Однако положение дел постепенно улучшается. В 1989 году отдельные группы практиков объектно-ориентированных технологий сформировали Object Management Group (OMG, рабочая группа по развитию стандартов объектного программирования) для обеспечения взаимодействия различных объектно-ориентированных систем (языки, инфраструктуры, базы данных и т. д.). Группа OMG разрабатывает независимые стандарты и спецификации для объектно-ориентированных систем и компонентов. Группа OMG разработала язык UML (Unified Modeling Language, унифицированный язык моделирования), представляющий собой графический язык для моделирования, проектирования и визуализации объектно-ориентированных систем. Язык UML используется не только для моделирования компонентов баз данных, но и для разработки процессов, модулей, сетевых компонентов и описания взаимодействия между ними. Группой OMG также были созданы стандарты объектов, определяющие Object Management Architecture (OMA, технология управления объектом), которая допускает взаимодействие объектов на различных платформах и системах. Стандарт OMA включает в себя Common Object Request Broker Architecture (CORBA, технология построения распределенных объектных приложений) и Common Object Services Specification (COSS, спецификация на общие средства объектного сервиса). Эта структура используется поставщиками ООСУБД и разработчиками для реализации систем, которые должны взаимодействовать с другими ООСУБД, а также СУРБД и более старыми СУБД.

Некоторые поставщики уже предлагают продукты, соответствующие спецификациям OMG CORBA и COSS, например, System Object Model (SOM) компании IBM и Object Request Broker (ORB) компании Hewlett-Packard. Другие объектные архитектуры также появляются как альтернатива разработкам стандартов, в частности это относится к технологиям OLE (object linking and embedding, технология связывания и внедрения объектов) и Component Object Model (COM, модель компонентных объектов) корпорации Microsoft. Хотя спецификации OLE/COM не являются общепринятым стандартом, широкое распространение на рынке сделало ее стандартом де-факто в среде Microsoft Windows.

## 11.8. ООСУБД: преимущества и недостатки

По сравнению с СУРБД, системам ООСУБД необходимо пройти еще долгий путь, чтобы доля их акций на рынке программного обеспечения достигла двузначных чисел. На самом деле, в настоящее время ООСУБД занимают определенную нишу на рынке, как, например, компьютеры фирмы Apple занимают свою нишу на рынке микрокомпьютеров. Как Макинтош в компьютерной сфере, так и ООСУБД являются двигателем для инноваций в технологиях баз данных. И опять-таки, как и Макинтош, ООСУБД не получила серьезных дивидендов от общего роста рынка высоких технологий. Несмотря на это ООСУБД заслуживают исследования, особенно потому, что их ОО-свойства подталкивают к изменениям в технологии баз данных, которые сегодня представлены на рынке объектно-реляционными СУБД (ОРСУБД).

Проблема рынка ООСУБД состоит еще и в том, что многие возможности ОО-подхода встраиваются в СУРБД при сохранении концептуальной простоты этих моделей, тем самым снижая привлекательность самих ООСУБД. Тем не менее, пока в РСУБД не включены рекомендованные К. Дейтом реализации доменов, ООСУБД обладает достаточными преимуществами, на которые стоит обратить внимание<sup>4</sup>. Большую часть этих достоинств в смысле возможностей управления сложными объектами мы рассмотрели достаточно подробно. Полнота использования всех достоинств ООСУБД зависит от объектно-ориентированного языка программирования. Именно поэтому, указывая на некоторые преимущества ООСУБД, мы ссылаемся на определенные языки программирования.

### Преимущества

- ❑ ООСУБД допускает включение подробной семантической информации в базу данных, что обеспечивает более естественное и реалистичное представление объектов реального мира.
- ❑ ООСУБД хорошо приспособлена для работы со сложными объектами, что делает ее особенно удобной в специализированных приложениях. Традиционные базы данных просто не могут эффективно работать в области CAD/CAM, медицине (обработка изображений), с пространственными изображениями и специализированными мультимедийными приложениями.
- ❑ ООСУБД позволяет расширять базовые типы данных, что позволяет увеличить как функциональные возможности БД, так и возможности моделирования.
- ❑ Если платформа допускает эффективное кэширование, то ООСУБД обеспечивает исключительную производительность по сравнению с реляционными БД при работе со сложными объектами.
- ❑ Контроль версий представляет собой очень важную особенность в специализированных приложениях CAD/CAM, при обработке изображений в медицине, при работе с пространственными изображениями, обработке текстов и в настольных типографских системах.

<sup>4</sup> См. C. J. Date "Back to the Relational Future", [www.dbpd.com/vault/9808date.html](http://www.dbpd.com/vault/9808date.html).

- ❑ Возможность многократного использования классов позволяет ускорить и упростить разработку баз данных и их приложений.
- ❑ Ускоренная разработка приложений за счет наследования и возможности многократного использования. Это преимущество достигается только после проработки таких особенностей ОО, как:
  - надлежащее использование иерархии классов; например, как использовать существующие классы для создания новых классов;
  - методология ОО-проектирования.
- ❑ ООСУБД предоставляет возможное решение проблемы интеграции с существующими и будущими СУБД в единую структуру. Это решение основано на ее сильных возможностях по созданию абстрактных типов данных и многообещающей переносимости.

## Недостатки

- ❑ ООСУБД сталкиваются с сильным и эффективным сопротивлением со стороны устоявшихся СУРБД, особенно в связи с тем, что СУРБД (например, DB2 Universal Database и Oracle 8.0) включили в себя многие ОО-возможности, не дав таким образом ООСУБД шанс выйти вперед в конкурентной борьбе за рынок обработки сложных данных.
- ❑ ООСУБД основаны на объектной модели, которой не хватает солидной теоретической базы, имеющейся у реляционной модели, используемой в СУРБД.
- ❑ В некотором смысле ООСУБД рассматриваются как возврат к устаревшим системам указателей, которые использовались в иерархических и сетевых моделях. Эта критика не связана с тем, что в старых СУБД система указателей связывалась с навигационным стилем манипулирования данными и фиксированными путями доступа, и в сравнении с этим реляционные системы выглядели более привлекательными. Тем не менее, *сложность* системы указателей ООСУБД нельзя отрицать.
- ❑ ООСУБД не имеет стандартного языка нерегламентированных запросов, как реляционные системы. В настоящее время разработка языка OQL еще далека от завершения. В некоторых реализациях ООСУБД имеются расширения реляционного SQL, обеспечивающие возможность интеграции ООСУБД и СУРБД.
- ❑ Реляционные СУБД предоставляют комплексное решение баз данных для бизнеса и управления, поддерживающих обе модели данных и набор четких правил нормализации для проектирования и оценки реляционных БД. ООСУБД пока еще не обладают такими возможностями.
- ❑ Стоимость изучения ООСУБД достаточно высока. Если вы рассмотрите прямые затраты на обучение и время, необходимые для овладения преимуществами объектно-ориентированного подхода, вы поймете, почему ООСУБД редко рассматривают в качестве лучшего варианта при поиске решений для несложных бизнес-задач.
- ❑ Недостаточное предложение ООСУБД на рынке вкупе с высокой стоимостью ее изучения приводит к недостаточному числу специалистов по использованию мощных ОО-технологий. Большинство технологий в настоящее время нацелены

на сферу инженерных приложений программного обеспечения. Поэтому только те компании, у которых есть достаточные ресурсы (деньги, время и квалифицированный персонал), могут позволить себе инвестиции в ОО-технологии.

- ☐ Недостаточная совместимость между различными ООСУБД затрудняет переход с одного программного обеспечения на другое. Продукты с СУРБД очень схожи и переход с одной системы на другую относительно прост.

Несколько лет назад специалисты полагали, что в будущем системы станут управлять объектами со встроенными данными и методами, а не записями, кортежами или файлами. Также считалось, что хотя с переносимостью пока еще не все ясно, все же ОО-подход будет оказывать основное влияние на последующее проектирование и использование БД. Теперь стало понятно, что достижения ООСУБД были не столь впечатляющими из-за успешного внедрения многих ОО-концепций в ОРСУБД в рамках реляционной структуры. И поскольку "Third Manifesto" ("Третий манифест") К. Дейта можно считать предвестником появления третьей волны реляционных баз данных, мы, возможно, вернемся "назад к реляционному будущему". В любом случае ООСУБД оказали сильное влияние на управление базами данных, и битва объектных и реляционных титанов далека от завершения. Наконец, поскольку объектные концепции, скорее всего, останутся в центре внимания будущих разработчиков СУБД, их несомненно стоит внимательно изучать.

## 11.9. Влияние объектно-ориентированных представлений на реляционную модель

Большинство реляционных БД спроектированы для обычных бизнес-приложений, в которых требуются возможность работы с нерегламентированными запросами и простое взаимодействие. Типы данных, встречающиеся в таких приложениях, хорошо известны и легко представляются в табличном виде и в равной степени удобны в небольших и четких транзакциях. Однако СУРБД не так хорошо, как ООСУБД, приспособлены к сложным приложениям CAD/CAM, медицинской сферы, для работы с пространственной графикой, в инженерном проектировании, имитационном моделировании, архитектурном проектировании и в теоретическом моделировании. К тому же СУРБД приближаются к пределу своих способностей в обработке бизнес-информации, где появляется возможность хранения и поиска данных на мультимедийных носителях.

Быстро меняющаяся среда данных вынудила сторонников СУРБД ответить на вызов ОО расширением концептуальных возможностей реляционной модели. Результатом этих усилий принято считать *Extended Relational Model (ERM, расширенная реляционная модель)* или точнее *объектно/реляционную модель (O/RM, или O/PM)*. Хотя разработка этой модели еще продолжается, ее характеристики выглядят так:

- ☐ расширяемость за счет новых определенных пользователем абстрактных типов данных;
- ☐ сложные объекты;
- ☐ наследование;
- ☐ процедуры (правила и триггеры);
- ☐ идентификаторы, генерируемые системой (заменители OID).



Мы не считаем, что представили исчерпывающий список всех новшеств, добавленных к реляционной модели. И при этом мы не уверены, что перечисленные выше дополнения включены во все современные модели ERM. Однако большинство исследователей считают, что этот список представляет наиболее важные и желаемые расширения реляционной модели.

### Примечание

Стоит вновь вспомнить, что "Третий манифест" К. Дейта основан на его предположении, что реляционная модель уже содержит в себе все необходимые возможности при должной поддержке доменов. Поэтому реализация поддержки доменов даст те же преимущества, которые сейчас приписываются объектно-ориентированным расширениям (см. С. J. Date, "Back to the Relational Future: The third manifesto is ready for prime time", апрель 1999, [www.dbpd.com/vault/9808date.html](http://www.dbpd.com/vault/9808date.html)). Однако в настоящий момент реляционные домены не имеют (пока) коммерческой реализации, в то время как ОО-расширения реляционной модели — свершившийся факт.

Стратегия расширения реляционной модели основана на следующих положениях:

- ☐ семантические и объектно-ориентированные принципы необходимы для поддержки нового поколения приложений — особенно если эти приложения развертываются через Интернет;
- ☐ эти принципы могут и должны быть добавлены к реляционной модели;
- ☐ преимущества реляционной модели должны сохраниться для поддержки инвестиций в реляционные технологии и обеспечения обратной совместимости.

Большинство современных расширенных реляционных СУРБД отражают идеи, выраженные К. Дейтом в "Третьем манифесте" (см. *предыдущее примечание*). Они также предоставляют дополнительные и очень важные функции.

- ☐ Корпорации Oracle и IBM разрабатывают набор программных продуктов, названных ими Universal Database Servers (универсальные серверы баз данных). Хотя Universal Database Servers не являются чисто объектно-ориентированными СУБД — им не достает компонента хранения объектов — этот продукт поддерживает сложные типы данных, например, мультимедиа и трехмерные данные, и он обеспечивает возможность работы через Интернет. Интернет позволяет пользователям делать запросы к БД с помощью World Wide Web (WWW, Сеть). Фирма Oracle также разработала СУБД Oracle 9<sup>5</sup>, в которую включена поддержка объектно-ориентированных расширений и хранилищ объектов, причем все это в едином устройстве базы данных. СУРБД DB2 Universal Database Server фирмы IBM имеет схожие характеристики.
- ☐ Система DB2 Universal Database фирмы IBM представляет собой проверенную базу данных, которая используется более чем в 1000 корпораций. Система IBM поддерживает оцифрованную информацию (видео и аудио), а также определенные пользователем типы данных и процедуры. Universal Database также претендует на ключевую роль в области Интернета с поддержкой Web-доступа и интерфейса Java-программирования.

<sup>5</sup> К моменту перевода книги выпущена версия Oracle 11. — *Прим. пер.*

## 11.10. Новое поколение систем управления базой данных

Применение ОО-концепций в отдельных компьютерных областях меняют как саму систему, так и ее поведение. В следующее поколение СУБД, скорее всего, будут включены свойства и объектно-ориентированных систем баз данных, и систем искусственного интеллекта, и экспертных систем, и распределенных БД, и Интернета.

ООСУБД представляют собой только первый шаг на пути к следующему поколению систем баз данных. Применение ОО-концепций в будущем позволит СУБД решать более сложные задачи как с нормализованными, так и с ненормализованными данными. Расширяемость систем баз данных это один из самых весомых вкладов объектно-ориентированной идеологии, который допускает использование новых типов данных, таких как множества, списки, массивы, видео, битовая графика, голос и карты. Новый стандарт SQL3 предоставляет такую возможность расширения, обеспечивая поддержку данных, определенных пользователем в дополнение к предопределенным типам данных (числовым, целым, строковым и т. д.). Например, в SQL3 администратор БД может создавать новые абстрактные типы данных, которые представляют собой наборы объектов, и использовать эти типы данных при определении таблиц. При этом допускается, чтобы в столбцах БД содержались *наборы* значений вместо единственного значения.

История развития рынка СУБД показывает, что ООСУБД будут по-прежнему занимать на рынке свою нишу, связанную с приложениями, в которых требуется обработка больших объемов данных специальных типов с множеством сложных отношений. Например, ООСУБД предпочтительно использовать в системах САПР/САУ (CAD/CAM), компьютеризованном производстве, специальных приложениях мультимедиа, медицинских и архитектурных приложениях, картографии, имитационном моделировании и научных исследованиях.

Однако в современных условиях рынка объектно/реляционные БД являются доминирующими для большинства сложных бизнес-приложений. Это вызвано потребностью обеспечить совместимость новых СУБД с имеющимися системами, фактическим принятием реляционной модели в качестве стандарта, а также явным преимуществом этой модели на рынке СУБД.

## Резюме

Объектно-ориентированные концепции берут свое начало в объектно-ориентированных языках программирования, ставших альтернативой традиционным программным методам. Эти идеи позволяют пользователям с высокой точностью моделировать объекты реального мира и повышают производительность труда программистов.

Объект представляет собой концептуальную модель реального мира, в которую встроены представление данных (атрибуты) и их поведение (методы). Каждый объект имеет уникальный идентификатор, который не зависит от его атрибутов. Любой объект имеет атрибуты (переменные экземпляра) и каждый атрибут может ссылаться

на другой объект. Состояние объекта представляет собой набор значений, которые объект имеет в данный момент времени. Методы реализуют поведение объектов. Методы иницируются с помощью сообщений. Методы и представление данных инкапсулируются для того, чтобы скрыть их от внешних источников.

Схожие объекты группируются в классы. Класс — это коллекция похожих объектов с общей структурой (данными) и поведением (методами). Каждый объект класса представляет собой экземпляр класса или экземпляр объекта. Набор сообщений, которые может получать объект, называется протоколом объекта. Классы организуются в иерархию классов. Объект принадлежит классу в иерархии классов. Объект наследует переменные экземпляра (атрибуты) и методы всех своих суперклассов. Единичное наследование допускает для объекта только один родительский суперкласс. Множественное наследование позволяет объекту иметь несколько родительских суперклассов. Полиморфизм — это свойство, посредством которого различные объекты могут получать одни и те же сообщения различными способами.

Абстрактные типы данных (АТД) реализуются через классы. АТД позволяют конечным пользователям описывать наборы схожих объектов с инкапсулированным представлением данных и их поведением, а также манипулировать ими. Сложный объект формируется из нескольких различных объектов, находящихся в сложных отношениях.

В объектно-ориентированной модели данных (ООМД) проектировщик может создавать более достоверные представления реальных объектов. ООМД поддерживает понятия инкапсуляции, наследования, OID, сложные объекты и расширяемые типы данных. Схема объекта, или пространство объекта, описывает конкретную базу данных. ООМД использует OID для связи между объектами и для навигации по пространству классов и суперклассов объектов. В ООМД нет необходимости использовать оператор JOIN для связывания данных, как этого требует реляционная модель.

ООСУБД обладает рядом преимуществ перед традиционными СУБД. К преимуществам относятся большее семантическое наполнение базы данных, поддержка сложных объектов, более высокая производительность в сложных приложениях, возможность повторного использования классов и расширяемость типов данных в БД. К недостаткам современных ООСУБД следует отнести отсутствие стандартного языка нерегламентированных запросов, сложность изучения и нехватку специалистов.

Сторонники реляционного подхода расширили реляционную модель с целью поддержки некоторых ОО-функций. Расширенная реляционная модель основана на реляционной модели и обеспечивает поддержку сложных объектов, расширяемость новых, определенных пользователем типов данных, наследование, методы и создаваемые системой идентификаторы (заменители OID).

## Основные термины

Абстрактный тип данных (АТД) — abstract data type (ADT)

Ассоциативный объект — associative object

Атрибут — attribute

Базовый тип данных — base data type

Договорные типы данных — conventional data type

Домен — domain

Единичное наследование — single inheritance

Идентификатор объекта — object ID (OID)

Иерархия класса — class hierarchy

Инкапсуляция — encapsulation

Класс — class

Класс пересечения — intersection class

Контроль версий — versioning

Метод — method

Множественное наследование — multiple inheritance

Наследование — inheritance

Объект — object

Объект-набор — collection object

Объектно-ориентированная модель данных (ООМД) — object-oriented data model (OODM)

Объектно-ориентированная система управления базой данных (ООСУБД) — object-oriented database management system (OODBMS)

Объектно-ориентированное программирование (ООП) — object-oriented programming (OOP)

Объектно-ориентированный язык запросов — object query language (OQL)

Объектно-ориентированный язык программирования — object-oriented programming language (OOPL)

Опрос — interrogate

Отношения внутри объекта — interobject relationship

Переменная экземпляра — instance variable

Подкласс — subclass

Позднее связывание — late binding

Полиморфизм — polymorphism

Простой объект — simple object

Пространство объекта, схема объекта — object space, object schema

Протокол — protocol

Раннее связывание — early binding

Расширяемость — extensibility

Сетка класса — class lattice

Сложный объект — complex object

Смешанный объект — hybrid object

Совместное использование адресуемых объектов — referential object sharing

Сообщение — message

Составной объект — composite object

Состояние объекта — object state

Суперкласс — superclass

Экземпляр класса — class instance

Экземпляр объекта — object instance

## Вопросы

1. Расскажите об эволюции объектно-ориентированных представлений. Поясните как эти идеи и понятия повлияли на разработку компьютерных приложений.
2. Как вы определите объектно-ориентированный подход? В чем его преимущества? Как объектно-ориентированные языки программирования связаны с объектно-ориентированным подходом?
3. Определите и опишите:
  - объект;
  - атрибуты;
  - состояние объекта;
  - идентификатор объекта (OID).
4. Определите и сравните понятия метода и сообщения. Какое понятие ОО обеспечивает разницу между методом и сообщением? Приведите пример.
5. Поясните, в чем инкапсуляция отличается от традиционных конструкций программирования (например, определение записи). Какие преимущества дает инкапсуляция? Приведите пример.
6. Проиллюстрируйте на примере понятия класса и экземпляра класса.
7. Что такое протокол класса и как он связан с понятиями метода и класса? Нарисуйте схему, показывающую связь между понятиями ОО: объект, класс, переменные экземпляра, методы, состояние объекта, OID, поведение, протокол и сообщения.
8. Дайте определение понятия иерархии классов, суперкласса и подкласса. Затем поясните на примерах понятия наследования и различные типы наследования.
9. Дайте определение и поясните понятия переопределения методов и полиморфизма. Приведите примеры.
10. Поясните концепцию абстрактных типов данных и то, чем они отличаются от традиционных базовых типов данных. Как соотносятся тип и класс в объектно-ориентированных системах?

11. Проведите краткий исторический обзор моделей данных и их эволюции. Какие свойства моделей данных являются общими? Каковы тенденции развития моделей данных?
12. Что представляют собой 5 минимальных атрибутов ОО-модели? Поясните разницу между ранним и поздним связыванием и их влияние на объектно-ориентированную модель. Приведите примеры.
13. Что такое пространство объекта? С помощью графического представления объектов нарисуйте связь (связи), которая есть между студентами, обучающимися в нескольких группах, и группой, где обучается несколько студентов. Какой тип объекта необходим для описания этой связи?
14. Сравните и сопоставьте ООМД с ER-моделью и реляционной схемой. Как в ООМД представляются слабые сущности? Приведите примеры.
15. Назовите и поясните 13 обязательных правил для ООСУБД.
16. В чем состоят преимущества и недостатки ООСУБД?
17. Поясните, какое влияние ОО-концепции оказывают на проектирование БД. Как инфраструктура ОО влияет на роль администратора БД?
18. В чем основное различие между реляционной моделью и объектной моделью БД?
19. Используя в качестве отправной точки простую систему учета счетов, поясните, чем ее представление в ER-модели отличается от представления в ООМД. (Подсказка: обратитесь к рис. 1.15.)
20. В чем состоит основное различие между РСУБД и ООСУБД?
21. Опишите свойства объектно-реляционной модели.

## Задачи

1. Конвертируйте следующие таблицы реляционной базы данных в эквивалентное ОО-представление. Поясните каждое ваше преобразование с помощью схемы. (Примечание. Модель компании перевозок RRE состоит из трех таблиц, представленных на рис. 11.35.)
2. С помощью таблиц, представленных на рис. 11.35, проделайте следующее:
  - определите необходимые бизнес-правила для связей;
  - руководствуясь вашим опытом, выберите тип участия сущностей в связи (обязательной или необязательной). Поясните ваш выбор;
  - разработайте концептуальную схему объекта.
3. На основе представления данных в задаче 1 разработайте диаграмму пространства объекта, представляющую состояние объекта для экземпляров перевозок, представленных ниже. Снабдите каждый компонент соответствующими OID и именами атрибутов.
  - Экземпляр класса Truck с TRUCK\_NUM = 5001.
  - Экземпляры класса Truck с TRUCK\_NUM = 5003 и 5004.

4. Учитывая данные задачи 1, определите суперкласс Vehicle для класса Truck. Перерисуйте пространство объектов, разработанное вами в задаче 3, принимая во внимание новые суперклассы, которые вы добавили в иерархию классов.

| База данных: PRE_Trucking |           |           |               |                  |             |                   |
|---------------------------|-----------|-----------|---------------|------------------|-------------|-------------------|
| TABLE CODE                | BASE CODE | TYPE CODE | VEHICLE MILES | VEHICLE BUY DATE | VEHICLE VIN |                   |
|                           | 5001      | 101       | 1             | 162123.5         | 08-Nov-99   | AA-322-12212-VW11 |
|                           | 5002      | 102       | 1             | 276984.3         | 23-Mar-97   | AC-342-22134-Q23  |
|                           | 5003      | 101       | 2             | 212346.6         | 27-Dec-98   | AC-445-78656-Z99  |
|                           | 5004      | 101       | 1             | 99894.3          | 21-Feb-99   | VWQ-112-23144-T34 |
|                           | 5005      | 103       | 2             | 245673.1         | 15-Apr-98   | FR-998-32245-VW12 |
|                           | 5006      | 101       | 6             | 293245.7         | 30-Aug-97   | AD-456-00845-R45  |
|                           | 5007      | 102       | 3             | 132012.3         | 01-Dec-98   | AA-341-96573-Z84  |
|                           | 5008      | 102       | 3             | 144213.6         | 21-Sep-98   | DR-559-22189-D33  |
|                           | 5009      | 103       | 2             | 80932.9          | 16-Jan-00   | DE-887-98456-E94  |
|                           | 5010      | 104       | 1             | 34213.4          | 12-Apr-01   | FD-221-21100-F32  |
|                           | 5011      | 101       | 1             | 42326.8          | 09-Nov-00   | DT-324-04056-H22  |
|                           | 5012      | 104       | 6             | 152339.4         | 23-May-99   | GF-657-22134-K48  |
|                           | 5013      | 101       | 3             | 298145.8         | 11-Apr-97   | HR-344-54560-J92  |
|                           | 5014      | 104       | 2             | 8122.2           | 09-Jun-01   | RVV-289-38956-H87 |
|                           | 5015      | 103       | 1             | 154667.9         | 26-Mar-98   | HH-231-55498-K37  |
|                           | 5016      | 105       | 1             | 1200.6           | 23-Jul-01   | FR-332-23459-G55  |
|                           | 5017      | 105       | 1             | 3345.5           | 23-Jul-01   | DD-545-78896-X39  |

| TABLE CODE | BASE CITY       | BASE STATE | BASE AREA CODE | BASE PHONE | BASE MANAGER        |
|------------|-----------------|------------|----------------|------------|---------------------|
|            | 101 Nashville   | TN         | 615            | 123-4567   | Andrea D. Gallagher |
|            | 102 Lexington   | KY         | 606            | 234-5678   | George H. Delarosa  |
|            | 103 Kansas City | MO         | 573            | 345-6789   | Maria J. Talindo    |
|            | 104 Athens      | GA         | 901            | 456-7890   | Peter F. McAfee     |
|            | 105 Gainesville | FL         | 904            | 678-6543   | Lee A. Chau         |

| TABLE CODE | TYPE                          |
|------------|-------------------------------|
|            | 1 Single box, double-axle     |
|            | 2 Single box, single-axle     |
|            | 3 Tandem trailer, single-axle |
|            | 4 Tandem trailer, double-axle |
|            | 5 Utility                     |
|            | 6 Dump truck, single-axle     |
|            | 7 Dump truck, double-axle     |

Рис. 11.35. База данных компании RRE

5. Пусть имеются следующие бизнес-правила:

- на курсе есть несколько групп, но каждая группа входит только в один курс;
- курс ведет один преподаватель, но каждый преподаватель может обучать одну или более различных групп на одном или более курсах;

- в группе может быть несколько студентов, и каждый студент записывается в несколько групп, но каждая группа принадлежит разным курсам (студенты могут обучаться на нескольких курсах, но они не могут обучаться в нескольких группах одного курса!);
- каждая группа привязана к одной аудитории, но каждая аудитория может использоваться для различных групп одного или более курсов;
- преподаватель курирует много студентов, но у студента есть только один куратор.

На базе этих бизнес-правил:

- определите и опишите основные классы объектов;
- модифицируйте ваше описание, сделанное в предыдущем пункте, чтобы включить в него абстрактные типы данных (Name, DOB и Address);
- используйте диаграммы представления объектов для обозначения связей между классами:
  - ◊ Course (курс) и Section (группа)
  - ◊ Section и Professor (преподаватель)
  - ◊ Professor и Students (студенты)
  - ◊ Section (группа) и Students (студенты)
  - ◊ Room (аудитория) и Section (группа)

Какой тип объекта необходим для представления этих связей?

- Используя ОО-представление, определите суперкласс Person для классов STUDENT и PROFESSOR. Опишите новый суперкласс и его связь с подклассами.
6. Конвертируйте следующие таблицы реляционной БД в эквивалентное ОО-представление. Поясните каждое ваше преобразование с помощью схемы. (*Примечание.* База данных компании R&C Stores включает в себя три таблицы, представленные на рис. 11.36).
  7. Конвертируйте следующие таблицы реляционной базы данных в эквивалентное ОО-представление. Поясните каждое ваше преобразование с помощью схемы. (*Примечание.* База данных компании Avion Sales состоит из таблиц, представленных на рис. 11.37).
  8. С помощью ER-диаграммы, представленной на рис. 8.22, создайте эквивалентное ОО-представление.
  9. С помощью ER-диаграммы компании, представленной на рис. 4.12, создайте эквивалентное ОО-представление.



Таблица: REGION

База данных: R&amp;C\_Stores

| REGION_CODE | REGION_LOCATION |
|-------------|-----------------|
| 1           | East            |
| 2           | West            |

Таблица: STORE

| STORE_CODE | STORE_NAME        | STORE_YTD_SALES | REGION_CODE | EMP_CODE |
|------------|-------------------|-----------------|-------------|----------|
| 1          | Access Junction   | \$1,403,455.76  | 2           | 108      |
| 2          | Database Corner   | \$1,821,987.39  | 2           | 112      |
| 3          | Tuple Charge      | \$1,386,783.22  | 1           | 107      |
| 4          | Attribute Alley   | \$1,344,568.56  | 2           | 103      |
| 5          | Primary Key Point | \$3,330,098.45  | 1           | 115      |

Таблица: EMPLOYEE

| EMP_CODE | EMP_TITLE | EMP_LNAME  | EMP_FNAME | EMP_INITIAL | STORE_CODE |
|----------|-----------|------------|-----------|-------------|------------|
| 101      | Mr.       | Williamson | John      | W           | 1          |
| 102      | Ms.       | Ratula     | Nancy     |             | 2          |
| 103      | Ms.       | Greenboro  | Lottie    | R           | 4          |
| 104      | Mrs.      | Rumpersfro | Jennie    | S           | 5          |
| 105      | Mr.       | Smith      | Robert    | L           | 3          |
| 106      | Mr.       | Renselaer  | Cary      | A           | 1          |
| 107      | Mr.       | Ogallio    | Roberto   | S           | 3          |
| 108      | Ms.       | Johnsson   | Elizabeth | I           | 1          |
| 109      | Mr.       | Eindsmar   | Jack      | W           | 2          |
| 110      | Ms.       | Jones      | Rose      | R           | 4          |
| 111      | Mr.       | Broderick  | Tom       |             | 3          |
| 112      | Mr.       | Washington | Alan      | Y           | 2          |
| 113      | Mr.       | Smith      | Robert    | N           | 3          |
| 114      | Mrs.      | Smith      | Sherry    | H           | 4          |
| 115      | Mr.       | Olenko     | Howard    | U           | 5          |
| 116      | Mr.       | Archialo   | Barry     | V           | 5          |
| 117      | Ms.       | Orimaldo   | Jeanine   | K           | 4          |
| 118      | Mr.       | Rosenberg  | Andrew    | D           | 4          |
| 119      | Mr.       | Rosten     | Peter     | F           | 4          |
| 120      | Mr.       | Zack       | Robert    | S           | 1          |
| 121      | Ms.       | Mcbee      | Jennifer  | A           | 1          |
| 122      | Mr.       | Ryan       | Herman    | G           | 3          |

Рис. 11.36. База данных компании R&amp;C Stores

| Таблица: CUSTOMER |              |               |             |          |           |             | База данных: Avion_Sales |         |             |            |
|-------------------|--------------|---------------|-------------|----------|-----------|-------------|--------------------------|---------|-------------|------------|
| CUS_ID            | CUS_LASTNAME | CUS_FIRSTNAME | CUS_INITIAL | CUS_CITY | CUS_STATE | CUS_COUNTRY | CUS_PHONE                | CUS_FAX | CUS_EMAIL   | CUS_CREDIT |
| 10010             | Ramos        | Alfred        | A           |          | 615       |             | 844-2573                 |         | \$2,011.56  | \$5,000.00 |
| 10011             | Dunne        | Leona         | K           |          | 713       |             | 894-1238                 |         | \$4,613.29  | \$5,000.00 |
| 10012             | Smith        | Kathy         | W           |          | 615       |             | 894-2285                 |         | \$3,217.94  | \$5,000.00 |
| 10013             | Olowski      | Paul          | F           |          | 615       |             | 894-2180                 |         | \$10,020.53 | \$5,000.00 |
| 10014             | Orlando      | Myron         |             |          | 615       |             | 222-1672                 |         | \$14,234.85 | \$7,500.00 |
| 10015             | O'Brien      | Amy           | B           |          | 713       |             | 442-3381                 |         | \$3,005.77  | \$5,000.00 |
| 10016             | Brown        | James         | G           |          | 615       |             | 297-1228                 |         | \$5,432.76  | \$5,000.00 |
| 10017             | Williams     | George        |             |          | 615       |             | 290-2556                 |         | \$1,629.85  | \$1,000.00 |
| 10018             | Farriss      | Anne          | G           |          | 713       |             | 382-7185                 |         | \$3,856.31  | \$2,500.00 |
| 10019             | Smith        | Olette        | K           |          | 615       |             | 297-3809                 |         | \$8,938.43  | \$7,500.00 |

Таблица: INVOICE

| INV_NUM | CUS_NUM | EXP_NUM | INV_DATE  | INV_SIB    | INV_AXOS | INV_TOTAL  | INV_TAX    |
|---------|---------|---------|-----------|------------|----------|------------|------------|
| 10001   | 10015   | 103     | 14-Jan-02 | \$2,325.00 | \$186.00 | \$2,511.00 | \$2,511.00 |
| 10002   | 10018   |         |           |            |          |            |            |
| 10003   | 10010   |         |           |            |          |            |            |

| INV_NUM | INVLINE_NUM | PROD_CODE   | INVLINE_UNITS | INVLINE_PRICE | INVLINE_TOTAL |
|---------|-------------|-------------|---------------|---------------|---------------|
| 10001   | 1           | GPS-55AVD   | 1             | \$625.00      | \$625.00      |
| 10001   | 2           | TMA-350D    | 2             | \$850.00      | \$1,700.00    |
| 10002   | 1           | KX-155      | 1             | \$1,599.00    | \$1,599.00    |
| 10003   | 1           | TNL1000(DC) | 1             | \$1,695.00    | \$1,695.00    |
| 10003   | 2           | KMA-24      | 1             | \$599.00      | \$599.00      |
| 10003   | 3           | CP-136M     | 4             | \$980.00      | \$3,920.00    |
| 10003   | 4           | TRT-250D    | 1             | \$1,070.00    | \$1,070.00    |

Таблица: INV\_LINE

Таблица: PRODUCT

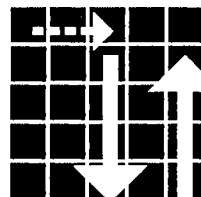
| PROD_CODE    | PROD_COST  | PROD_PRICE | PROD_STOCK | PROD_MIN_STOCK | PROD_LAST_ORDER |
|--------------|------------|------------|------------|----------------|-----------------|
| GPS-55AVD    | \$1,699.00 | \$2,595.00 | 11         | 5              | 20-Feb-02       |
| AIRMAP       | \$619.00   | \$895.00   | 34         | 15             | 18-Aug-01       |
| APOLLO2001   | \$1,529.00 | \$2,245.00 | 17         | 10             | 22-Nov-01       |
| APOLLO360    | \$1,229.00 | \$1,995.00 | 32         | 20             | 30-Dec-01       |
| APOLLO-920   | \$816.00   | \$1,225.00 | 26         | 12             | 27-Jan-02       |
| AT-150       | \$849.00   | \$950.00   | 16         | 10             | 04-Jun-01       |
| CP-136M      | \$605.00   | \$980.00   | 15         | 5              | 06-May-01       |
| EC-10X       | \$1,545.00 |            |            |                |                 |
| FLTPRO       | \$499.00   |            |            |                |                 |
| GPS-150      | \$1,349.00 |            |            |                |                 |
| GPS-155      | \$3,888.00 |            |            |                |                 |
| GPS-55AVD    | \$439.00   |            |            |                |                 |
| GPS-95XL     | \$689.00   |            |            |                |                 |
| KLN-89       | \$2,289.00 |            |            |                |                 |
| KLN-89B      | \$3,899.00 |            |            |                |                 |
| KLN-90B      | \$6,311.00 |            |            |                |                 |
| KMA-24       | \$399.00   |            |            |                |                 |
| KMA-24H      | \$459.00   |            |            |                |                 |
| KN-62A       | \$1,344.00 |            |            |                |                 |
| KX-155       | \$1,119.00 | \$1,599.00 | 26         | 12             | 23-Oct-01       |
| KX-165       | \$1,299.00 | \$1,695.00 | 18         | 10             | 23-Oct-01       |
| KY-196A/197A | \$659.00   | \$999.00   | 12         | 5              | 20-May-01       |
| SKYNAV5000   | \$1,249.00 | \$1,799.00 | 18         | 10             | 01-Dec-01       |
| TMA-350D     | \$499.00   | \$850.00   | 15         | 10             | 28-Dec-01       |
| TNL1000(DC)  | \$1,250.00 | \$1,695.00 | 35         | 10             | 21-Dec-01       |
| TNL-2000     | \$3,999.00 | \$6,650.00 | 12         | 5              | 20-Feb-02       |
| TNL-2000A    | \$1,999.00 | \$2,685.00 | 27         | 10             | 22-Dec-01       |
| TRT-250D     | \$693.00   | \$1,070.00 | 19         | 10             | 07-Jan-02       |

| EMPLOYEE | EMPLOYEE | EMPLOYEE  | EMPLOYEE | EMPLOYEE | EMPLOYEE | EMPLOYEE | EMPLOYEE  |
|----------|----------|-----------|----------|----------|----------|----------|-----------|
| 100      | Mr.      | Kolmycz   | Andrew   | H        |          |          | 15-Mar-85 |
| 101      | Dr.      | Lewis     | Barbara  | K        |          |          | 25-Apr-86 |
| 102      | Ms.      | Vandam    | Marie    | L        |          |          | 20-Dec-90 |
| 103      | Mr.      | Jones     | Robert   |          |          |          | 28-Aug-94 |
| 104      | Dr.      | Lange     | William  | B        |          |          | 20-Oct-94 |
| 105      | Mr.      | Williams  | George   | O        |          |          | 08-Nov-94 |
| 106      | Mrs.     | Duzak     | Judith   | E        |          |          | 05-Jan-89 |
| 107      | Ms.      | Dante     | Melanie  | K        |          |          | 02-Jul-94 |
| 108      | Ms.      | Wesenbach | Roberta  | F        |          |          | 18-Nov-92 |
| 109      | Mr.      | Travis    | Peter    |          |          |          | 14-Apr-89 |
| 110      | Mr.      | Genkazi   | George   | R        |          |          | 01-Dec-90 |

Таблица  
EMPLOYEE

Рис. 11.37. База данных компании Avion Sales

## Глава 12



# Клиент/серверные системы

В этой главе мы будем изучать:

- что собой представляет работа в системе "клиент/сервер";
- преимущества использования клиент/серверных систем;
- историю развития клиент/серверных систем;
- компоненты клиент/серверных систем и их взаимодействие;
- влияние клиент/серверных систем на СУБД;
- способы внедрения клиент/серверных систем на предприятии;
- факторы, влияющие на реализацию клиент/серверных систем.

## Обзор

Даже беглый обзор научной и профессиональной литературы показывает, что о вычислениях в клиент/серверных средах больше всего писали начиная с середины 1990-х годов. К началу нового тысячелетия клиент/серверные системы перестали быть чем-то необычным, они стали повседневной реальностью. Интернет и его производные *интранет* (технология создания корпоративной локальной сети повышенной надежности с ограниченным доступом, аналогичной Интернету) и *экстра-нет* (объединение корпоративных сетей различных компаний, взаимодействующих друг с другом через Интернет) — это, возможно, наиболее яркий пример клиент/серверных систем, и они явно позиционируют себя как основные платформы для разработки приложений. Учитывая широкое распространение и доступность Интернета (кто из нас не входил в Сеть?), необходимо иметь представление о вычислениях в клиент/серверных средах, об основных компонентах этой среды, их взаимодействии и влиянии, которое оказывают клиент/серверные приложения на проектирование, реализацию и управление базами данных. До известной степени эта глава представляет собой основу для изучения *гл. 14 и 15*.

### 12.1. Что такое клиент/серверные вычисления?

Когда профессионалы менеджмента говорят о клиент/серверных вычислениях, они обычно имеют в виду сравнительно новые технологии, позволяющие решить многие

проблемы управления данными, с которыми сталкиваются компании. Не удивительно, что руководители быстро пришли к выводу, что клиент/серверные вычисления открывают новые возможности в мире, которым правит информация. Конечные пользователи привыкли считать, что программное обеспечение, основанное на графическом интерфейсе пользователя (GUI, Graphical User Interface), обеспечивает им совместимость с различными источниками данных практически на всех платформах. Иначе говоря, вы очень скоро обнаружите, что все ваши задумки в сильной степени зависят от людей, с которыми вы общаетесь, и от рода работы, которую они выполняют. Недостаток взаимопонимания или даже общего словаря — это одна из проблем, с которыми мы сталкиваемся при работе с новыми технологиями.

Как изучать клиент/серверную методологию? Правильнее всего начать изучение с определения основных компонентов и функций клиент/серверных моделей.

Термин "клиент/сервер" используется при разработке компьютеризованных систем для описания вычислительной модели. Эта модель основана на распределении функций между двумя типами независимых и автономных процессов: серверами и клиентами. *Клиент* — это любой процесс, который запрашивает определенные ресурсы или сервисы от других (серверных) процессов. *Сервер* — это процесс, который предоставляет необходимые сервисы (услуги) другому процессу (клиенту). Процессы клиента и сервера могут находиться на одном и том же компьютере или же на разных компьютерах, подключенных к сети.

Когда процессы клиента и сервера находятся на двух или более независимых компьютерах сети, сервер может предоставлять сервисы для более чем одного клиента. Кроме того, клиент может запрашивать сервисы от нескольких серверов сети независимо от их расположения или физических характеристик компьютера, на котором находится процесс сервера. Сеть связывает воедино серверы и клиенты, предоставляя клиентам и серверам средства связи (рис. 12.1). На рис. 12.1 видно, что сервисы могут предоставляться различными сетевыми компьютерами. Например, один компьютер может предоставлять сервисы по управлению файлами и принтерами, другой — сервисы связи и факсимильных сообщений, некоторые могут использоваться в качестве Web-серверов или серверов баз данных и т. д.

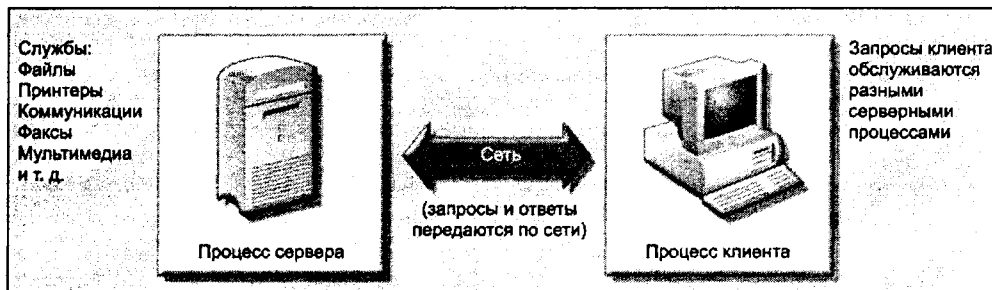


Рис. 12.1. Базовая модель клиент/серверных вычислений

Секрет успеха клиент/серверных приложений в том, где выполняется обработка запросов. Если, например, в клиент/серверной базе данных клиент запрашивает дан-

ные с сервера базы данных, то фактическая обработка запроса (выбор записей) осуществляется на компьютере сервера базы данных. Другими словами, сервер выбирает записи, соответствующие заданным критериям, и отсылает их по сети клиенту. Обработку информации можно распределить даже по компьютерам (серверам) нескольким типам, например, ПК на базе процессора Pentium, PowerPC, рабочим станциям на базе RISC-процессоров<sup>1</sup>, мини-компьютерам или мэйнфреймам.

Уровень распределения задач обработки данных — главное отличие клиент/серверных систем от систем с мэйнфреймом. В системах с мэйнфреймом вся обработка ведется на мэйнфрейме, а терминал (обычно неинтеллектуальный) используется только для отображения информации на экране. Такую инфраструктуру нельзя считать независимой — терминал является попросту придатком главной машины. В отличие от этого в клиент/серверной модели предлагается четкое разделение процессов клиента и сервера, которые не зависят друг от друга. Серверы и клиенты находятся между собой в связи "многие-ко-многим" (M:N), когда один сервер может предоставлять сервисы многим клиентам, а один клиент может запрашивать сервисы от многих серверов.

В зависимости от степени разделения процессов между клиентом и сервером, сервер и клиент считаются либо сильным, либо слабым. *Слабый ("тонкий") клиент* (thin client) выполняет минимум обработки на стороне клиента, в то время как *сильный ("толстый") клиент* (fat client) берет на себя относительно большую часть обработки данных. Соответственно, *сильный ("толстый") сервер* (fat server) несет основную нагрузку по обработке данных, в то время как нагрузка на *слабый ("тонкий") сервер* (thin server) относительно невелика. Поэтому, как правило, слабые клиенты связаны с сильными серверами и соответственно сильные клиенты связаны со слабыми серверами.

С точки зрения разделения нагрузки по обработке данных между сервером и клиентом, система с мэйнфреймом представляет собой пример максимально сильного сервера и максимально слабого клиента, т. к. все процессы обработки данных выполняются на серверной стороне (главная машина), а на клиентской стороне (неинтеллектуальные терминалы) никакая обработка данных не ведется. Поскольку работа в Интернете хорошо вписывается в модель с мэйнфреймом (сильный сервер/слабый клиент), можно считать, что, по крайней мере, в этом отношении Интернет возвращает нас именно к такой модели. (Во всяком случае, широкое проникновение Интернета в современные бизнес-системы возродило интерес к клиент/серверной вычислительной модели. Для получения более полной информации по базам данных в Интернете см. гл. 14 и 15.)

Наконец, клиент/серверные системы можно подразделить на двухзвенные (2-tier) или трехзвенные (3-tier). В *двухзвенных клиент/серверных системах* клиент запрашивает сервисы непосредственно от сервера. В *трехзвенных клиент/серверных системах* клиентские запросы обрабатываются промежуточными серверами, которые координируют выполнение клиентских запросов с подчиненными им серверами.

Основные свойства клиент/серверной модели здесь были представлены лишь схематически. Чтобы понять, почему клиент/серверные приложения играют столь важную

---

<sup>1</sup> Reduced Instruction Set Computer (сокращенный набор команд) — тип архитектуры процессора, ориентированной на быстрое и эффективное выполнение относительно небольшого набора встроенных команд. — *Прим. пер.*

роль в современной вычислительной среде, мы должны изучить причины возникновения этой модели, ее развитие, архитектуру и функциональные возможности.

## 12.2. Причины использования клиент/серверных систем

Клиент/серверные вычисления играют столь важную роль при разработке систем, поскольку они помогают компаниям выживать в условиях глобальной конкуренции. Быстро изменяющаяся инфраструктура бизнеса выдвигает требование обеспечения широкого доступа к данным в масштабах предприятия, что в свою очередь должно резко повысить производительность конечных пользователей. Требование широкого доступа к данным стало причиной возникновения инфраструктуры, где компьютеры формируют единую систему, называемую *средой распределенной обработки данных*, *кооперативной вычислительной средой* и т. п. Поэтому в известном смысле клиент/серверный подход не является чем-то новым. Однако все преимущества и эффективность клиент/серверной модели стали очевидны именно вследствие бурного развития компьютерных технологий, сделавших платформы персональных компьютеров столь же надежными, как и традиционные системы с мэйнфреймами. На самом деле, ускоренное развитие систем на основе интернет-технологий, особенно таких, которые могли работать в глобальной Сети (Web), сделало клиент/серверную модель значительно более доступной. Например, чтобы сохранить конкурентоспособность в глобальном бизнесе при проведении маркетинговых операций и обслуживании клиентов, предприниматели все больше попадают в зависимость от глобальной Сети. Электронная коммерция (или е-коммерция), основанная на использовании глобальной Сети, стала нормой для предпринимателей всех уровней. Влияние е-коммерции на инфраструктуру БД мы рассмотрим в гл. 14.

Ниже перечислены основные факторы, влияющие на клиент/серверную архитектуру.

- ☐ Изменение в инфраструктуре бизнеса.
- ☐ Возросшие требования доступа к данным предприятия.
- ☐ Необходимость повышения производительности конечных пользователей на базе эффективного использования информации.
- ☐ Развитие компьютерных технологий, обеспечивающих эффективность использования клиент/серверных моделей.

### **Дополнительная информация**

Более подробные сведения по каждому из этих факторов можно найти на сайте [www.course.com](http://www.course.com) (ищите по ISDN этой книги 0-619-06268-X).

## 12.3. История развития клиент/серверных информационных систем

Современные компьютерные системы, как правило, основаны на сети, объединяющей большое число разнотипных компьютеров. Персональный компьютер становится

общедоступной станцией для конечных пользователей корпоративной сети и местом доступа к базе данных всего предприятия. Современные приложения позволяют пользователям получать прямой доступ к информации независимо от ее местонахождения, используемых моделей или характеристик других компьютеров сети. Сеть позволяет каждому конечному пользователю использовать информацию совместно с другими пользователями, получать доступ к ресурсам главного компьютера, внешним информационным ресурсам.

Сегодняшние пользователи для анализа информации и принятия решений требуют предоставить им возможность обеспечения доступа к информации по запросу. И, что более важно, они увеличивают свою производительность, перестраивая основные бизнес-операции внутри предприятия. Требование к разделению ресурсов в сильно связанной компьютерной инфраструктуре подготовило почву для возникновения клиент/серверных систем. На рис. 12.2, в качестве итога всех предыдущих рассуждений, представлены четыре стадии развития информационных систем от системы с мэйнфреймом до инфраструктуры клиент/серверной модели на базе персональных компьютеров.

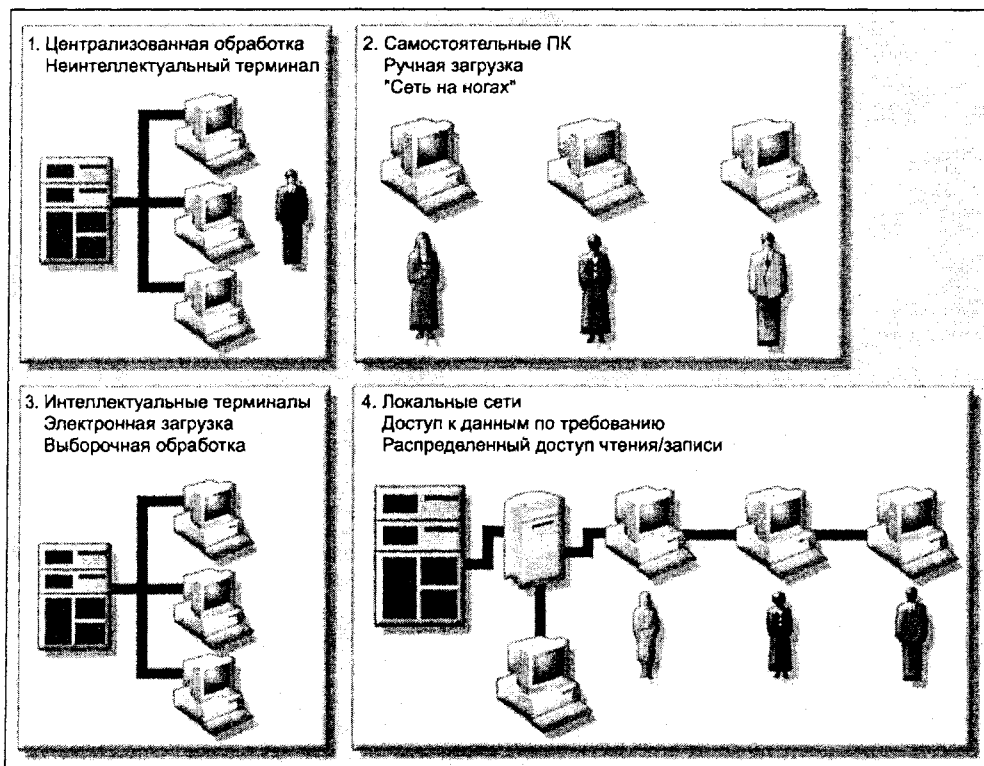


Рис. 12.2. История развития компьютерной инфраструктуры

В настоящее время сетевые системы выходят за рамки отдельных предприятий. Интернет создал Сеть из сетей, упростившую в значительной степени свободный обмен разнообразной информацией во всех сферах деятельности.

### **Дополнительная информация**

Дополнительную информацию можно найти на сайте [www.course.com](http://www.course.com) (ищите 0-619-06269-X) для детального изучения развития клиент/серверных систем.

Эволюция от универсальных машин до клиент/серверных информационных систем на базе персональных компьютеров произвела радикальные изменения в некоторых ключевых подходах к управлению информацией. Некоторые из этих изменений представлены в табл. 12.1.

**Таблица 12.1.** Сравнение информационных систем на базе мэйнфрейма и клиент/серверных систем на базе персональных компьютеров

| <b>Компонент</b>                 | <b>Информационная система на базе мэйнфрейма</b> | <b>Клиент/серверная информационная система на базе персональных компьютеров</b> |
|----------------------------------|--------------------------------------------------|---------------------------------------------------------------------------------|
| Управление                       | Централизованное                                 | Распределенная/децентрализованная                                               |
| Поставщик                        | Единственный поставщик                           | Несколько поставщиков                                                           |
| Оборудование                     | Частное                                          | Несколько поставщиков                                                           |
| Программное обеспечение          | Частное                                          | Несколько поставщиков                                                           |
| Безопасность                     | Высокоцентрализованная                           | Децентрализованная                                                              |
| Манипулирование данными          | Очень ограниченные возможности                   | Гибкая настройка                                                                |
| Управление системой              | Интегрированное                                  | Ограниченные возможности                                                        |
| Разработка приложений            | Чрезмерная структурированность                   | Гибкость                                                                        |
|                                  | Большие временные затраты                        | Среда быстрой разработки приложений                                             |
|                                  | Ведение журналы незавершенных приложений         | Высокоэффективные инструментальные средства                                     |
| Платформа конечного пользователя | Неинтеллектуальный терминал                      | Интеллектуальный персональный компьютер                                         |
|                                  | Символьный интерфейс                             | Графический интерфейс пользователя (GUI)                                        |
|                                  | Однозадачность                                   | Многозадачная операционная система                                              |
|                                  | Ограниченная производительность                  | Высокоэффективные инструментальные средства                                     |



## 12.4. Что ожидает руководство от клиент/серверных систем

Клиент/серверные вычисления получили столь широкое распространение, потому что они обладают потенциальной возможностью более эффективно использовать ресурсы оборудования и программного обеспечения. Говоря кратко, эффективное использование ресурсов — это главное достоинство этих систем. Такая производительность определяется следующими тремя особенностями клиент/серверных систем.

- Нацеленность на разработку систем, не зависящих от оборудования и программной платформы.
- Оптимизация распределения обработки данных на различных компьютерных платформах с использованием лучших особенностей каждой платформы.
- Использование технологий, методик и специализированных инструментальных средств для разработки дружественных пользователю, недорогих систем, имеющих возможность взаимодействовать с различными платформами и оборудованием.

Кроме перечисленных основных преимуществ клиент/серверного подхода, необходимо высказать и следующие замечания:

- хотя клиент/серверные вычисления позволяют разрешить проблемы управления информацией, это не является основной целью подобных систем. Эти системы лишь средство, с помощью которого можно достичь эффективного управления информацией;
- клиент/серверные вычисления не решают *все* проблемы управления данными. Например, ни плохой проект БД, ни плохо написанные приложения не станут лучше работать в клиент/серверной инфраструктуре;
- клиент/серверные вычисления основаны на очень сложной технологии, которая сама по себе порождает проблемы управления. Если в этой технологии не разобраться как следует и не организовать ее управление должным образом, то вряд ли можно ожидать от нее каких-то выгод. Фактически при неправильном планировании и реализации некоторые клиент/серверные системы могут привести даже к неблагоприятным результатам, например, к задержке выполнения проекта, вызванной сложностью системы, технической несовместимости, перегрузке сети и проблемам с производительностью системы в целом.

Не стоит удивляться тому, что при таком изобилии описаний клиент/серверных систем в компьютерных журналах, рекламах разработчиков и на конференциях, надежды руководства на перспективы использования таких систем были очень радужными. Однако нужно помнить, что достоинства клиент/серверной модели часто зависят от рода деятельности предприятия и способа реализации системы. И особенно нужно иметь в виду, что достоинства клиент/серверных вычислений в большой степени зависят от того подразделения предприятия, которое эксплуатирует эту систему. Отдел MIS (Management Information System department, отдел информационных систем), в обязанности которого входит управление клиент/серверной инфраструктурой, скорее всего, будет оценивать преимущества этой системы иначе, нежели подразделения, которые пользуются услугами системы незаметно для себя. Далее мы перечислим организационные достоинства клиент/серверных систем с двух разных точек зрения.

### 12.4.1. Что ожидает от клиент/серверных вычислений отдел MIS

Руководство отдела MIS обычно оценивает достоинства клиент/серверных систем по следующим шести показателям:

- ☐ сокращение стоимости разработки и реализации;
- ☐ уменьшение времени на разработку и повышение производительности труда программистов;
- ☐ расширение жизненного цикла системы за счет масштабируемости и переносимости;
- ☐ уменьшение стоимости эксплуатации системы;
- ☐ передача части обязанностей отдела MIS от разработчиков конечным пользователям;
- ☐ улучшение размещения информации.

#### ***Дополнительная информация***

Дополнительный материал по этой теме можно найти на сайте [www.course.com](http://www.course.com) (поиск по 0-619-06269-X).

### 12.4.2. Что ожидает от клиент/серверных вычислений предприятие

В то время как менеджеры отдела MIS, как правило, обращают внимание на технические аспекты реализации и функционирования клиент/серверной инфраструктуры, руководители других подразделений основное внимание уделяют принятию тактических и стратегических решений. Эти руководители оценивают достоинства клиент/серверных вычислений с учетом следующих факторов.

- ☐ Гибкость и адаптивность.
- ☐ Повышение производительности труда сотрудников.
- ☐ Улучшение производственного процесса компании и методов перестройки бизнес-операций.
- ☐ Новые возможности по повышению конкурентоспособности.
- ☐ Повышение качества обслуживания клиентов.

#### ***Дополнительная информация***

Дополнительный материал по этой теме можно найти на сайте [www.course.com](http://www.course.com) (поиск по ISBN этой книги 0-619-06269-X).

## 12.5. Архитектура "клиент/сервер"

Клиент/серверная инфраструктура, или архитектура "клиент/сервер", обеспечивает необходимые предпосылки для нормального развертывания клиент/серверных сис-

тем. Архитектура "клиент/сервер" базируется на компонентах оборудования и программного обеспечения, которые, взаимодействуя друг с другом, формируют систему. Система "клиент/сервер" включает в себя три основных компонента: клиенты, серверы и промежуточное программное обеспечение передачи данных, ППО (middleware).

- *Клиент* представляет собой любой процесс компьютера, который запрашивает сервис от сервера. Клиент также называется еще *интерфейсным приложением* (*front-end application*), что отражает факт взаимодействия конечного пользователя с клиентским процессом.
- *Сервер* — это любой компьютерный процесс, предоставляющий сервис клиентам. Сервер также называют *серверным приложением* (*back-end application*), что отражает факт предоставления сервером сервиса клиентскому процессу.
- *Промежуточное программное обеспечение передачи данных, ППО (communications middleware)* представляет собой любой компьютерный процесс (процессы), посредством которого клиенты и серверы взаимодействуют друг с другом. Это программное обеспечение, которое еще называют *уровнем коммуникаций* (*communications layer*), состоит из нескольких уровней программного обеспечения, помогающих передавать данные и управляющую информацию между клиентами и серверами. Промежуточное программное обеспечение (ППО) обычно привязано к сети. Все клиентские запросы и ответы сервера передаются по сети в форме сообщений, в которых содержатся управляющая информация и данные.

### 12.5.1. Как взаимодействуют компоненты

Для иллюстрации взаимодействия трех перечисленных компонентов посмотрим, как запросы клиента обслуживаются сервером баз данных. Взгляните на рис. 12.3 и обратите внимание, что обработка приложения разделяется на два главных независимых процесса: клиентский и серверный. ППО позволяет взаимодействовать процессам клиента и сервера. На рис. 12.3 обратите внимание на то, что ППО представляет собой своего рода опорную платформу для клиентов и серверов. Именно промежуточное программное обеспечение, являясь ключевым компонентом системы, определяет все затраты на разработку, слабые места системы и сложность ее реализации.



Рис. 12.3. Взаимодействие клиента и сервера

На рис. 12.3, например, процесс-клиент отвечает за интерфейс конечного пользователя, проверку локальных данных, логику обработки и представление данных. ППО

гарантирует, что сообщения между клиентами и серверами будут правильно маршрутизироваться и доставляться по нужному адресу. SQL-запросы обрабатываются сервером баз данных, который проверяет запросы, выполняет их и посылает результаты клиентам.

Как уже отмечалось, совсем необязательно, чтобы сервер и клиент находились на разных компьютерах. Они могут размещаться на одном и том же компьютере и совместно использовать один и тот же процессор, если операционная система позволяет это делать, т. е. если используется многозадачная операционная система. Однако в большинстве реализаций клиент/серверных систем процессы клиента и сервера размещаются на разных машинах. На рис. 12.4 представлена клиент/серверная система с двумя серверами и тремя клиентами.

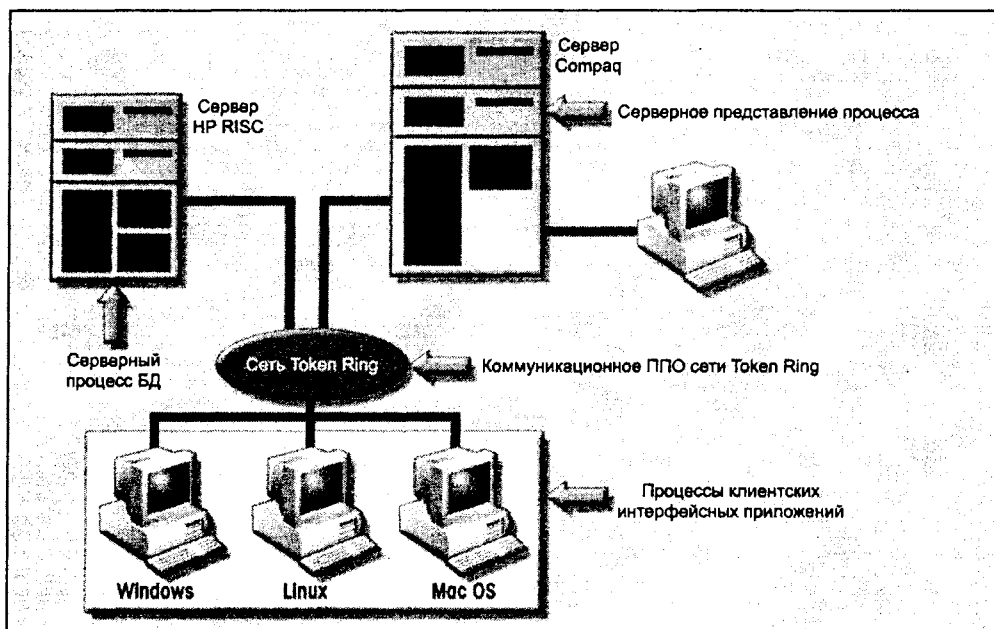


Рис. 12.4. Пример архитектуры "клиент/сервер"

В инфраструктуре, представленной на рис. 12.4, сервер баз данных выполняется на RISC-компьютере, в то время как процесс сервера выполняется на компьютере фирмы Compaq. Три клиентских процесса выполняются под управлением трех различных операционных систем: Windows 98, Open Linux и Apple Mac OS. Процессы клиента и сервера соединяются посредством сети Token Ring (кольцевая сеть с маркерным доступом). Интерфейсные прикладные программы на клиентских компьютерах запрашивают данные и изображения от серверных процессов (сервер баз данных и сервер изображений). Сетевое и инструментальное программное обеспечение формирует промежуточное программное обеспечение (ППО), посредством которого взаимодействуют между собой клиенты и серверы. Обратите внимание, что комму-

никации имеют место не только между клиентами и серверами, но и между серверами. Вспомните (см. гл. 10), что такой сценарий типичен для распределенных баз данных, где запрашиваемые данные могут храниться в разных местах.

Рис. 12.4 иллюстрирует сложную но, тем не менее, обычную клиент/серверную инфраструктуру, в которой процессы сервера выполняются под управлением двух различных операционных систем, клиентские процессы выполняются под управлением трех различных операционных систем и в системе имеются три различных аппаратные платформы. В этом случае ППО (сетевое и инструментальное программное обеспечение) становится объединяющей платформой для всех компонентов. В следующем разделе мы рассмотрим компоненты ППО более подробно. Но прежде следует ознакомиться с правилами, по которым взаимодействуют клиент/серверные компоненты.

### 12.5.2. Правила архитектуры "клиент/сервер"

Для корректного взаимодействия компонентов клиент/серверной архитектуры между собой требуется их соответствие некоторым основным правилам. Эти правила должны в равной степени выполнять и клиенты, и серверы, и промежуточное программное обеспечение. Хотя представленный ниже список не претендует на полноту, приведенные здесь правила архитектуры "клиент/сервер" составляют основу современного поколения клиент/серверных систем.

- ☐ Независимость от оборудования.
- ☐ Независимость от программного обеспечения:
  - операционной системы;
  - сетевой системы;
  - приложений.
- ☐ Открытый доступ к сервисам.
- ☐ Распределение процессов:
  - автономность процессов;
  - максимальное использование локальных ресурсов;
  - масштабируемость и гибкость;
  - способность к взаимодействию и интегрируемость.
- ☐ Стандартизация.

*Независимость от оборудования.* Правило независимости от оборудования требует, чтобы процессы клиента, сервера и ППО могли выполняться на различных аппаратных платформах (IBM, DEC, Apple и т. д.) без какого-либо изменения функциональных возможностей.

*Независимость от программного обеспечения.* Правило независимости от программного обеспечения требует, чтобы процессы клиента, сервера и ППО поддерживали несколько операционных систем (например, Windows 98, Windows NT, OS/2, Linux и UNIX), различные сетевые протоколы (например, IPX, TCP/IP) и различные приложения (таблицы, базы данных, электронную почту и т. д.).

*Открытый доступ к сервисам.* Все клиенты в системе должны иметь открытый (неограниченный) доступ ко всем сервисам, предоставляемым внутри сети, и эти сервисы не должны зависеть от расположения клиента или сервера. Задача состоит в том, что сервисы должны предоставляться по запросу клиента. Фактически обеспечение сервисами по запросу — одна из основных задач клиент/серверной компьютерной модели.

*Распределение процессов.* Главное свойство клиент/серверных систем состоит в том, что обработка информации распределяется между клиентами и серверами. Распределение нагрузки по обработке данных должно соответствовать следующим правилам.

- Процессы клиента и сервера должны быть автономными с четко определенными границами и функциями. Это свойство позволяет выявить функциональные возможности каждой стороны и усилить модульность и гибкость системы.
- Локальное использование ресурсов (как со стороны клиента, так и со стороны сервера) должно быть максимальным. Процессы клиента и сервера должны максимально использовать мощности хост-компьютеров. Это свойство позволяет системе назначать определенную функциональность компьютеру, который лучше всего может с этим справиться. Иначе говоря, для максимального использования всех ресурсов процесс сервера должен совместно использоваться всеми клиентскими процессами, т. е. процесс сервера должен уметь обслуживать несколько запросов от нескольких клиентов.
- Масштабируемость и гибкость требуют, чтобы процессы клиента и сервера можно было легко обновить для выполнения на более мощной аппаратной и программной платформе. Это свойство расширяет функциональные возможности клиент/серверных процессов, если требуется обеспечить дополнительные возможности или улучшить производительность.
- Способность к взаимодействию и интеграция требуют, чтобы процессы клиента и сервера *легко* интегрировались между собой при формировании системы. Обмен данными серверного процесса должен быть прозрачным для клиентского процесса.

*Стандартизация.* Наконец, все правила клиент/серверной архитектуры должны быть основаны на стандартах. Например, стандартными должны быть пользовательский интерфейс, сетевые протоколы, взаимный обмен между процессами и т. д. Стандарты гарантируют, что для достижения желаемого результата все компоненты будут взаимодействовать упорядоченно.

По-настоящему универсальных стандартов для всех компонентов не существует. В действительности имеется множество различных стандартов, из которых можно сделать выбор. Например, приложение для доступа к данным может использовать Open Database Connectivity (ODBC, открытый интерфейс доступа к базам данных) вместо Integrated Database Application Programming Interface (IDAPI). (ODBC и IDAPI являются компонентами промежуточного программного обеспечения баз данных, обеспечивающими стандартный доступ к данным для нескольких процессов в системе). Или приложения могут использовать в качестве сетевого протокола Internetwork Packet Exchange (IPX) вместо Transmission Control Protocol/Internet Protocol (TCP/IP). То, что приложение не использует единый стандарт, не означает, что оно не может выступать в роли клиент/серверного приложения. Вопрос состоит в

том, чтобы гарантировать взаимодействие всех компонентов (сервер, клиенты и ППО) при использовании ими одних и тех же стандартов. Что действительно характеризует клиент/серверные вычисления, так это то, что разделение прикладных процессов не зависит от используемых сетевых протоколов.

### 12.5.3. Компоненты клиента

Как отмечалось ранее, клиент — это любой процесс, который запрашивает сервисы от процесса сервера. Клиент — это активный процесс, и потому он всегда инициирует связь с сервером. Клиент включает в себя аппаратные и программные компоненты. Желательно, чтобы в состав клиента были включены следующие компоненты:

- ☐ мощное оборудование;
- ☐ операционная система с возможностью многозадачной обработки информации;
- ☐ графический интерфейс пользователя (GUI);
- ☐ коммуникационные возможности.

Поскольку процессы клиента требуют значительных аппаратных ресурсов, они должны размещаться на компьютере, обладающем достаточной мощностью, например, на рабочей станции на базе процессоров Pentium или RISC. Такие мощности облегчают создание систем с мультимедийными возможностями. Мультимедийные системы обрабатывают множество типов данных, таких как звук, изображение, видео и т. д. Клиентские процессы, ко всему прочему, требуют много места на диске и большого объема физической памяти. Сколько необходимо ресурсов? Чем больше оперативной памяти и дискового пространства — тем лучше.

Клиент должен иметь доступ к операционной системе, по крайней мере, к некоторым ее многозадачным возможностям. В настоящее время наиболее распространенными клиентскими платформами являются Microsoft Windows 95/98<sup>2</sup>. Эти операционные системы обеспечивают доступ к памяти, обладают всеми достоинствами многозадачных систем и имеют развитый графический интерфейс пользователя. Все эти достоинства вкупе с изобилием приложений, разработанных в среде Windows-интерфейса, делают Windows-платформу предпочтительной для большинства клиент/серверных реализаций. Другие операционные системы (Microsoft Windows NT, Windows 2000 Server, OS/2 фирмы IBM и множество разновидностей UNIX, включая Linux<sup>3</sup>) столь же популярны при выполнении обработки информации на стороне сервера.

Для эффективной работы в клиент/серверной инфраструктуре компьютер клиента должен иметь возможность взаимодействия с другими компьютерами в сети. Поэтому связка "оборудование плюс операционная система" должна также обеспечивать возможность связи с несколькими сетевыми операционными системами. Причина этого очевидна: сервисы могут быть расположены в различных компьютерных сетях.

---

<sup>2</sup> К клиентским системам можно отнести и недавно появившуюся операционную систему Windows XP (Professional или Home Edition). — *Прим. пер.*

<sup>3</sup> Сюда можно отнести также новые операционные системы Windows XP Professional и Windows Server 2003. — *Прим. пер.*

Клиентское, или интерфейсное, приложение функционирует поверх операционной системы и взаимодействует с коммуникационным промежуточным программным обеспечением (ППО) для получения доступа ко всем доступным сервисам сети. Для создания интерфейсных прикладных программ используются языки программирования третьего и четвертого поколений (3GL и 4GL). Большинство таких прикладных программ используют графический интерфейс пользователя с тем, чтобы скрыть сложность клиент/серверных компонентов от конечного пользователя. На рис. 12.5 представлены основные компоненты клиента.

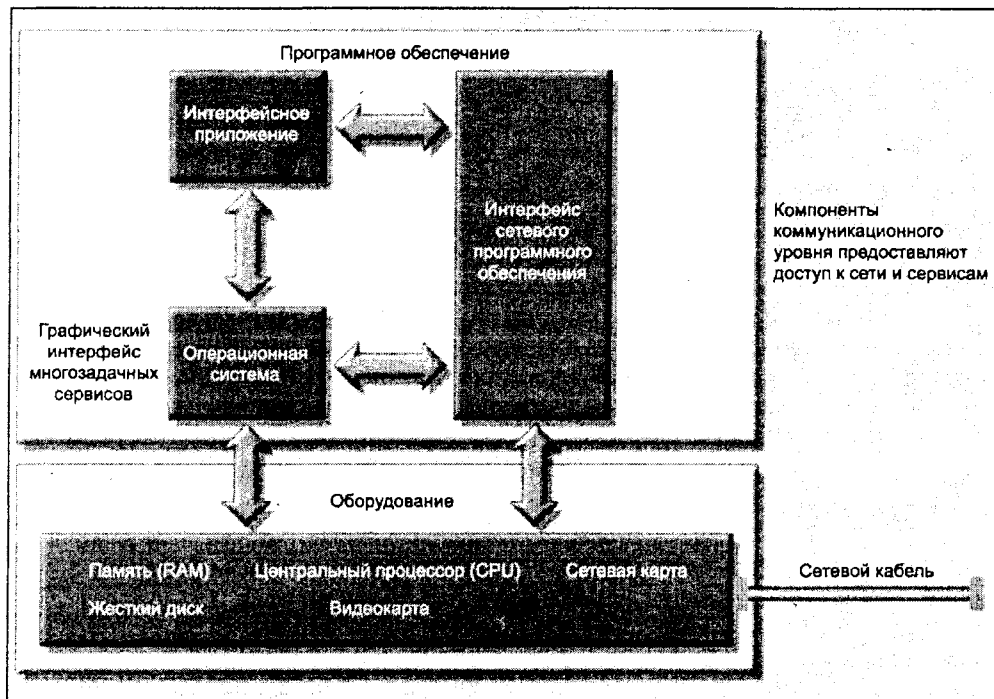


Рис. 12.5. Компоненты клиента

На рис. 12.5 обратите внимание на то, что интерфейсное приложение взаимодействует с операционной системой для получения доступа к многозадачным функциям и графическому интерфейсу пользователя, обеспечиваемым системой. Интерфейсное приложение взаимодействует также с компонентами сетевого промежуточного программного обеспечения (ППО) для получения доступа к сервисам, расположенным в сети. Компоненты оборудования сетевого ППО (сетевой кабель и сетевые платы) физически передают запросы и ответы между клиентами и серверами. Пока запросы обрабатываются сервером, клиент может выполнять любые другие задачи.



## 12.5.4. Компоненты сервера

Как уже упоминалось, сервером называют любой процесс, предоставляющий сервис клиентскому процессу. Сервер — это реактивный процесс, поскольку он всегда находится в состоянии ожидания запросов со стороны клиента. Как правило, сервер обеспечивает следующие возможности:

- *файловые сервисы* для локальных сетей, в которых компьютер с быстрыми дисками большой емкости совместно используется разными пользователями. Клиент, подключенный к сети, может хранить файлы на файл-сервере как на дополнительном локальном жестком диске;
- *сервисы печати* для локальных сетей, в которых персональный компьютер (ПК) с одним или более принтерами совместно используется несколькими клиентами. Клиент может получить доступ к любому из принтеров так, словно он напрямую подключен к его собственному компьютеру. Данные, которые необходимо напечатать, передаются от клиентского ПК на сервер печати, где они временно размещаются на диске. Когда клиент заканчивает пересылку данных на печать, информация передается с жесткого диска сервера печати на соответствующий принтер;
- *сервисы факсимильной связи*, которые требуют, по крайней мере, чтобы один сервер был оборудован факсом (внутренним или внешним). Иметь факс-устройство или даже подключение к телефонной линии на клиентском ПК совершенно необязательно. Вместо этого клиент представляет данные, которые надо передать по факсу, на факс-сервер, сопроводив их необходимой вспомогательной информацией, например, номером факса или именем получателя. Факс-сервер планирует отправку этого факса, в нужное время наберет номер факса адресата и передаст факс. Факс-сервер должен также уметь обрабатывать все ошибки, возникающие при выполнении этой задачи;
- *сервисы передачи данных*, которые позволяют клиентским ПК подключаться к серверу передачи данных, чтобы получить доступ к другим хост-компьютерам или сервисам, к которым клиент не может подключиться непосредственно. Например, сервер передачи данных позволяет клиентским ПК подключаться к электронным доскам объявлений, к удаленной локальной сети и т. д.;
- *сервисы баз данных*, в которых содержатся наиболее распространенные и наиболее успешные клиент/серверные реализации. При наличии сервера баз данных клиент посылает SQL-запросы на сервер. Сервер получает SQL-код, проверяет его, выполняет, а клиенту отсылает только результат. Данные и устройство базы данных размещаются на компьютере сервера базы данных. Клиенту необходимо иметь только интерфейсную программу для получения доступа к серверу базы данных;
- *сервисы транзакций*, предоставляемые серверами транзакций, подключенными к серверу базы данных. Сервер транзакций содержит код транзакции базы данных или процедуры, которые манипулируют информацией в базе данных. Интерфейсная прикладная программа на клиентском ПК посылает запрос на сервер транзакций для выполнения специфической процедуры, хранящейся на сервере баз данных. При этом по сети не передается никакой SQL-код. Серверы тран-

закций снижают нагрузку сети и обеспечивают лучшую производительность, чем серверы баз данных;

- *другие сервисы*, включая обслуживание устройств CD-ROM, видео, резервное копирование и т. д.

Обычно неверно полагают, что серверный процесс должен запускаться на компьютере, где имеется сетевая операционная система. Цитируя "Порги и Бесс", можно сказать что "это не обязательно так". На самом деле, если обстоятельства не вынуждают поступить иначе, настоятельно рекомендуется разделять серверный процесс и сетевую операционную систему. Такое разделение позволяет размещать серверный процесс на любом компьютере сети и делать его доступным для любого клиентского компьютера. Например, предположим, что у нас имеется сервер CD-ROM в сети Novell NetWare. Если программное обеспечение CD-ROM-сервера требует, чтобы серверный процесс выполнялся на том же компьютере, что и операционная система NetWare, то хост-компьютер будет слишком сильно нагружен. Хост теперь должен выполнять и роль файл-сервера, и роль CD-ROM-сервера. Если продукту не нужно такое "дублирование", то он может быть установлен на любом сетевом ПК, что позволит распределить нагрузку более равномерно. Оба типа продуктов используют сетевые протоколы (IPX или TCP/IP), предоставляемые сетью Novell NetWare для передачи сообщений между клиентами и серверами. Каждое решение имеет свои преимущества и недостатки. Лучшее решение всегда зависит от конкретных обстоятельств.

Сервер, как и клиент, также имеет компоненты оборудования и программного обеспечения. К компонентам оборудования относятся компьютер, центральный процессор, жесткий диск, видео, сетевая карта и т. д. Компьютер, на котором располагается процесс сервера, должен быть более мощным компьютером, чем "среднестатистический" клиентский компьютер, поскольку серверный процесс должен уметь одновременно обрабатывать запросы от нескольких клиентов. Компоненты сервера представлены на рис. 12.6.

Серверное приложение выполняется поверх операционной системы и взаимодействует с коммуникационным ППО, которое "слушает" (ожидает) запросы клиента на предоставление сервиса. В отличие от интерфейсного приложения, серверному процессу не нужен графический интерфейс. Нужно помнить, что серверное приложение взаимодействует с операционной системой (сетевой или локальной) для получения доступа к локальным ресурсам (жесткий диск, память, время центрального процессора и т. д.). Сервер постоянно "слушает" клиентские запросы. Как только запрос получен, сервер обрабатывает его локально. Сервер знает, как обрабатывать запрос; клиент сообщает серверу только то, ЧТО необходимо выполнить, а не КАК это нужно выполнять. После выполнения запроса ответ отсылается обратно клиенту с помощью коммуникационного ППО.

Характеристики серверного оборудования зависят от уровня предоставляемого сервиса. Например, для сервера баз данных, предназначенного для использования в сети из 50 клиентов, может потребоваться компьютер со следующими характеристиками:

- быстрый процессор (RISC, Pentium, PowerPC или многопроцессорный);
- высокая отказоустойчивость:
  - наличие системы двойного подвода питания для предотвращения проблем при отключении электропитания компьютера;

- резервный источник питания для защиты от обрыва основной линии электропитания;
- обнаружение и исправление ошибок (Error Checking and Correction, ECC) памяти для защиты от сбоев модулей памяти;
- массив резервных жестких дисков (Redundant Array of Independent Disks, RAID) или RAID-массив для обеспечения защиты от физической поломки дисков;

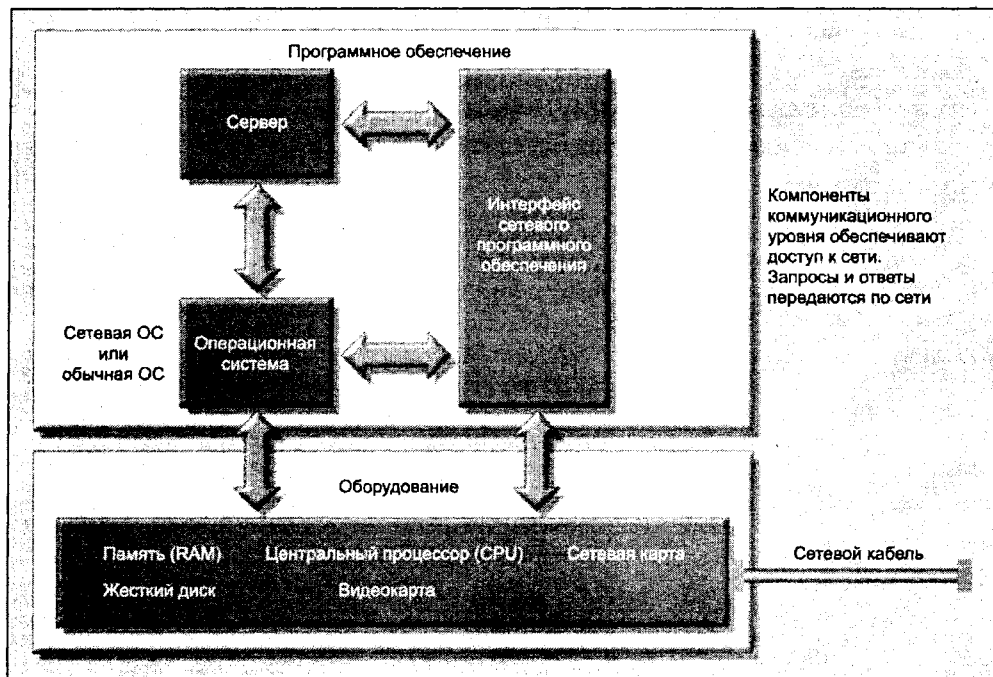


Рис. 12.6. Компоненты сервера

- ❑ возможность модернизации центрального процессора, расширения объема памяти, диска и изменения периферии;
- ❑ поддержка шины для подключения дополнительного оборудования;
- ❑ различные коммуникационные возможности.

Теоретически любой процесс компьютера, разделяемый на клиентский и серверный компоненты, можно реализовать с помощью клиент/серверной модели. При правильной реализации принципов клиент/серверной архитектуры по распределению обработки данных можно получить ряд преимуществ.

- ❑ *Независимость от местоположения.* Процесс сервера может быть размещен в любом месте компьютерной сети.

- ☐ *Оптимизация ресурсов.* Процесс сервера может совместно использоваться несколькими клиентскими процессами.
- ☐ *Масштабируемость.* Процесс сервера можно модернизировать для выполнения на более мощной платформе.
- ☐ *Интеграция и способность к взаимодействию.* Процесс сервера должен обеспечивать возможность работы в среде "plug-and-play" (технология автоматического конфигурирования аппаратных средств, "подключи и работай", P-n-P).

Эти преимущества вкупе с независимостью от аппаратного и программного обеспечения клиент/серверной модели облегчают интеграцию ПК, мини-компьютеров и мэйнфреймов в почти идеальную инфраструктуру.

### 12.5.5. Компоненты коммуникационного промежуточного программного обеспечения (ППО)

Коммуникационное ППО предоставляет средства, с помощью которых клиенты и серверы взаимодействуют при выполнении специфических задач. Оно также предоставляет специальные сервисы клиентскому процессу, отделяющие интерфейсные приложения от внутренних действий сервера баз данных и сетевых протоколов. В прошлом прикладные программисты должны были писать код, который напрямую взаимодействовал со специфическим языком базы данных (как правило, это был SQL) и специфическими сетевыми протоколами, используемыми сервером БД. Например, при написании интерфейсного приложения для доступа к базе данных IBM OS/2 Database Manager программист должен был включать в программу команды SQL и NetBIOS. Команды NetBIOS позволяли клиентскому процессу устанавливать сеанс обмена данными с сервером БД, посылать специфическую управляющую информацию, запросы и т. д. (позже в этой главе мы рассмотрим протокол NetBIOS и другие сетевые протоколы). Если одно и то же приложение использовалось в различных базах данных и сетях, приходилось заново переписывать приложения в соответствии с требованиями других баз и протоколов. Очевидно, что это было очень неудобно, и здесь на помощь пришло ППО. Описание ППО, приводимое в этой главе, основано на его основном предназначении и основных функциях этой новой категории программ.

Хотя ППО может использоваться в разных вариантах, например, e-mail (электронная почта), факсимильная связь, или сетевые протоколы, большая часть ППО первого поколения, используемого в клиент/серверных приложениях, была ориентирована на обеспечение прозрачности доступа к данным некоторых серверов БД. Применение ППО дает определенные преимущества.

- ☐ *Независимость от сети* позволяет интерфейсным приложениям получать доступ к данным независимо от сетевых протоколов.
- ☐ *Независимость от сервера БД* позволяет интерфейсным приложениям получать доступ к данным нескольких серверов БД без необходимости адаптации кода под специфические требования каждого сервера БД.

Промежуточное программное обеспечение БД позволяет программисту использовать выражения стандартного языка SQL для получения доступа к различным серверам

БД. ППО избавляет программистов от необходимости помнить о различиях диалектов SQL, трансформируя выражения стандартного SQL в подходящий для данной БД синтаксис. Например, проблема разработки интерфейсных систем для нескольких серверов БД состояла в том, что прикладные программисты должны были иметь глубокие познания в области сетевых коммуникаций и языков доступа к каждой базе данных для извлечения из них необходимой информации. Проблема усугублялась еще и тем, что каждый поставщик БД реализовал собственную версию SQL (с другим синтаксисом, дополнительными функциями и расширениями по сравнению со стандартным SQL). Более того, данные могли размещаться в нереляционных БД (в иерархических, сетевых или в системе плоских файлов), которые вообще не поддерживали SQL, что очень затрудняло извлечение информации из них. На фоне таких сложностей программирование в клиент/серверных системах становилось более сложным, чем программирование в традиционных системах с мэйнфреймом. ППО баз данных упростило задачу доступа к нескольким источникам данных в различных сетях, а также освободило программистов от необходимости управления сетевыми коммуникациями.

Для этого коммуникационное ППО действует по двухуровневой схеме.

- *Физический уровень* имеет дело с коммуникациями между клиентским и серверным компьютерами (связь "компьютер-компьютер"). Иначе говоря, на этом уровне определяется, как физически связаны компьютеры. Физические связи включают в себя сетевое аппаратное и программное обеспечение. Сетевое программное обеспечение состоит из сетевых протоколов. (Вспомните, что сетевые протоколы представляют собой правила, определяющие способ взаимодействия компьютеров в сети и обеспечивающие возможность компьютерам отправлять и получать сигналы друг другу.) Физически коммуникационное ППО в большинстве случаев представляет собой локальную сеть. Но поскольку клиент/серверная модель позволяет клиенту и серверу размещаться на одном и том же компьютере, она может существовать и без локальной сети.
- *Логический уровень* относится к связям между клиентским и серверным процессами (связь "процесс-процесс"), т. е. определяет взаимодействие процессов клиента и сервера. Логика определяется *протоколами связи между процессами* (*Interprocess Communication Protocols, IPC*), которые придают сигналам определенное значение. Именно на этом уровне происходит большая часть обмена между клиентом и сервером.

Чтобы проиллюстрировать два уровня клиент/серверного обмена, используем следующую аналогию. Предположим, вы заказываете по телефону пиццу. Вы начинаете с того, что снимаете трубку, набираете номер и ждете ответа. Когда номер отвечает, вы называете себя, сообщаете агенту, какую пиццу вы хотели бы получить, количество и другие сведения. В свою очередь, агент спрашивает вас адрес, называет стоимость заказа и сообщает примерные сроки доставки. Эта простая транзакция требует действий и на физическом, и на логическом уровне:

- действия на физическом уровне включают в себя телефонное преобразование звука в аналоговые сигналы и последовательную передачу этих сигналов по телефонным линиям на центральную АТС компании, а оттуда на телефон агента по доставке пиццы;

- ❑ действия на логическом уровне выполняются вами и агентом. Поскольку вы говорите на одном языке, то формат сервиса понятен агенту, и детали транзакции можно спокойно обсуждать.

Кроме того, что вы должны уметь пользоваться телефоном, остальные физические подробности телефонной связи от вас скрыты. Телефонная компания берет на себя обеспечение физического уровня вашего разговора, где бы вы и ваш агент не обсуждали логическую сторону транзакции.

Хотя представленная аналогия и помогает понять основы взаимодействия "клиент/сервер", все же необходимо знать некоторые подробности компьютерных коммуникаций для лучшего понимания управления потоком данных в клиент/серверных инфраструктурах. Для этого мы представим эталонную сетевую модель программы OSI (Open Systems Interconnection, взаимодействие открытых систем — международная программа стандартизации обмена данными между компьютерными системами различных производителей). Эта модель, опубликованная в 1984 году, была разработана ISO (International Organization for Standardization, международная организация по стандартизации) в рамках работ по стандартизации различных сетевых систем. Модель OSI основана на семи независимых друг от друга уровнях. Эталонная модель OSI, представленная в табл. 12.2, разработана так, что каждый ее уровень предоставляет определенные сервисы для уровня, находящегося выше него.

При изучении эталонной сетевой модели OSI обратите внимание на то, как происходит обмен потоками данных в сети. Уровни приложения и представления предоставляют сервисы локальному компьютеру, предназначенному для подготовки и форматирования отсылаемых данных. Уровни сеанса, транспортный, сетевой, канала передачи данных и физический предоставляют сервисы уровням, находящимся над ними, для обеспечения безопасности доставки данных от одного сетевого узла к другому.

Цель нижних уровней — скрыть сложности реализации сети от вышестоящих уровней. Вкратце можно сказать, что:

- ❑ уровни приложения и представления обеспечивают конечного пользователя функциями, ориентированными на приложение;
- ❑ уровень сеанса обеспечивает управление связью "программа-программа";
- ❑ уровни транспортный, сети, канала передачи данных и физический предоставляют функции, имеющие отношение к сети.

**Таблица 12.2. Эталонная сетевая модель OSI**

| Уровень       | Описание                                                                                                                                                                                                          |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Приложения    | Программа-приложение конечного пользователя. Клиент: интерфейсное приложение, например, e-mail или электронные таблицы. Сервер: серверное приложение, например, файл-сервер, сервер базы данных или e-mail-сервер |
| Представления | Предоставляет функции форматирования, конвертирования, компрессии, кодирования и т. д. для преобразования протокола приложения                                                                                    |
| Сеанса        | Устанавливает связь между приложениями и управляет ею. Обеспечивает безопасность, доставку и восстановление связи                                                                                                 |

Таблица 12.2 (окончание)

| Уровень                | Описание                                                                                                                                                                                        |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Транспортный           | Предоставляет распознавание ошибок и восстановление, гарантирует, что все данные будут должным образом доставлены и обеспечены специфическим идентификатором транспортного уровня               |
| Сетевой                | Предоставляет сквозную маршрутизацию пакетов. Разделяет длинные сообщения на более мелкие                                                                                                       |
| Канала передачи данных | Создает "фреймы" для передачи и управления совместным доступом к сетевому физическому устройству (кабель). Включает выявление ошибок, коррекцию и т. д.                                         |
| Физический             | Предоставляет стандарты, связанные с электрическими деталями передачи данных (сетевые карты, тип кабеля, напряжение и т. д.). Физически передает фреймы данных по кабелю или другим устройствам |

Чтобы лучше проиллюстрировать функции эталонной модели OSI, посмотрим, как сервер базы данных обслуживает запросы клиента в сети. На рис. 12.7 представлен информационный поток на каждом уровне модели OSI.

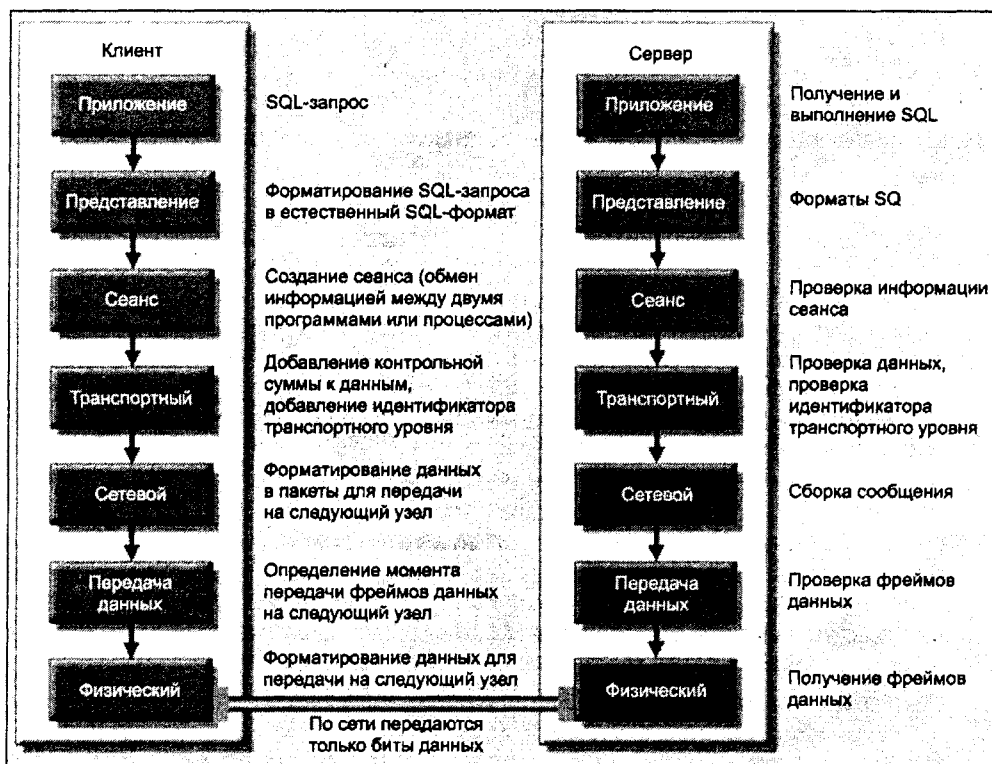


Рис. 12.7. Информационный поток в модели OSI

С помощью рис. 12.7 мы можем проследить поток информации:

1. Клиентское приложение создает SQL-запрос.
2. SQL-запрос передается на уровень представления, где он преобразуется в формат, понятный серверной SQL-машине. Сюда входят трансляция символов ASCII, выявление чисел с простой и двойной точностью и определение форматов данных (например, mm/dd/yyyy вместо dd/mm/yyyy).
3. SQL-запрос передается на уровень сеанса. На уровне сеанса устанавливается связь клиентского процесса с серверным процессом. Если процесс сервера БД требует подтверждения полномочий пользователя, то на уровне сеанса создаются необходимые сообщения для регистрации и проверки конечного пользователя. В это время, обычно в начале сеанса, от конечного пользователя может потребоваться ввод идентификатора пользователя (ID) и пароля для получения доступа к серверу БД, после чего между клиентом и сервером могут передаваться другие сообщения. На уровне сеанса определяется, какие из сообщений являются управляющими, а какие несут в себе данные.
4. После установки и проверки сеанса SQL-запрос передается на транспортный уровень. На этом уровне создается контрольная сумма для проверки ошибок и добавляется специфическая идентификационная информация транспортного уровня. Например, когда на клиенте выполняются несколько процессов, все процессы могут обрабатывать разные SQL-запросы или получать доступ к разным серверам БД. Идентификатор транспортного уровня помогает определять принадлежность данных к сеансу.
5. Когда транспортный уровень завершает свои функции, SQL-запрос передается на уровень сети. Сетевой уровень принимает SQL-запрос, идентифицирует адрес узла получателя (местоположение сервера), добавляет к пути адрес следующего узла (если это необходимо), разделяет SQL-запрос на несколько мелких пакетов и присваивает всем пакетам последовательные номера, чтобы обеспечить их сборку в нужном порядке.
6. Пакет передается на уровень канала передачи данных. На этом уровне добавляется дополнительная управляющая информация, зависящая от сети и используемого физического носителя. Эта информация добавляется в начало (*заголовок*) и в конец (*трейлер*) пакета. Результат этого процесса называется *фреймом*. Затем уровень канала передачи данных передает фрейм на следующий узел. Уровень канала передачи данных отвечает за совместное использование сетевого устройства (кабель) и гарантирует целостность фрейма.
7. Когда на уровне канала передачи данных будет установлена возможность безопасной передачи фрейма, он передается на физический уровень, который переводит его в последовательность нулей и единиц (битов), а затем передает эти биты по сетевому кабелю. На физическом уровне происходит только передача сигнала, а не интерпретация данных.
8. Сигналы, переданные на физическом уровне, получают на физическом уровне принимающего сервера, который передает данные на уровень канала передачи данных. На этом уровне данные реконструируются из последовательности битов во фреймы, а затем проверяются. В это время уровни канала передачи данных клиента и сервера могут обмениваться дополнительными сообщениями, которые



- либо подтверждают корректность принятых данных, либо требуют повторной передачи. Затем на уровне канала передачи данных из пакета удаляется информация заголовка и трейлера, а пакет передается на сетевой уровень.
9. На сетевом уровне проверяется адрес доставки пакета. Если адресом является какой-то другой узел сети, то этот узел идентифицируется, и пакет опять передается на уровень канала передачи данных для отправки его на соответствующий узел. Если адресатом является текущий узел, то пакеты объединяются и им присваивается определенный номер. Затем генерируется SQL-запрос и передается на транспортный уровень.
  10. На транспортном уровне происходит дополнительная проверка данных, а затем сообщение направляется на соответствующий сеанс на основе идентификационного номера транспортного уровня. На рис. 12.8 показано, как с помощью идентификатора транспортного уровня корректно распределяются сетевые запросы к серверу БД, обслуживающему множество клиентов.

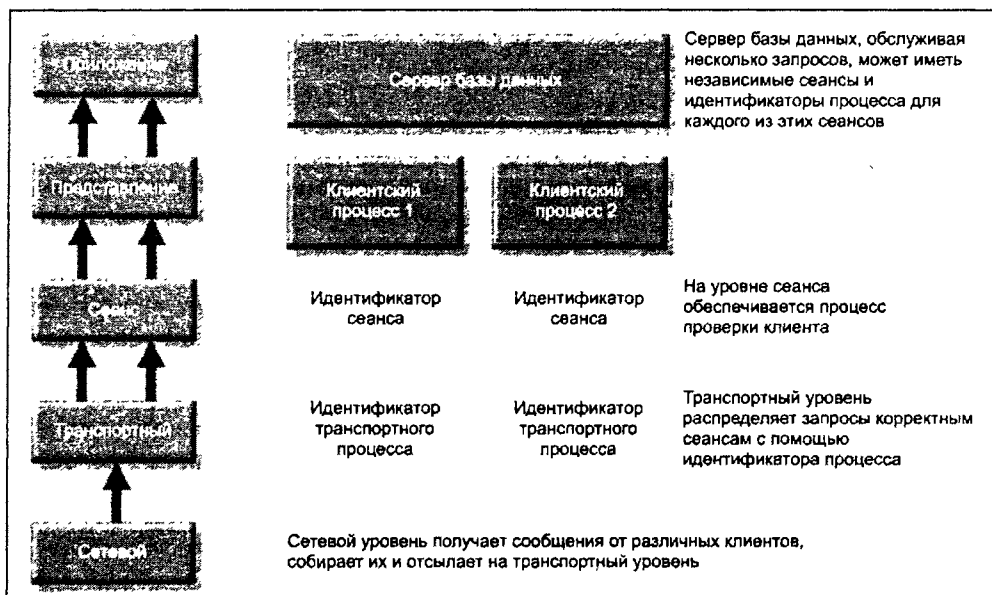


Рис. 12.8. Идентификатор транспортного уровня

11. Большая часть "разговора" клиента и сервера приходится на уровень сеанса. Если связь между клиентом и сервером прерывается, то на уровне сеанса происходит попытка восстановления сеанса. Уровень сеанса идентифицирует и проверяет запрос, а затем передает его на уровень представления.
12. На уровне представления выполняются дополнительная проверка и форматирование.
13. SQL-запрос посылается на сервер БД или на уровень приложения, где он и выполняется.

Помните, что хотя модель OSI и помогает понять сетевые взаимосвязи, она функционирует внутри системы с развитой инфраструктурой (основные компоненты этой инфраструктуры описаны в *Приложении*). Сетевые протоколы составляют ядро сетевой инфраструктуры, поскольку все данные, передаваемые по сети, должны соответствовать какому-то сетевому протоколу. В клиент/серверной среде могут использоваться несколько различных протоколов. В предыдущем разделе этой главы было отмечено, что для связи по сети различные процессы сервера могут использовать различные сетевые протоколы. В следующем разделе мы исследуем основные сетевые протоколы, используемые в клиент/серверных вычислениях.

## 12.5.6. Сетевые протоколы

*Сетевой протокол* представляет собой набор правил, определяющих передачу, интерпретацию и обработку сообщений между компьютерами. Сетевые протоколы позволяют компьютерам взаимодействовать в сети и работать на разных уровнях модели OSI. Для обозначения сетевого протокола используются и другие термины, например, *LAN-протокол* и *сетевой транспортный протокол*. К основным сетевым протоколам относятся:

- ❑ *Transmission Control Protocol/Internet Protocol, TCP/IP (Протокол управления передачей данных/интернет-протокол)* представляет собой официальный протокол связи в Интернете — всеобщей сети гетерогенных компьютерных систем. TCP/IP — основной коммуникационный протокол, используемый в UNIX-системах. Он поддерживается большинством операционных систем среднего уровня и уровня персональных компьютеров и является фактическим стандартом для гетерогенных сетевых подключений. Поскольку UNIX считается предпочтительной операционной системой для средних и больших серверов баз данных, TCP/IP играет значительную роль в клиент/серверных приложениях;
- ❑ *Internetwork Packet Exchange/Sequenced Packet Exchange, IPX/SPX (Протокол межсетевого обмена пакетами/протокол последовательного обмена)* — коммуникационный протокол, разработанный фирмой Novell, одной из лидирующих компаний в разработке операционных систем. Протокол IPX/SPX не очень хорошо зарекомендовал себя в сетях MAN (Metropolitan Area Networks, региональная вычислительная сеть) или WAN (Wide Area Networks, глобальная вычислительная сеть), в которых очень высок уровень сетевого трафика. Именно поэтому последние версии операционных систем Novell адаптированы под использование протокола TCP/IP в качестве основного;
- ❑ *Network Basic Input/Output System, NetBIOS (Сетевая базовая система ввода/вывода)* — сетевой протокол, изначально разработан корпорациями IBM и Sytek в 1984 году в качестве стандарта для коммуникационных приложений персональных компьютеров. NetBIOS поддерживается большинством операционных систем для ПК и большинством приложений для ПК. Ограничения NetBIOS не позволяют использовать этот протокол в географически распределенных сетях. Производительность его еще ниже, чем у протокола IPX/SPX;
- ❑ *Advanced Program-to-Program Communications, APPC (Улучшенный интерфейс связи между программами)* представляет собой коммуникационный протокол, используемый в среде мэйнфреймов IBM *Systems Network Architecture (SNA, Системная*

*сетевая архитектура*). Этот протокол обеспечивает связь между персональными компьютерами и приложениями мэйнфреймов (например, СУБД DB2, выполняющаяся на мэйнфрейме). APPC используется в системных решениях компании IBM при создании клиент/серверных приложений, в которых задействованы ПК, компьютеры средней мощности (IBM AS/400 и RISC/6000) и мэйнфреймы;

- *AppleTalk* представляет собой набор коммуникационных протоколов, используемых для сетевых коммуникаций компьютеров фирмы Apple.

Протоколы TCP/IP и IPX/SPX в настоящее время являются доминирующими. Хотя протокол NetBIOS постепенно сходит со сцены, некоторые значимые бизнес-приложения продолжают его использовать. Сетевые протоколы мэйнфреймов также используются во многих компаниях, особенно в тех случаях, когда компания использует мэйнфрейм в качестве основного хранилища информации. В результате бума клиент/серверных вычислений многие мэйнфреймы и компьютеры средней мощности стали поддерживать не только собственные коммуникационные протоколы, но общепринятые протоколы типа TCP/IP, что позволяет обеспечить прямой доступ к клиент/серверным приложениям, организованным на базе ПК.

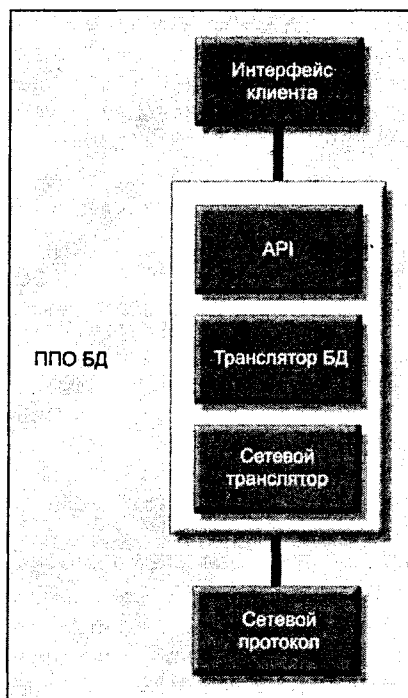
При реализации сетевой инфраструктуры выбор сетевого протокола непосредственно влияет на используемый программный продукт. Например, сервер базы данных на персональном компьютере может ограниченно использовать сетевой протокол IPX. Большинство серверов баз данных, основанных на операционной системе UNIX, используют протокол TCP/IP. В компаниях с несколькими серверами, сетями и клиентами инфраструктура сетевых коммуникаций (мосты, маршрутизаторы и т. д.) должна обеспечивать передачу данных от одного протокола к другому.

Выбор сетевой топологии и сетевых протоколов — одно из самых важных решений в разработке клиент/серверных систем. Для разработчиков коммерческих систем это решение, влияющее на уровень продаж, поскольку они хотят продать свой продукт на рынке как можно быстрее. Поэтому разработчики коммерческого программного обеспечения будут использовать те сетевые протоколы, которые обеспечат им наибольшее число потенциальных покупателей. Коммерческая разработка клиент/серверных интерфейсных и серверных программ должна обеспечивать поддержку нескольких сетевых протоколов для связи между различными серверами и клиентами. При принятии решения о выборе сетевого протокола разработчикам систем MIS (управленческих информационных систем) приходится учитывать и дополнительную важную информацию. Например, имеется ли уже в компании сетевая инфраструктура? Есть ли у компании мэйнфрейм или глобальная сеть, которые необходимо интегрировать в систему? Каков уровень профессионализма? Заново разрабатывать приложения для поддержки других сетевых протоколов и других баз данных невыгодно ни коммерческим программистам, ни корпоративным разработчикам систем MIS.

### 12.5.7. Компоненты ППО базы данных

Как показано на рис. 12.9, ППО базы данных подразделяется на три основных компонента:

- программный интерфейс приложения (Application Programming Interface, API);
- транслятор базы данных;
- сетевой транслятор.



**Рис. 12.9.** Промежуточное программное обеспечение базы данных

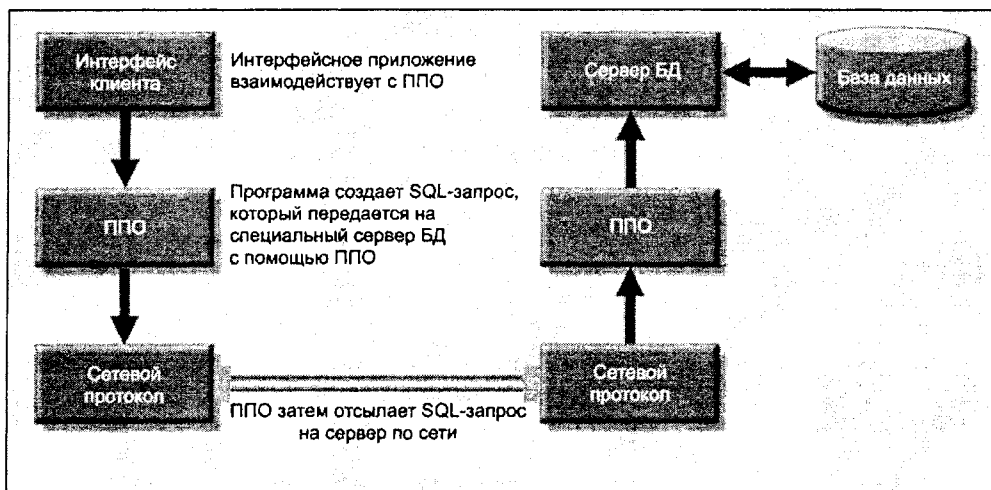
Эти компоненты (или их функции) в основном распределяются по нескольким уровням программного обеспечения, которые можно заменять по типу технологии Р-п-Р.

*Программный интерфейс приложения (API)* открыт для клиентского приложения. Программист взаимодействует с ППО через API, поставляемый вместе с ППО. API ППО позволяет программисту писать стандартный код SQL вместо кода, специфичного для данного сервера БД. Иначе говоря, API обеспечивает независимость клиентского процесса от сервера БД. Такая независимость означает, что сервер можно заменить без необходимости переписывать клиентское приложение.

*Транслятор баз данных* транслирует SQL-запросы в специфичный синтаксис сервера БД. На уровне транслятора БД SQL-запрос отображается на SQL-протокол сервера БД. Поскольку сервер базы данных может иметь какие-то специальные функциональные возможности, транслятор БД может оптимально транслировать основной SQL-запрос в специфичный формат, используемый на сервере БД. Если SQL-запрос использует данные от двух различных серверов БД, то транслятор БД возьмет на себя заботу о связи с каждым сервером и об извлечении данных в формате, подходящем для клиентского приложения.

*Сетевой транслятор* управляет сетевыми коммуникационными протоколами. Напомним, что серверы БД могут использовать любые сетевые протоколы, которые мы

обсуждали ранее. Поэтому если клиентское приложение подключается к двум базам данных, в одной из которых используется протокол TCP/IP, а в другой IPX/SPX, то сетевой транслятор обрабатывает все детали связи каждой базы данных прозрачно для клиентского приложения. На рис. 12.10 показано взаимодействие между клиентом и компонентами ППО БД.



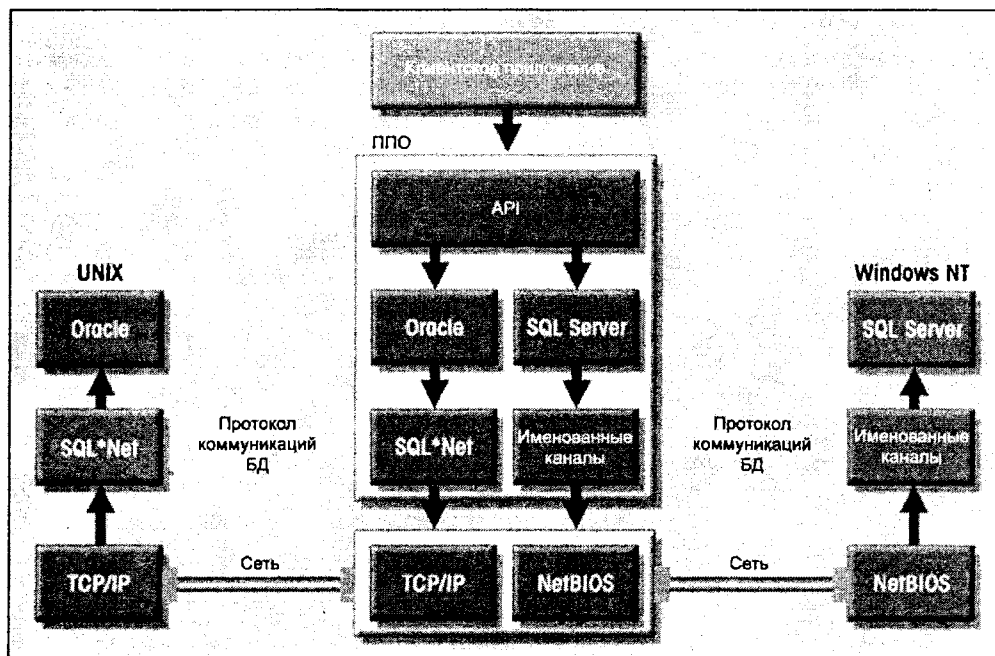
**Рис. 12.10.** Взаимодействие между клиент/серверными компонентами ППО

С учетом трех компонентов ППО, представленных на рис. 12.9, можно определить три основных преимущества использования ППО. Клиенту предоставляется:

- ☐ доступ к нескольким (и достаточно разным) базам данных;
- ☐ независимость от сервера БД;
- ☐ независимость от сетевого протокола.

Чтобы показать совместную работу всех этих составляющих, продемонстрируем, как осуществляется доступ клиента к двум различным серверам базы данных. На рис. 12.11 показано, каким образом клиентское приложение запрашивает данные от сервера базы данных Oracle (корпорация Oracle) и от сервера баз данных SQL Server (корпорация Microsoft). Сервер Oracle в качестве протокола связи с клиентом использует SQL\*Net; SQL Server использует в качестве коммуникационного протокола Named Pipes (именованные каналы). Протокол *SQL\*Net* — решение, предназначенное для баз данных Oracle; используется для передачи SQL-запросов по сети. *Named Pipes* — это протокол связи между процессами (interprocess communications protocol, IPC) для многозадачных операционных систем, таких как UNIX и OS/2, используемый в СУБД SQL Server для управления связями клиентов и серверов по сети.

На рис. 12.11 обратите внимание, что сервер Oracle выполняется под управлением UNIX и использует сетевой протокол TCP/IP. SQL Server выполняется под управлением операционной системы Windows NT и использует сетевой протокол NetBIOS.



**Рис. 12.11.** Доступ к нескольким серверам базы данных с помощью ППО

В данном случае клиентское приложение использует базовый SQL-запрос для доступа к данным в двух таблицах: в таблице Oracle и в таблице SQL Server. Транслятор БД ППО содержит два модуля по одному на каждый тип сервера БД, к которому необходимо получить доступ.

Каждый модуль обрабатывает тонкости каждого сетевого протокола базы данных. Сетевой транслятор берет на себя задачу поддержки надлежащего сетевого протокола для доступа к каждой БД. При получении результатов запроса они представляются в формате, пригодном для клиентского приложения. Конечный пользователь или программист не должен беспокоиться о тонкостях получения данных от серверов. Фактически конечный пользователь может даже не знать местоположение данных или то, какой тип СУБД используется при извлечении данных.

Другой пример использования ППО для предоставления прозрачного доступа к базе данных представлен на рис. 12.12. В этом случае сеть обслуживает нескольких клиентов, получающих данные от мэйнфрейма IBM, где находится база данных DB2. Клиенты представлены компьютерами с операционными системами Windows 98, Windows 2000, Open Linux и OS/2, которые запрашивают данные по сетевому протоколу NetBIOS.

Обратите внимание (в нижней части рис. 12.12), что база данных DB2 мэйнфрейма для связи с компьютерами сети использует протокол APPC. Для трансляции запро-

сов NetBIOS клиентов в протокол APPC и доступа к данным БД мэйнфрейма используется сетевой компьютер. Такой компьютер называется *шлюзом* (*gateway*). Компьютер-шлюз предоставляет функции трансляции данных между компьютерами и сетью. На самом деле, термин "шлюз" относится к особому типу промежуточного программного обеспечения; поэтому компьютер-шлюз обязательно использует ППО. В данном случае ППО установлено на нескольких компьютерах.

В соответствии со сценарием, представленным на рис. 12.12, клиентские приложения запрашивают данные у мэйнфрейма IBM DB2. Компонент DB2/2 на клиентском компьютере выполняет некоторые функции транслятора баз данных. Компонент CM/2 на клиентском компьютере управляет сетевыми коммуникациями в сети Token Ring. Компьютер-шлюз использует компоненты DB2/2, DDCS/2 и CM/2 для обеспечения прозрачности базы данных в сети. Компонент CM/2 на шлюзе транслирует запросы от NetBIOS на APPC и посылает запросы к базе данных мэйнфрейма DB2. Компоненты ППО, расположенные на клиентском и шлюзовом компьютерах, работают в сети сообща, обеспечивая прозрачность сети и БД для всех клиентских приложений.

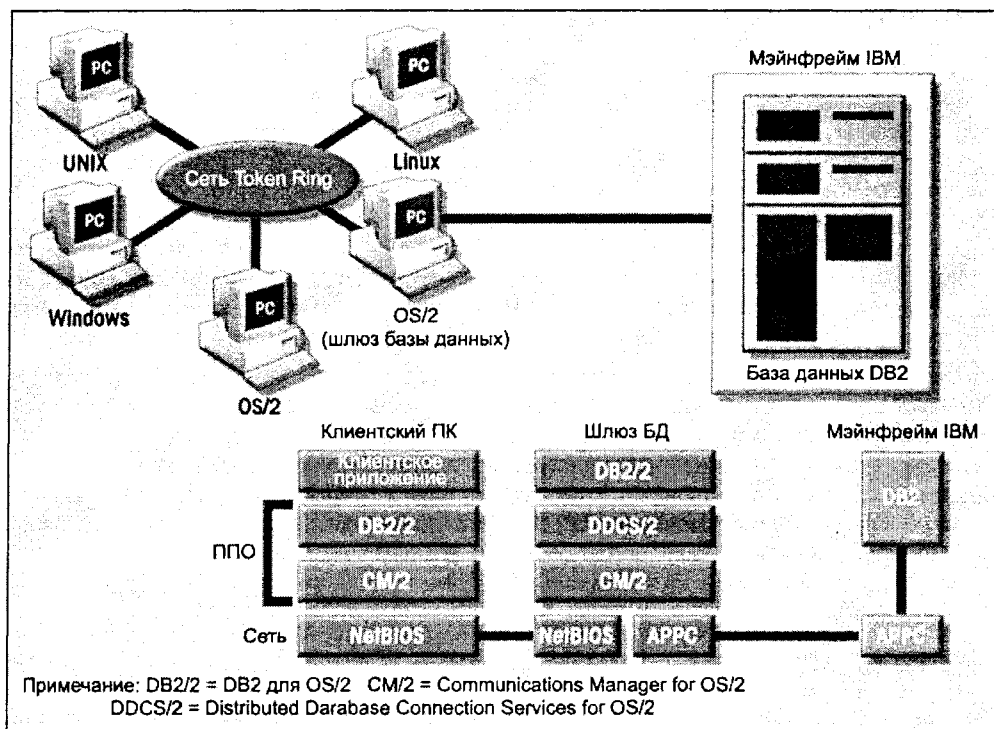


Рис. 12.12. Доступ к базе данных мэйнфрейма с помощью ППО

## 12.5.8. Классификация ППО

ППО базы данных можно классифицировать по способу связи клиентов и серверов по сети. По этому признаку ППО обычно классифицируют следующим образом.

- ❑ ППО, ориентированное на сообщения (message-oriented middleware, MOM).
- ❑ ППО, основанное на вызовах удаленных процедур (RPC-based).
- ❑ ППО, основанное на объектах.

Выбор ППО в большой степени зависит от приложения. Например, ППО, основанное на вызовах удаленных процедур, больше подходит для высокоинтегрированных систем, в которых целостность данных имеет высший приоритет, а также для сетей с высокой пропускной способностью. ППО, ориентированное на сообщения, в основном более эффективно в локальных сетях с ограниченной пропускной способностью и в приложениях, где целостность данных не столь критична. ППО, основанное на объектах, представляет собой новый тип ППО, использующего объектно-ориентированную технологию. Хотя данный тип ППО не имеет такого широкого распространения, как первые два, ожидается, что оно сможет обеспечить лучшую интеграцию систем и лучшее управление.

ППО это самая активная область разработок прикладных программистов. Поэтому поставщики сосредотачиваются на разработке новаторских решений ППО и продуктов, наиболее успешные из которых быстро доминируют на рынке, фактически становясь стандартами. Доминирующим стандартом де-факто ППО базы данных стал интерфейс Open Database Connectivity (ODBC, открытый интерфейс доступа к базам данных), разработанный в корпорации Microsoft. Интерфейс ODBC первоначально разрабатывался для обеспечения ППО API Windows-приложений, не зависящих от источника данных. Для доступа к каждой базе данных используется специфический драйвер ODBC (например, Oracle ODBC-драйвер или SQL Server ODBC-драйвер). ODBC требует наличия коммуникационного протокола (например, Named Pipes или SQL\*Net) для связи с сервером баз данных. ODBC также обеспечивает возможность доступа к специфическим функциям базы данных, если они требуются клиентскому приложению.

Воодушевленная успехом ODBC, Microsoft в настоящее время продвигает стандарт OLE DB (технология OLE для баз данных) в качестве нового стратегического ППО баз данных. OLE DB обеспечивает универсальную интеграцию данных на уровне предприятия на основе спецификации Component Object Model (COM, модель компонентных объектов), составляющей основу для хранения и извлечения информации из БД. OLE DB может использоваться для доступа к различным типам данных — звук, видео, текст и числовые данные — от нескольких источников информации.

## 12.6. В поисках стандартов

Стандарты гарантируют, что разные компьютеры, сети и приложения могут взаимодействовать друг с другом, образуя систему. Но что представляет собой стандарт? *Стандарт* — это официально установленная методика выполнения специфических задач или намерений в рамках данной дисциплины или технологии. С помощью стандартов можно получать на экране телевизора видеоизображения от различных



телеканалов, просматривать на видеомагнитофоне кассеты от производителей из разных стран и т. д. Стандарты позволяют использовать сеть в практических целях.

Есть несколько организаций, работающих над стандартами в определенных сферах деятельности. Например, Institute of Electrical and Electronics Engineers (IEEE, Институт инженеров по электротехнике и электронике) работает в области стандартизации сетевого оборудования, American National Standards Institute (ANSI, Национальный институт стандартизации США) создает стандарты языков программирования таких, например, как COBOL и SQL, International Organization for Standardization (ISO, Международная организация по стандартизации) разрабатывает эталонную модель Open Systems Interconnection (OSI, взаимодействие открытых систем) для совместимости коммуникаций сетевых систем.

Конечно, есть разница между разработкой стандартов и их применением. На самом деле, стандарты подразделяются на две группы — полученные на основе общего соглашения групп членов организации и разработанные в угоду требованиям рынка. Например, примером стандарта, выработанного на основе соглашения, является спецификация ANSI COOBOL, которая заставила программистов приспособиться к нему. К стандартам, которые появились благодаря рынку, можно отнести интерфейс Windows, являющийся фактическим стандартом графического интерфейса пользователя. Точно так же сетевую операционную систему фирмы Novell можно считать сетевым стандартом.

Стандарты играют важную роль в успехе клиент/серверных систем. Однако в клиент/серверной среде используется множество конкурирующих стандартов. Вот лишь несколько примеров того, с чем сталкиваются проектировщики.

- ❑ *Клиентские операционные системы.* Стандартом де-факто является линейка Microsoft Windows 95/98/2000. К другим конкурирующим системам относятся OS/2, Apple Mac OS и различные реализации UNIX, включая Linux.
- ❑ *GUI клиента.* Фактическим стандартом является Windows 98/2000. Конкурирующими GUI являются OS/2 Presentation Manager, Macintosh и множество графических интерфейсов для UNIX, например, Motif и OpenLook.
- ❑ *Серверные операционные системы и сетевые протоколы.* Серверу требуется операционная система, обеспечивающая многозадачные функции и доступ к большим ресурсам (память, жесткий диск или сетевая операционная система). Борьба вокруг сетевых операционных систем только разгорается и новые операционные системы предоставляют все новые сетевые возможности, встроенные непосредственно в ОС. В настоящее время имеется большой выбор — от UNIX, OS/2 и Windows NT Server/2000 Server для серверов БД до Novell NetWare NOS для серверов печати и файл-серверов. Доступно и множество протоколов локальной сети: TCP/IP, IPX/SPX и NetBIOS. В этой области доминируют протоколы TCP/IP и IPX/SPX. В то же время для мэйнфреймов IBM при обеспечении коммуникации между клиентами и серверами главной машины выбор только один — протокол APPC.
- ❑ *Промежуточное программное обеспечение, ППО.* Какой стандарт необходимо использовать для доступа к базе данных? Самые популярные стандарты — ODBC и Integrated Database Application Programming Interface (IDAPI, встроенный интерфейс базы данных для прикладных задач). Стандарт ODBC используется для до-

ступа к базам данных в Windows-приложениях. Стандарт IDAPI для доступа к базам данных представлен корпорацией Borland. Для систем IBM с базой данных на мэйнфреймах в качестве ППО при разработке клиент/серверных приложений используется Distributed Relational Database Architecture (DRDA, архитектура распределенной реляционной базы данных). Поскольку борьба за лидерство среди стандартов доступа к данным далека от завершения, некоторые разработчики применяют инструментальные средства, позволяющие использовать любые стандарты доступа к БД. Такие продукты, как Q+E Database Library от компании Q+E Software (Северная Каролина), предоставляют возможность доступа к множеству баз данных как с помощью их собственных драйверов, так и с помощью драйверов ODBC. Такое ППО позволяет программистам не вникать в подробности войны стандартов. Один из самых новых стандартов — продукт OLE DB корпорации Microsoft, обеспечивающий доступ к данным в операционных системах Windows.

На рис. 12.13 представлены некоторые опции, доступные разработчикам клиент/серверных систем. В конечном счете, цель состоит в том, чтобы иметь набор функций, позволяющий системам взаимодействовать вне зависимости от выбора, сделанного в этом списке.

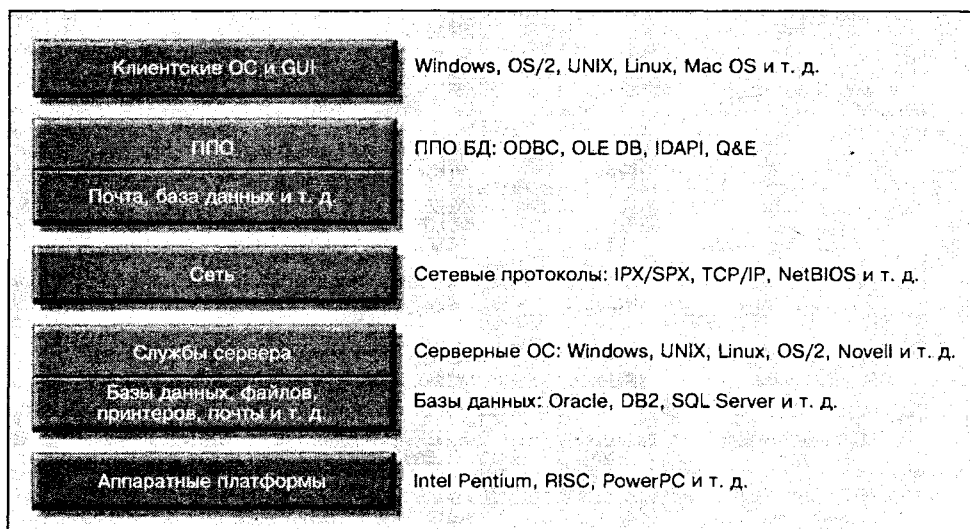


Рис. 12.13. Опции систем клиент/сервер

В конце концов, стандарты должны разрабатываться с целью обеспечения способности систем к взаимодействию на всех уровнях. Последние достижения технологии сняли некоторые преграды на пути интеграции систем, что поставило развитие клиент/серверных систем на новый уровень, совсем недавно казавшийся недостижимым: стандартные системы, работающие совершенно одинаково с разными операционными системами, графическими интерфейсами пользователя, сетями и аппаратными платформами.

## 12.7. Клиент/серверные базы данных

На сегодняшний день система управления базой данных (СУБД) — центральная часть большинства клиент/серверных систем. Для нормальной работы клиент/серверные СУБД должны обеспечивать:

- прозрачный доступ к данным для многих и разнородных клиентов, независимо от аппаратной, программной и сетевой платформы клиентского приложения;
- доставку по сети клиентских запросов к серверу баз данных (с помощью SQL);
- обработку клиентских запросов на локальном сервере;
- пересылку клиенту по сети только результатов SQL-запроса

Клиент/серверные СУБД уменьшают нагрузку на сеть, поскольку клиенту возвращаются только строки, удовлетворяющие условию запроса. Поэтому ресурсы клиентского компьютера могут выполнять другие системные задачи, например, управлять пользовательским интерфейсом. Клиент/серверные СУБД отличаются от других СУБД по месту обработки данных, по набору данных, пересылаемых клиенту по сети. Однако далеко не всегда клиент/серверные СУБД используют методику распределения данных.

Клиент/серверные системы в корне меняют подход к обработке данных. Данные могут храниться на одном сайте или на нескольких сайтах. Когда данные хранятся на нескольких сайтах, клиент/серверные БД очень напоминают распределенные БД (см. гл. 10). Распределенные клиент/серверные системы баз данных (СУРБД) должны обладать следующими свойствами:

- размещение данных прозрачно для пользователя. Данные могут размещаться на локальном ПК, на сервере подразделения или на мэйнфрейме в любой точке страны. Данные могут также распределяться по нескольким местам и по различным базам данных, использовать одну и ту же или даже разные модели данных. СУБД должна обеспечивать управление распределением данных по нескольким узлам. СУБД должна обеспечивать прозрачность распределения данных (см. разд. 10.6). Данные в БД могут разделяться несколькими способами (см. разд. 10.11). Фрагментированные данные могут размещаться несколькими различными способами (см. разд. 10.13) и могут реплицироваться на нескольких сетевых узлах (см. разд. 10.12). Сеть может быть локальной (LAN), региональной (MAN) или глобальной (WAN). Пользователю не нужно знать, где размещаются данные, как их получить или какой протокол используется для их получения;
- конечный пользователь может получать доступ к данным и манипулировать ими в любой момент времени и различными способами. Доступность данных повышается, поскольку конечные пользователи могут получать доступ к ним напрямую и без труда, как правило, выполнив несколько щелчков мышью в среде графического интерфейса. Конечные пользователи могут манипулировать данными несколькими способами, в зависимости от их информационных потребностей. Например, одному пользователю необходимо получить отчет в обычном формате, в то время как другому необходимо графическое представление. Мощные приложения, имеющиеся на стороне пользователя, позволяют получать доступ к данным и манипулировать ими способами, о которых раньше можно было толь-

ко мечтать. Запрос данных обрабатывается на стороне сервера, форматирование и представление данных выполняются на стороне клиента;

- обработка данных (поиск, хранение, проверка, форматирование, представление и т. д.) распределяется между несколькими компьютерами. Например, пусть распределенная клиент/серверная система используется для доступа к данным от трех СУБД, расположенных на различных сайтах. Когда пользователь запрашивает отчет, клиентская интерфейсная программа создает SQL-запрос к серверу СУБД. Сервер БД сам определит местонахождение данных, извлечет их из разных мест, соединит и перешлет клиенту. В этом случае доступ к данным и их извлечение выполняются на трех различных компьютерах. Интерфейсное приложение клиента берет на себя заботу о форматировании данных и их представлении на локальном ПК.

Различия между клиент/серверными системами и СУРБД иногда весьма расплывчаты. Клиент/серверная система распределяет обработку данных между несколькими сайтами. В СУРБД данные также распределяются по нескольким сайтам. Другими словами, клиент/серверные системы и СУРБД выполняют зачастую перекрывающиеся функции. Фактически в СУРБД распределенная обработка используется для получения доступа к данным, находящимся на разных сайтах. Поэтому СУРБД очень напоминает клиент/серверную реализацию.

## 12.8. Стили архитектуры "клиент/сервер"

Поскольку распределение логических компонентов приложения является первейшей характеристикой клиент/серверной системы, разработчик клиент/серверных систем должен дать ответ на два ключевых вопроса<sup>4</sup>.

- Как выполнить это распределение?
- Где в системе должны храниться результаты этого распределения?

Чтобы ответить на эти вопросы, мы должны исследовать различные уровни логического распределения обработки данных в приложениях.

Для того чтобы уметь распределять логические компоненты приложения, прежде всего, мы должны их выявить (рис. 12.14). Поскольку предметом этой книги является часть информационной системы, непосредственно связанная с базой данных, мы чаще всего используем слово "данные" по отношению к этим компонентам. Однако чтобы представить такие логические компоненты сервисов, как печать и факс, можно просто подставить на место слова "данные" слово "сервис".

На рис. 12.14 показано, что логику приложения можно разделить на три основных компонента: ввод/вывод, обработка и хранение.

- Компонент *Ввод/Вывод* (input/output) осуществляет форматирование и представление данных на устройстве вывода, например, экране, и управляет вводом данных конечным пользователем через устройство ввода, например, клавиатуру. Например, логика ввода представляет экранное меню, ожидает ввода данных

---

<sup>4</sup> См. подробности у Herb Edelstein "Unraveling Client/Server Architectures", DBMS, стр. 34—42, 7(5), май 1994, и у William Semich, "Where Do Client/Server Apps Go Wrong?", Datamation, стр. 30—36, 40(2), январь 21, 1994.

пользователем, а затем реагирует на введенные данные. (В рамках этого компонента ввода/вывода приложение использует логику представления для управления графическим интерфейсом пользователя и форматированием данных.)

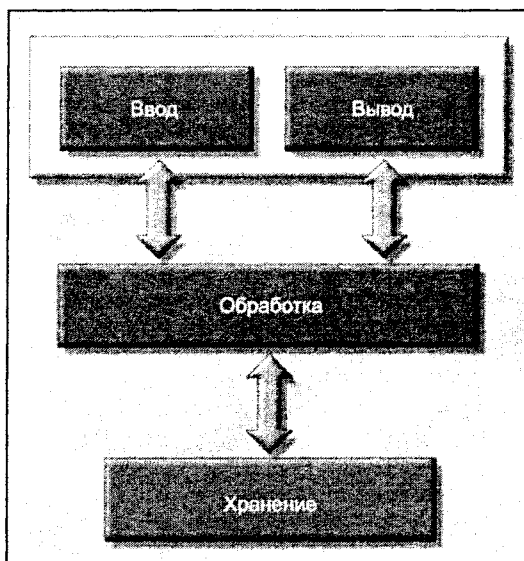


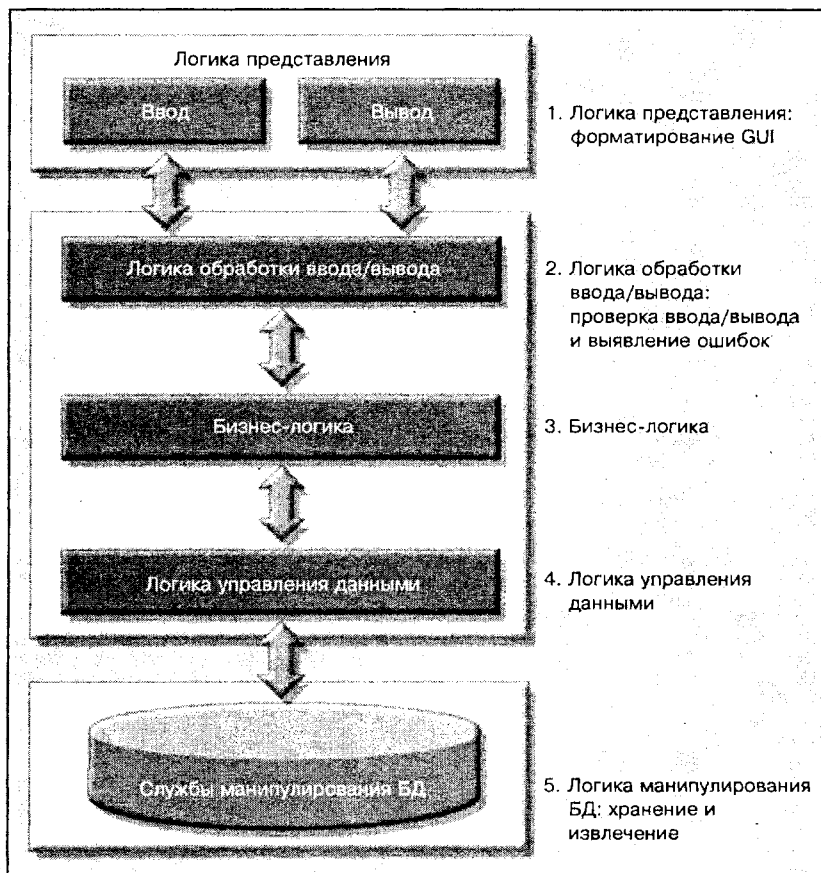
Рис. 12.14. Логические компоненты приложения

- Компонент *Обработка* связан с кодом приложения, выполняющим проверку данных, выявление ошибок и т. д. Логика обработки данных представляет собой бизнес-правила и правила управления данными по извлечению и хранению информации. Например, логика обработки данных "знает", что транзакции продаж создают запись счета-фактуры, обновляют склад и записывают данные на счет клиента. Логика обработки данных выполняет несколько функций, включая управление вводом и выводом, слежение за выполнением бизнес-правил, управление информационными потоками предприятия и отражение реальных транзакций в компьютерной базе данных. Поэтому компонент *Обработка данных* можно разделить на три функциональных подкомпонента:
  - логика обработки ввода/вывода — управляет проверкой ввода данных и выявлением основных ошибок;
  - бизнес-логика — относится ко всему коду, представляющему бизнес-правила;
  - логика управления данными — определяет, какие данные необходимы для каждой бизнес-транзакции. Например, транзакциям продаж могут потребоваться данные по поставщику, клиенту и товару.
- Компонент *Хранение* использует логику манипулирования данными для фактического хранения и извлечения данных на физических устройствах. Например, логика манипулирования данными используется при получении доступа к файлам и проверке целостности данных.

Говоря кратко, три основных логических компонента клиент/серверного приложения можно подразделить на пять основных функциональных логических компонентов:

- ☐ логика представления;
- ☐ логика обработки ввода/вывода;
- ☐ бизнес-логика;
- ☐ логика управления данными;
- ☐ логика манипулирования данными.

На рис. 12.15 представлены эти пять функциональных компонентов, составляющих основу разделения логики обработки приложения в клиент/серверной модели.



**Рис. 12.15.** Функциональные логические компоненты приложения

Хотя не существует методики, устанавливающей точное распределение логических компонентов между клиентами и серверами (см. рис. 12.15), в качестве основы

можно использовать принципы архитектуры "клиент/сервер" по распределению обработки данных (автономность, максимальное использование ресурсов, способность к взаимодействию), а также аппаратную и программную независимость.

Итак, где же разместить каждый из компонентов? Возможно, за исключением логики представления, обычно размещаемой на стороне клиента, каждый из оставшихся компонентов может быть размещен на сервере, таким образом, становясь сервисом для каждого из клиентов. При таком размещении даже систему с мэйнфреймом можно считать клиент/серверной средой. Можно считать, что система с мэйнфреймом напоминает клиент/серверную реализацию, в которой мэйнфрейм предоставляет сервисы неинтеллектуальному терминалу. Однако если целью является создание распределенной среды, система с мэйнфреймом обладает меньшими преимуществами по сравнению с действительно распределенной клиент/серверной архитектурой, в которой клиенты совсем необязательно должны зависеть от единственного сервера. К тому же архитектура с мэйнфреймом не допускает распределения различных функциональностей, потому что сервисы не могут быть вынесены за пределы главной машины. (Мы не берем в расчет случаи, когда мэйнфреймы являются составной частью клиент/серверной схемы!) Фактически в среде с мэйнфреймом невозможно разделять логические компоненты обработки данных: на стороне клиента может храниться только логика представления. Плата за такую жесткую схему высока, поскольку, как мы уже несколько раз демонстрировали в этой главе, наибольшая выгода от распределения процессов обработки получается только в том случае, когда логика обработки данных разделена между сервером (серверами) и клиентом (клиентами).

Комбинирование размещения отражает различные стили вычислений. Например, все компоненты в обычном домашнем компьютере размещаются на одном ПК. В системе с мэйнфреймом на клиентской стороне находятся только компоненты представления данных, в то время как вся остальная обработка данных выполняется на мэйнфрейме. Вряд ли имеет смысл размещать каждый компонент на отдельном сервере (за исключением логики представления), но в принципе возможно и это.

Хотя можно выбирать любую комбинацию размещения логических компонентов, практика требует, чтобы специфические сервисы были логически определены и разделены, например, можно выделить сервисы для управления файлами, печати, связи, факсимильной связи, а затем решить вопрос о размещении каждого из этих компонентов. Типичное размещение может выглядеть следующим образом:

- ☐ *логика представления* всегда размещается на стороне клиента, поскольку она предполагает взаимодействие с конечным пользователем. Графический интерфейс пользователя обычно предоставляет сервисы для служб интерфейсного приложения;
- ☐ *логика обработки ввода/вывода* может размещаться на стороне клиента или на стороне сервера. Хотя, как правило, в клиент/серверной модели она располагается на стороне клиента, ее можно разместить на стороне сервера при наличии схемы "сильный сервер/слабый клиент". (Естественно, последний сценарий обычен для модели с мэйнфреймом.) Если используется трехзвенная клиент/серверная система, промежуточные серверы обычно содержат всю логику обработки ввода/вывода, делая ее доступной для всех клиентов;
- ☐ *бизнес-логика* может также размещаться и на клиенте, и на сервере. Однако обычно ее размещают на стороне клиента. Этот логический компонент также

можно подразделить на клиентские и серверные подкомпоненты. Если используется трехзвенная клиент/серверная система, то промежуточные серверы обычно содержат все элементы бизнес-логики. При такой трехзвенной архитектуре все изменения в бизнес-логике сразу доступны всем клиентам;

- ❑ *логика управления данными* также может размещаться и на клиенте, и на сервере. Однако обычно она размещается на стороне клиента или на промежуточном сервере с бизнес-логикой. Логика управления данными может также подразделяться на клиентские и серверные подкомпоненты, выполняемые ППО базы данных. Или, в случае распределенных баз данных, подкомпоненты могут размещаться на нескольких серверных компьютерах;
- ❑ *логика манипулирования данными*, как правило, размещается на стороне сервера. Однако логика манипулирования данными также может распределяться между несколькими компьютерами в среде распределенных баз данных.

Разделение и распределение компонентов обработки данных также зависит от стиля архитектуры. На рис. 12.16 показано возможное распределение компонентов обработки данных в рамках четырех основных стилей клиент/серверной архитектуры: файл-серверная модель, модель сервера базы данных, модель сервера транзакций и модель сервера приложений.

| Компонент                      | Файл-сервер |        | Сервер БД |        | Сервер транзакций |        | Сервер приложений |        |
|--------------------------------|-------------|--------|-----------|--------|-------------------|--------|-------------------|--------|
|                                | Клиент      | Сервер | Клиент    | Сервер | Клиент            | Сервер | Клиент            | Сервер |
| Логика представления           |             |        |           |        |                   |        |                   |        |
| Логика обработки ввода/вывода  |             |        |           |        |                   |        |                   |        |
| Логика бизнес-приложения       |             |        |           |        |                   |        |                   |        |
| Логика управления данными      |             |        |           |        |                   |        |                   |        |
| Логика манипулирования данными |             |        |           |        |                   |        |                   |        |

**Рис. 12.16.** Разделение функциональной логики для четырех стилей клиент/серверной архитектуры

На рис. 12.16 обратите внимание, что серверная сторона предоставляет сервисы нескольким клиентам; столбцы "Сервер" представляют один или более серверных компьютеров. Также надо принять во внимание следующее:

- ❑ стиль архитектуры с *файл-сервером* отражает схему, в которой клиент выполняет большую часть обработки данных, в то время как сервер только управляет хране-



нием и извлечением данных. Если клиентскому приложению необходимо выбрать строки из таблицы, то фактически выбор записей будет происходить на стороне клиента, а не сервера;

- логика управления данными в стиле архитектуры *с сервером баз данных* разделяется между клиентским и серверным компьютерами. Например, клиент посылает SQL-запрос серверу: сервер выполняет его локально и возвращает только нужные клиенту строки. Помните, что в системе может быть несколько серверов, и клиент может получать доступ к нескольким серверам одновременно. Если клиентское приложение выполняет транзакцию, требующую доступа к нескольким серверам, то клиентский компьютер должен иметь доступ ко всей управляющей информации этой транзакции. Поэтому каждая SQL-транзакция должна переходить от клиента к серверу, что увеличивает сетевой трафик;
- стиль архитектуры *с сервером транзакций* позволяет совместно использовать информацию о транзакции и клиенту, и серверу. Например, если серверная сторона имеет некоторые сведения о транзакции, то часть бизнес-логики может быть размещена на сервере. Такой стиль архитектуры предпочтителен, когда сведения о транзакции известны заблаговременно (т. е. запрос не является нерегламентированным) и не очень быстро изменяются. В таком случае часть бизнес-логики хранится на сервере в форме SQL-кода или другого специфичного языка СУБД. Такой хранимый код, обычно называемый *хранимой процедурой* (см. гл. 5), проверяется, компилируется и хранится в СУБД. Клиентское приложение просто вызывает хранимую процедуру, передавая ей необходимые параметры для выполнения. Поэтому по сети не передается никакой код, и сервер транзакций может подключаться к нескольким серверам баз данных;
- архитектура *с сервером приложений* позволяет получить все преимущества клиент/серверных вычислений, когда клиентские компьютеры не обладают достаточной мощностью для выполнения каких-то клиент/серверных приложений. Этот стиль архитектуры позволяет размещать любое приложение на мощном компьютере, называемом *сервером приложений*, выполнять его там и использовать совместно несколько менее сильными клиентами. В этом случае вся обработка данных выполняется на стороне сервера приложений, а клиентские компьютеры выполняют только представление результатов работы приложения. Например, предположим, что несколько компьютеров IBM PC/486, на каждом из которых имеется 640 Кбайт оперативной памяти, подключены к серверу приложений (например, Pentium 4) и на них необходимо выполнять приложения Windows 2000 (архитектура Windows NT). Очевидно, что на отдельном клиентском компьютере невозможно выполнить такое приложение, но благодаря серверу приложений они могут получать доступ к приложениям Windows NT и выполнять их. Стиль архитектуры с сервером приложений особенно предпочтителен, когда требуется удаленное управление компьютерами по сети или когда сотрудникам необходимо получать доступ к настольному офисному компьютеру из дома по модему.

Использование одного из стилей архитектуры не мешает одновременно использовать и другой стиль. На самом деле, можно создать несколько уровней серверов для последовательного выполнения серверных процессов. Например, какой-то сервер приложений может получить доступ к серверу транзакций, который в свою очередь

может получать доступ к нескольким серверам базы данных. Другими словами, в одной сети может иметься несколько стилей клиент/серверных вычислений, работающих одновременно. Такая гибкость требует тщательного планирования клиент/серверной сетевой инфраструктуры с тем, чтобы она смогла удовлетворять разнообразным информационным требованиям. Возможность клиент/серверных систем работать с несколькими серверами позволяет им функционировать так же надежно, как интегрированные платформы, в которых объединены персональные компьютеры, мини-компьютеры и мэйнфреймы.

### Примечание

Отметим, что сервер Web-приложений представляет новый компьютерный стиль, объединяющий все архитектурные стили, представленные в этой главе. Web-сервер действует так же, как файл-сервер, передавая файлы клиенту на выполнение. В то же время Web-сервер действует как сервер транзакций, координируя доступ к базе данных нескольких клиентов. Web-сервер также может предоставлять статус управления сеансом каждому клиенту, когда он получает доступ к серверу, фактически выполняя роль сервера приложений (мы подробно исследуем модель сервера Web-приложений в гл. 15).

## 12.9. Проблемы реализации клиент/серверных систем

Реализация клиент/серверных систем является сложной задачей. Разработка таких систем значительно отличается по стилю и исполнению от методов разработки традиционных информационных систем. Например, подход к разработке систем, ориентированных на мэйнфрейм, на основе традиционных языков программирования вряд ли позволит создать эффективную клиент/серверную систему, которая предполагает использование разнообразного аппаратного и программного обеспечения. Кроме того, современные конечные пользователи стали более требовательны и, скорее всего, больше знают о компьютерных технологиях, чем пользователи эпохи "до персональных компьютеров". Поэтому руководители отдела MIS стараются соответствовать современному уровню развития компьютерной технологии и использовать последние технологии, базирующиеся на множестве платформ, использующие различные GUI, множество сетевых протоколов и т. д. Кроме того, руководители отдела MIS должны сдерживать напор потока быстро разрабатываемых приложений, а также решать проблемы, связанные с повышением автономности конечных пользователей в управлении информацией.

В этом разделе мы рассмотрим некоторые организационные и технические проблемы, возникающие при разработке и реализации клиент/серверных систем. Мы начнем с изучения различий между клиент/серверной моделью и традиционными системами обработки данных. Затем мы исследуем проблемы управления, возникающие при использовании клиент/серверной модели. Потом мы рассмотрим некоторые важные технические проблемы. И, наконец, мы разработаем основную схему, позволяющую нам подойти к разработке и реализации клиент/серверной системы.

### 12.9.1. Клиент/серверные системы и традиционные системы обработки данных

Вам уже известно, что новые модели клиент/серверных вычислений технически становятся намного более сложными, чем традиционные системы обработки данных, поскольку разработчик может использовать различные платформы, операционные системы и сети. И все же основные технические характеристики клиент/серверной модели вряд ли объясняют их столь высокую популярность. Важно обратить внимание на то, что клиент/серверная модель в корне меняет сам подход к основным проблемам обработки информации. Иначе говоря, влияние клиент/серверной модели отнюдь не исчерпывается только смелыми технологическими решениями.

Клиент/серверные вычисления повышают доступность информационных систем, меняя подход к работе и обеспечивая необходимыми знаниями конечных пользователей, которые требуют информационной автономности. Пользователи при необходимости получить какую-то информацию традиционно полагались на отдел MIS, однако в настоящее время они могут получать необходимые сведения самостоятельно, подключаясь к общему источнику информации и создавая собственные запросы и отчеты для поддержки своих решений.

Этот новый взгляд на мир информации создает некий парадокс. С одной стороны, конечные пользователи провозгласили свою независимость от отдела MIS, но, с другой стороны, они же попали в сильную зависимость от клиент/серверной инфраструктуры (серверы, сети, ППО и клиентские интерфейсные программы), которая управляется (что неудивительно) отделом MIS.

Очевидно, что клиент/серверные вычисления сильно изменили традиционный подход к обработке данных.

- *От частных систем к открытым системам.* Традиционная архитектура обработки данных базировалась на единственном поставщике. Интегрированные продукты, связанные с множеством производителей, с трудом вписывались в такую архитектуру. В новых клиент/серверных конфигурациях системы должны легко объединяться, т. е. системы должны быть открыты для других систем.
- *От обслуживающего кодирования к анализу, проектированию и сервису.* В традиционной инфраструктуре с мэйнфреймом основной задачей отдела MIS было обслуживание приложений. Хотя клиент/серверные системы тоже нуждаются в существенной поддержке, отдел MIS, в ведении которого находятся подобные системы, значительно больше времени отводит на поддержку функциональных возможностей для конечных пользователей. В жизненном цикле разработки традиционных систем основное внимание уделялось кодированию редко используемых приложений и обслуживанию. Клиент/серверная среда изменяет методы работы программистов, позволяя им использовать сложные языки четвертого поколения, CASE-системы и другой инструментарий разработки, освобождая их от рутинного кодирования. За использование этого нового инструментария приходится расплачиваться временем, которое программисты тратят на системный анализ и проектирование, поскольку стоимость ошибки на этом этапе возрастает. (Кстати, автономность конечного пользователя приводит к тому, что ошибок становится значительно больше.) Короче говоря, центр тяжести переносится с кодирования на проектирование.

- *От локализации данных к распространению данных.* Вместо того чтобы делать упор на централизованное хранение данных и управление информацией, отдел MIS в клиент/серверной конфигурации должен сконцентрироваться на повышении эффективности доступа к данным конечных пользователей.
- *От централизованного стиля к распределенной форме управления данными.* Обычно управление данными в традиционных системах с мэйнфреймом было сильно структурировано и предполагало использование строгих процедур. Новая клиент/серверная среда требует более гибкого стиля в управлении информацией. Клиент/серверные вычисления, характеризующиеся децентрализованным подходом к принятию решений и управлением, переносят центр тяжести на решение информационных проблем конечного пользователя и клиентов.
- *От вертикального негибкого стиля к более горизонтальному, гибкому организационному стилю.* Традиционные структуры отдела MIS будут уплотнены. Прямой доступ к данным дает конечным пользователям большую информационную независимость от отдела MIS. Следовательно, отдел MIS должен изменить или реструктурировать свою деятельность с тем, чтобы обучить сотрудников работе на различных ПК, в сети и использованию графического интерфейса.

Изменение среды обработки данных клиент/серверными системами можно также оценить, исследуя компоненты информационных систем.

- *Оборудование.* Информационные системы теперь не зависят от единственного поставщика, вместо этого они, как правило, объединяют множество различных аппаратных платформ.
- *Программное обеспечение.* Традиционные системы состоят из программ, написанных на процедурных языках третьего поколения, таких как Cobol, Fortran, и поддерживающих только текстовый интерфейс. Вся обработка выполняется на мэйнфрейме. Новые клиент/серверные системы появились в результате интеграции многих программ, в которых используются графический интерфейс, базы данных, сетевое программное обеспечение и связь. В новых системах обработка приложений разделена на несколько взаимосвязанных подкомпонентов. Обычно эти системы создаются с использованием таких языков программирования, как Visual Basic, C++ и Java.
- *Данные.* Традиционно данные размещались централизованно, в одном хранилище. В новых системах данные, как правило, распределяют по нескольким компьютерам, размещая данные ближе к конечному пользователю. Кроме того, возможно использование данных различного формата (звук, изображение, видео, текст и т. д.).
- *Процедуры.* Традиционные системы базировались на жестких, достаточно сложных централизованных процедурах. В новых распределенных системах происходит обмен имеющимися процедурами, что повышает гибкость системы и делает ее децентрализованной.
- *Люди.* Клиент/серверные вычисления меняют роль человека и его функции. Для работы в условиях новых технологий требуются соответствующие знания и навыки. Чтобы не отставать от жизни, необходима организация обучения и переподготовки кадров. Это касается не только отдела MIS, но и всей организации.

## 12.9.2. Администрирование

Мы рассмотрели, как клиент/серверные системы изменили стиль обработки информации и как эти изменения повлияли на предприятие в целом. Теперь рассмотрим некоторые административные проблемы, возникающие в клиент/серверных системах и связанные с управлением несколькими платформами, аппаратным обеспечением, сетями, операционными системами и средой разработки, взаимодействием с несколькими поставщиками.

- *Управление и поддержка инфраструктуры коммуникаций.* Одна из наиболее сложных проблем в клиент/серверных системах — управление коммуникационной инфраструктурой (сеть, оборудование и программное обеспечение). Руководители имеют дело с несколькими уровнями сетевого оборудования, чтобы обеспечить нормальную совместную работу аппаратуры, полученной от различных поставщиков. Ситуация еще более усложняется тем, что в настоящее время нет надежных инструментальных средств управления интегрированными клиент/серверными сетями. Администраторы систем с мэйнфреймом часто опасаются изменений, вызванных внедрением клиент/серверной среды, поскольку они привыкли к интегрированным инструментальным средствам управления, имеющимся в системе с мэйнфреймом. Руководители не могут рассчитывать на эквивалентные инструментальные средства наблюдения и управления в клиент/серверных системах.
- *Управление и поддержка приложений.* Клиент/серверные приложения, как правило, распределяются по нескольким компьютерам. Все эти компьютеры могут работать под управлением разных операционных систем. Клиент/серверная система предприятия может поддерживать несколько разных GUI (Windows, Apple Macintosh) на стороне клиента, несколько операционных систем (Novell NetWare, UNIX или Windows NT на стороне сервера и Windows 98, Windows 2000 Professional и Apple Macintosh на стороне клиента) и подходящую версию компонентов промежуточного программного обеспечения. Руководители должны обеспечивать соответствие версий всех компонентов на всех уровнях: в модулях клиентских приложений, в компонентах ППО, в сетевых компонентах и на серверной стороне. К счастью, имеется программный инструментарий, позволяющий системе автоматически распространять и обновлять программное обеспечение на клиентских компьютерах по сети. Поддержку конечного пользователя можно расширить путем создания отдела поддержки (Help Desk), обеспечивающего пользователей централизованным обслуживанием. Кадровый состав и обучение сотрудников отдела Help Desk — приоритетная задача отдела MIS.
- *Управление ростом цен и скрытые издержки.* Ожидалось, что клиент/серверные системы уменьшат затраты отдела MIS. Частично это предполагалось сделать за счет роста масштабов индустрии персональных компьютеров (можно продавать сложное программное обеспечение БД и за \$299, если планируется продать 2 000 000 его копий). Несмотря на то что ожидалось значительное снижение стоимости клиент/серверной системы по сравнению с системами с мэйнфреймом, все же начальные затраты оказались очень высокими. На самом деле, затраты, связанные с использованием ресурсов (персональных компьютерных систем)

в клиент/серверных средах, оказались выше, чем предполагалось, по следующим причинам:

- тяжело и дорого набрать штат с такими разнообразными профессиональными знаниями;
- подготовка (и переподготовка) штата, занимающегося обработкой информации, управленческого персонала и конечных пользователей требует много времени и больших затрат;
- приобретение нового сложного оборудования и программного обеспечения требует больших затрат;
- установка новых процедур или адаптация старых процедур к новой системе может быть очень трудоемкой.

Начальная стоимость клиент/серверных систем должна рассматриваться как долговременная инвестиция, позволяющая сэкономить на обновлении, повышении гибкости и улучшении обслуживания клиентов. Тем не менее, скрытые затраты на обслуживание и поддержку клиент/серверных систем должны тщательно выявляться еще на стадии планирования. (Один из примеров скрытых затрат, которые руководители часто упускают из виду, — стоимость переподготовки не только сотрудников, занимающихся обработкой данных, но также руководителей и конечных пользователей. Инвестиции в переподготовку помогут свести к минимуму потрясение, вызванное свободой управления информацией в клиент/серверных вычислениях.) Итак, стоимость реализации клиент/серверных вычислений складывается из платы за:

- *руководящий персонал и изменение интеллектуального уровня сотрудников.* Работа над психологическим воздействием, оказываемым на сотрудников изменением образа действий, — это одна из нескончаемых задач. Руководители должны привлекать конечных пользователей к реализации клиент/серверной инфраструктуры. В конечном счете, эффективность системы зависит от ее правильного использования конечными пользователями. Хотя феномен GIGO (garbage-in-garbage-out — "мусор на входе — мусор на выходе") можно встретить в любой системе, именно повсеместная доступность клиент/серверных вычислений и мощь их приложений приводят к тому, что пользователи могут очень легко и быстро принять массу ошибочных решений. Руководители также должны понимать, что не все конечные пользователи одинаковы. Некоторые очень хотят больше узнать об использовании инструментальных средств SQL или графических средств создания запросов для извлечения данных или выполнения отчетов, в то время как другие при получении информации целиком полагаются на отдел MIS. Отделу MIS необходимо разработать и реализовать последовательный и прогрессивный план обучения персонала;
- *управление взаимосвязями между различными поставщиками.* В прошлом руководители отдела MIS в системах с мэйнфреймом могли просто набрать единственный номер телефона, чтобы разрешить проблемы с оборудованием и программным обеспечением. Клиент/серверные системы вынуждают руководство MIS работать с несколькими поставщиками. Поэтому руководители часто развивают партнерские взаимоотношения с поставщиками для обеспечения согласованной работы всех поставщиков.

### 12.9.3. Инструментарий разработки клиент/серверных приложений

В современной быстро меняющейся обстановке одна из наиболее важных задач — правильный выбор инструментальных средств разработки клиент/серверных приложений. На практике руководители выбирают инструментарий с долгосрочным потенциалом. Однако при выборе проекта или инструмента разработки приложений необходимо также руководствоваться системными требованиями. Как только эти требования определены, необходимо определить свойства инструментария, который вы хотели бы иметь. К инструментарию клиент/серверных приложений относятся:

- ☐ разработка на основе графического интерфейса пользователя (GUI);
- ☐ конструктор GUI, который поддерживает несколько интерфейсов (Windows, OS/2, Motif, Macintosh и т. д.);
- ☐ объектно-ориентированная разработка с поддержкой многократного использования кода;
- ☐ словарь данных с центральным хранилищем для данных и приложений;
- ☐ поддержка нескольких баз данных (реляционные, сетевые, иерархические, плоские файлы);
- ☐ доступ к данным независимо от модели данных (с помощью SQL или естественного навигационного доступа);
- ☐ удобный доступ к нескольким базам данных;
- ☐ полная поддержка SDLC (Synchronous Data Link Control, синхронное управление передачей данных) от планирования до реализации и обслуживания;
- ☐ возможность работы в группе;
- ☐ поддержка инструментария, разработанного сторонними фирмами (CASE-приложения, библиотеки и т. д.);
- ☐ возможность создания приложений на основе прототипов и в среде быстрой разработки приложений (RAD, Rapid Application Development);
- ☐ поддержка нескольких платформ (операционные системы, оборудование и GUI);
- ☐ поддержка протоколов ППО (ODBC, IDAPI, APPC и т. д.);
- ☐ поддержка нескольких сетевых протоколов (TCP/IP, IPX/SPX, NetBIOS и т. д.).

Ни одно из инструментальных средств нельзя считать лучшим. Например, далеко не во всех инструментальных средствах поддерживаются все GUI, операционные системы, ППО и базы данных. Руководство должно выбрать тот инструментарий, который соответствует требованиям разработки приложения и имеющимся людским ресурсам, а также имеющемуся оборудованию. Возможно, чтобы выполнить все требования проекта, потребуется несколько инструментов. Но выбор инструментального средства — только первый шаг. Другая задача — убедиться, что система отвечает требованиям со стороны клиента, сервера и сети.

## 12.9.4. Интегрированный подход

Разработка клиент/серверных систем основана на предположении, что такие системы окажут существенную помощь руководству в организационных вопросах. Системы, основанные на клиент/серверных концепциях, никогда не разрабатываются только потому, что имеются доступные инструментальные средства или потому, что руководство хочет использовать последние технологические достижения. Как уже говорилось, и это стоит повторить вновь, — клиент/серверные технологии это не самоцель, а лишь один из способов достижения цели.

Если тщательное изучение технических и людских аспектов клиент/серверной системы показывает, что ее использование поможет достигнуть желаемого результата, то важно, чтобы маркетинговый план был разработан *до* начала фактического проектирования и разработки клиент/серверной среды. Хотя простого рецепта для организации этого процесса нет, общая идея состоит в осмыслении клиент/серверных систем с точки зрения сферы их действия, оптимизации ресурсов и преимуществ в управлении. Говоря кратко, планирование требует объединенных действий всех подразделений организации. Если клиент/серверное решение принимается впервые, то в этих действиях можно выделить шесть этапов:

1. *Изучение инфраструктуры информационных систем.* Цель состоит в определении реального состояния доступных компьютерных ресурсов. Предполагается изучить, по крайней мере, следующее:
  - запасы оборудования и программного обеспечения;
  - детальный и наглядный список важнейших приложений;
  - подробные сведения о кадровых ресурсах (люди и их навыки).
2. *Определение клиент/серверной инфраструктуры.* Результаты первого этапа в сочетании с задачами компьютерной инфраструктуры компании являются входными данными для этапа проектирования основ инфраструктуры "клиент/сервер". Этот эскизный проект имеет отношение к основным проблемам оборудования и программного обеспечения клиента, сервера и сети.
3. *Выбор удобного момента.* Следующий этап состоит в нахождении правильной системы, на которой можно построить клиент/серверный проект. После утверждения пилотного проекта необходимо проработать его более тщательно, сосредоточившись на проблемах, доступных ресурсах и определении реальных целей. Проект должен быть описан в терминах бизнеса, а не на техническом жаргоне. После определения системы необходимо тщательно спланировать затраты на проект. Необходимо попытаться найти баланс между затратами и эффективностью системы. Также нужно обеспечить создание первоначального варианта, который обеспечит скорые и видимые преимущества. Система, которую необходимо разрабатывать в течение двух лет, при этом реальные преимущества будут получены еще через три года, абсолютно непригодна.
4. *Сквозное управление.* Сквозное управление требуется, когда вы имеете дело с внедрением новых технологий, затрагивающих всю организацию. Управление необходимо для обеспечения доступности всех ресурсов (люди, оборудование, программное обеспечение, денежные средства, инфраструктура). На практике обычно назначают консультанта или посредника по смене рода деятельности.



Основная роль этого человека состоит в упрощении процесса изменения рода деятельности людей внутри организации.

5. *Реализация.* К основным рекомендациям по реализации можно отнести:

- использование "открытого" или стандартизованного инструментария;
- способствование продолжению образования в области аппаратных средств, программного обеспечения, инструментария и принципов разработки;
- поиск поставщиков и консультантов для обеспечения подготовки специалистов определенных направлений и реализации проектов, оборудования и прикладного программного обеспечения.

6. *Обзор и оценка.* Необходимо убедиться, что системы соответствуют критериям, определенным на третьем этапе. Нужно постоянно измерять производительность системы по мере увеличения ее нагрузки, поскольку типичные клиент/серверные решения, как правило, увеличивают сетевой трафик и замедляют работу сети. Тщательное моделирование производительности сети необходимо для гарантии того, что система будет работать нормально при ужесточении требований конечных пользователей. Такое моделирование производительности должно выполняться на сторонах сервера и клиента, а также на сетевом уровне.

## Резюме

"Клиент/сервер" — это термин, используемый для описания расчетной модели при проектировании компьютеризованных систем. Эта модель основана на распределении функций между двумя типами независимых и автономных сущностей: серверов и клиентов. Клиентом называется любой процесс, которому требуются определенные сервисы от процессов сервера. Сервер — это процесс, предоставляющий требуемые сервисы клиенту. Клиенты и серверы могут располагаться на одном компьютере или на разных компьютерах, объединенных в сеть.

Изменения в структуре бизнеса, вызвавшие появление клиент/серверных систем: необходимость доступа к информации конечным пользователям всего предприятия; увеличение производительности труда сотрудников; технологические достижения микропроцессорных систем, сетевых технологий, операционных систем, графического интерфейса пользователя и сложного прикладного программного обеспечения ПК. К достоинствам клиент/серверных систем относятся уменьшение затрат на разработку и реализацию, сокращение времени на разработку, увеличение продуктивности, масштабируемость и переносимость системы и расширенные возможности размещения информации. К другим организационным преимуществам относятся гибкость и адаптируемость к изменяющимся условиям бизнеса, увеличение производительности труда сотрудников, улучшение технологии и более полное удовлетворение требований клиентов. Однако сложность разработки и использования клиент/серверных систем может свести на нет некоторые их преимущества.

Основу клиент/серверной архитектуры составляют несколько важнейших принципов.

- ☐ Независимость от оборудования.
- ☐ Независимость от программного обеспечения.
- ☐ Открытый доступ к сервисам.

- Распределение процессов.
- Стандартизация.

Клиентский процесс, как правило, базирующийся на графическом интерфейсе пользователя (GUI, Graphical User Interface), запрашивает сервисы у серверного приложения. Клиентский процесс и серверный процесс взаимодействуют при помощи промежуточного программного обеспечения (ППО). Серверный процесс может обеспечивать несколько различных типов сервисов, например, файловый сервис, сервис печати, сервис факсимильной связи, сервис связи, сервис баз данных, сервис транзакций и CD-ROM-сервис. Серверный процесс характеризуется независимостью от местоположения, оптимизацией использования ресурсов, масштабируемостью и способностью к взаимодействию с другими системами. Для иллюстрации сетевых коммуникаций используется эталонная модель OSI. К основным сетевым протоколам, используемым в настоящее время, относятся TCP/IP, IPX/SPX, NetBIOS и APPC.

ППО логически размещается между клиентом и сервером. Его основная функция состоит в отделении клиента от тонкостей работы сетевых протоколов и сервера. В основном ППО используется в сфере баз данных. ППО баз данных используется для обеспечения прозрачности соединения клиентских интерфейсных приложений с серверными приложениями. ППО БД состоит из трех основных компонентов: программный интерфейс приложения (API), транслятор БД и сетевой транслятор.

Клиент/серверные базы данных предоставляют прозрачный доступ к информации нескольким разнородным клиентам независимо от оборудования, программного обеспечения и сетевой платформы, используемых клиентским приложением. Клиент/серверные СУБД отличаются от других СУБД по месту обработки данных и виду информации, пересылаемой по сети клиентскому компьютеру. Клиент/серверные СУБД освобождают клиента от локальной обработки данных (на компьютере клиента) и уменьшают сетевой трафик, поскольку клиенту возвращаются только необходимые ему строки.

Реализация клиент/серверных систем влечет за собой радикальные изменения в любой организации и этими изменениями необходимо соответствующим образом управлять. Поскольку это связано и с изменением основной идеологии доступа к информации, и со значительными изменениями в технологии, отдел MIS в частности должен работать над переходом от традиционного стиля обработки данных к стилю клиент/серверных моделей. Необходим последовательный и интегрированный подход к реализации клиент/серверных систем.

## Основные термины

APPC (Advanced Program-to-Program Communications) — усовершенствованный интерфейс связи между программами

IDAPI (Integrated Database Application Programming Interface) — встроенный интерфейс базы данных для программирования прикладных задач

ODBC (Open Database Connectivity) — открытый интерфейс доступа к базам данных

SNA (System Network Architecture) — системная сетевая архитектура

SQL\*Net

Архитектура "клиент/сервер" — client/server architecture

Двухзвенная клиент/серверная система — 2-tier client/server system

Именованные каналы — Named Pipes

Интеллектуальный терминал — intelligent terminal

Интерфейсное приложение — front-end application

Клиент — client

Клиент/серверные вычисления — client/server computing

Коммуникационное промежуточное программное обеспечение — communication middleware

Локальная сеть — local area network (LAN)

Масштабируемость — scalability

Программный интерфейс приложения — Application Programming Interface (API)

Промежуточное программное обеспечение (ППО) — middleware

Протокол DECnet — протокол сетевой архитектуры компании DEC

Протокол Internet Packet Exchange/Sequence Packet Exchange (IPX/SPX)

Протокол NetBIOS (Network Basic Input/Output System)

Протокол TCP/IP (Transmission Control Protocol/Internet Protocol), протокол управления передачей/протокол Интернета

Протокол взаимодействия между процессами — Interprocess Communication Protocol (IPC)

Сервер — server

Сервер изображений — imaging server

Серверное приложение — back-end application

Сетевая архитектура AppleTalk

Сетевая операционная система — Network Operating System (NOS)

Сетевой протокол — network protocol

Сетевой транслятор — network translator

Сильный ("толстый") клиент — fat client

Сильный ("толстый") сервер — fat server

Слабый ("тонкий") клиент — thin client

Слабый ("тонкий") сервер — thin server

Стандарт — standard

Транслятор базы данных — database translator

Трехзвенная клиент/серверная система — 3-tier client/server system

Уровень коммуникаций — communication layer

Фрейм — frame

Хранимая процедура — stored procedure

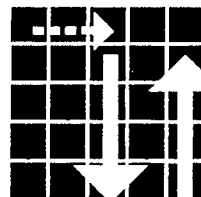
Шлюз — gateway

Электронная коммерция, е-коммерция — e-commerce

## Вопросы

1. Вычислительные системы на основе мэйнфрейма были единственным средством управления данными предприятия. Появление ПК в корне изменило подход к управлению данными. В чем различие этих двух стилей вычислений и в каком направлении развиваются вычислительные системы на основе ПК?
2. Что такое клиент/серверные вычисления и какие преимущества можно ожидать от использования клиент/серверных систем?
3. Поясните, как взаимодействуют компоненты клиент/серверных систем?
4. Опишите и поясните принципы клиент/серверной архитектуры.
5. Опишите компоненты клиента и сервера в клиент/серверной модели. Приведите примеры сервисов серверной стороны.
6. Используя эталонную модель OSI, поясните функции компонента коммуникационного промежуточного программного обеспечения.
7. Какой сетевой протокол в настоящее время считается основным?
8. Объясните, что такое промежуточное программное обеспечение и каковы его функции. Почему руководители MIS заинтересованы в таком программном обеспечении?
9. Предположим, вы рассматриваете возможность приобретения клиент/серверной СУБД. Какие характеристики вы будете рассматривать в первую очередь? Почему?
10. Опишите и сопоставьте стили архитектуры клиент/серверных вычислений, представленные в этой главе.
11. Сопоставьте клиент/серверную обработку данных и традиционную схему.
12. Обсудите и оцените следующее предложение: "Не существует нестандартных организационных проблем, связанных с внедрением клиент/серверных систем".
13. Средняя по размеру компания ROBCOR решила обновить свою компьютерную структуру. Фирма ROBCOR в течение нескольких лет представляла собой магазин по продаже мини-компьютеров, и весь ее управленческий персонал и сотрудники имеют персональные компьютеры. ROBCOR предлагает вам контракт для оказания помощи в переходе к клиент/серверной архитектуре. Объясните, как бы вы реализовали такую структуру.
14. Выявите основной стиль компьютерной инфраструктуры вашего университета. Затем определите рекомендации по ее улучшению на базе клиент/серверной стратегии. (Возможно, вам придется переговорить с секретарем факультета или с вашим куратором, чтобы выяснить, насколько имеющаяся система отвечает информационным требованиям.)

## Глава 13



# Информационное хранилище

В этой главе мы изучим:

- ☐ чем отличаются операционные данные и информация системы поддержки решений;
- ☐ что представляет собой информационное хранилище;
- ☐ что такое схема "звезда" (star schema) и как она устроена;
- ☐ какие действия необходимо предпринять для успешной реализации хранилища данных;
- ☐ что такое добыча данных (data mining) и какую роль она играет в поддержке принятия решений.

## Обзор

В предыдущих главах мы часто говорили о том, что в наши дни информация является ключевым элементом бизнеса. Поэтому хранение данных и управление ими становятся центральной частью проектирования и реализации баз данных. Ранее неоднократно упоминалось, что в хорошем проекте базы данных контролируется избыточность данных и, следовательно, устраняется (или, по крайней мере, сильно уменьшается) возможность деструктивных аномалий данных. Хороший проект базы данных является основой для эффективных операционных баз данных, с помощью которых можно отслеживать клиентов, операции по продажам, вести учет материальных ценностей и исследовать другие интересующие нас объекты. Операционные базы данных — основа успеха любой компании.

Ранее было показано, что, в конечном счете, основная цель сбора, хранения и управления "полезными сведениями" состоит в получении упорядоченной информации, составляющей основу дееспособного проекта. Для упрощения процесса разработки проекта были разработаны системы поддержки решений — системы DSS (Decision Support System). Для повышения эффективности работы системы DSS оснащены графическим интерфейсом, позволяющим ответственным работникам творчески подходить к проблемам бизнеса, их анализу и пониманию. Однако сегодня требования к информации стали настолько сложными, что даже с помощью систем DSS стало трудно извлекать необходимую информацию из структур данных, которые обычно хранятся в операционной БД. Поэтому была разработана новая технология хранения информации, называемая *хранилищем данных* (data warehouse). Хра-

нилища данных извлекают или получают данные из операционных баз данных, а также из других источников, тем самым образуя обширный фонд данных, из которого можно извлекать информацию. Что более важно, данные в хранилище данных хранятся в виде структур, в значительной степени упрощающих получение информации и позволяющих определять ее тип и объем, что в других случаях сделать просто невозможно. Неудивительно, что в последние несколько лет хранилища данных стали основным источником информации для современных систем DSS.

Одновременно с появлением концепции хранилища данных прогресс в областях аппаратных средств, клиент/серверных технологий и программного обеспечения СУБД сделал возможной разработку нового компонента систем DSS, называемого *системой оперативного анализа данных* (OLAP, on-line analytical processing). Система OLAP предоставляет современные инструментальные средства анализа информации (такие, например, как многомерный анализ данных) для извлечения информации из хранилищ данных, реляционных СУБД и/или многомерных СУБД. Независимо от используемого подхода к структурированию информации, лучше всего полагаться на данные, полученные из хранилища данных какого-либо типа. Извлечение и анализ данных упрощаются при использовании средств *автоматизации систем добычи данных* (DMS, Data Mining System), которые наряду с системой OLAP играют важнейшую роль в системах поддержки решений (DSS). Для выявления связей данных были разработаны процедуры анализа информации, подготовившие почву для раскрытия новых возможностей в современном бизнесе.

В этой главе исследуются основные концепции и компоненты, а также различные архитектурные решения хранилища данных. Здесь будут описаны инструменты, позволяющие сделать работу с хранилищем данных эффективной и удобной.

## 13.1. Необходимость анализа информации

Организации растут и процветают, когда они точнее понимают окружающую их действительность. Обычно руководство предприятия должно отслеживать ежедневные транзакции, чтобы оценить эффективность деятельности предприятия. Подключаясь к операционной базе данных, руководство может разработать стратегию, позволяющую достичь поставленной цели. Кроме того, анализ данных может дать информацию для решения краткосрочных тактических вопросов, например: "Срабатывает ли наше стимулирование сбыта? Какой процент рынка контролируется предприятием? Привлекаем ли мы новых клиентов?" Тактические и стратегические решения формируются в условиях постоянного давления внешних и внутренних сил, к которым следует отнести глобализацию, государственные и юридические структуры и, возможно, в большей степени, — высокие технологии.

Учитывая этот прессинг и постоянно растущую конкуренцию, руководители постоянно пытаются вырваться вперед за счет разработки и внедрения новых продуктов, повышения качества сервиса, увеличения доли своих товаров на рынке и т. д. Короче говоря, руководители понимают, что атмосфера бизнеса весьма динамична, и это принуждает их к быстрому реагированию на изменения в окружающей обстановке с

тем, чтобы предприятие оставалось конкурентоспособным. Другими словами, время, отводимое жизнью на принятие решений, постоянно сокращается. Кроме того, современный бизнес требует, чтобы руководство обращало внимание на все более сложные задачи, учитывающие все больше внешних и внутренних параметров. Неудивительно, что интерес к разработке систем поддержки, предназначенных для быстрого принятия решений в столь сложной обстановке, все время растет.

На разных организационных уровнях требуются разные уровни поддержки решений. Например, для удовлетворения информационных потребностей в сфере оперативного складского учета, покупок или учета счетов предназначены системы обработки транзакций, основанные на операционных базах данных. Руководители среднего звена, директора предприятий, вице-президенты и президенты компаний сосредотачивают свои усилия на принятии стратегических и тактических решений. Для принятия решений в сложной информационно-аналитической обстановке руководству необходимы подробные и точные сведения. Для облегчения принятия подобных решений информационными отделами создаются системы поддержки решений, DSS (Decision Support System).

В табл. 13.1 представлены некоторые действующие компании, где реализованы концепции хранилища данных и витрины данных (data mart — подмножество хранилища данных), системы OLAP и/или инструментальные средства добычи данных, позволившие разрешить имеющиеся проблемы, также в ней отображены способы использования инструментальных средств для получения определенных преимуществ.

**Таблица 13.1.** Решение проблем бизнеса на основе хранилища данных

| Компания                                                                                                                                                                           | Проблемы                                                                                                                                                                                                                                                                       | Полученные выгоды                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Корпорация Моеп<br>Изготовление арматуры<br>и подводов для ванн<br>комнат и кухонь<br><i>Источник:</i> корпорация<br>Cognos,<br><b>www.cognos.com</b>                              | <ul style="list-style-type: none"> <li>Получение информации ограничено и требует много времени</li> <li>Только пять сотрудников знают, как получить данные с помощью языка 3GL</li> <li>Время отклика слишком велико для оперативного принятия решения руководством</li> </ul> | <ul style="list-style-type: none"> <li>Обеспечены быстрые ответы на нерегламентированные запросы при принятии решений</li> <li>Предоставлен доступ к данным при принятии решений</li> <li>Дано полное представление об эффективности продукции и прибыли клиента</li> </ul> |
| Компания Pacific Gas<br>Transmission<br>Поставщик природного<br>газа на северо-западном<br>побережье Тихого Океана<br><i>Источник:</i> корпорация<br>Oracle, <b>www.oracle.com</b> | <ul style="list-style-type: none"> <li>Быстрые изменения на рынке из-за отмены регулирования цен</li> <li>Сокращение прибылей в традиционной сфере услуг</li> </ul>                                                                                                            | <ul style="list-style-type: none"> <li>Руководители получили возможность быстрого анализа информации</li> <li>Компания получила возможность быстро прогнозировать тенденции развития рынка</li> <li>Созданы новые службы и обновлена ценовая политика</li> </ul>            |

Таблица 13.1 (окончание)

| Компания                                                                                                                                                                                         | Проблемы                                                                                                                                                                                                                                                                                                         | Полученные выгоды                                                                                                                                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Корпорация Sega<br>Интерактивные системы и видеоигры<br><i>Источник:</i> корпорация Oracle, <a href="http://www.oracle.com">www.oracle.com</a>                                                   | <ul style="list-style-type: none"> <li>• Необходим способ быстрого анализа большого объема данных</li> <li>• Необходимо отслеживать рекламу, проценты по дивидендам и скидки, связанные с изменением цен</li> <li>• Использование электронных таблиц (типа Excel) приводит к многочисленным опечаткам</li> </ul> | <ul style="list-style-type: none"> <li>• Устранены ошибки ввода данных</li> <li>• Определены успешные стратегии рынка сбыта по доминированию интерактивных систем</li> <li>• Использован анализ продукции для выявления лучшего предложения на рынке</li> </ul>    |
| Корпорация Owens and Minor<br>Оптовые поставки медицинского и хирургического оборудования<br><i>Источник:</i> CFO Magazine, <a href="http://www.cfomagazine.com">www.cfomagazine.com</a>         | <ul style="list-style-type: none"> <li>• Потеря самых крупных клиентов, представляющих 10% годового дохода (\$360 млн)</li> <li>• Падение акций на 23%</li> <li>• Трудоемкий процесс получения информации в устаревшей системе с мэйнфреймом</li> </ul>                                                          | <ul style="list-style-type: none"> <li>• Всего за 5 месяцев — увеличение стоимости акций на 25%</li> <li>• Расширение бизнеса благодаря открытому хранилищу данных для клиентов</li> <li>• Быстрый доступ руководства к информации при принятии решений</li> </ul> |
| LA Cellular<br>Компания сотовой связи Лос-Анджелеса, ставшая в 1999 году частью компании AT&T<br><i>Источник:</i> PC Week Online, <a href="http://www.zdnet.com/eweek/">www.zdnet.com/eweek/</a> | <ul style="list-style-type: none"> <li>• Необходимость уменьшения времени отклика на запросы</li> <li>• Необходимость определения степени продвижения товара на рынке</li> <li>• Необходимость телефонных звонков клиентам (база содержит миллионы клиентов) для предложения новой продукции</li> </ul>          | <ul style="list-style-type: none"> <li>• Увеличение клиентуры на 20%, вследствие правильного информирования клиентов</li> <li>• Возможность определения степени продвижения товаров</li> <li>• Сокращение времени отклика с 14 до 1 мин</li> </ul>                 |

## 13.2. Системы поддержки решений

*Поддержка решений (decision support)* представляет собой методику (или серию методик), разработанную с целью упорядочивания информации и использования ее впоследствии в качестве основы для принятия решения. *Система поддержки решений, DSS (Decision Support System)* — это набор компьютеризованных инструментальных средств, предназначенных для обеспечения организационных решений в среде бизнеса. Системам DSS при структурировании информации обычно требуется интенсивная работа с данными.

Системы DSS используются на всех уровнях предприятия и часто разрабатываются для специфических областей (финансы, страхование, здравоохранение, банковское дело, торговля, производство и т. д.). DSS — это интерактивная система, обеспечи-



вающая возможность создания нерегламентированных запросов для поиска и отображения информации в различной форме. Например, пользователю DSS могут потребоваться следующие функции:

- сравнение относительного роста эффективности подразделения компании за некоторый промежуток времени;
- определение связи между видами рекламы и уровнем продаж. Эту зависимость можно впоследствии использовать при прогнозировании рынка продукции;
- определение относительной доли на рынке данной линейки товара.

Как система DSS помогает ответить на такие вопросы? Путем объединения ретроспективных операционных данных с бизнес-моделями, в которых отражаются деловые операции.

### Примечание

Хотя система DSS играет бесспорно важную роль в современном бизнесе, необходимо помнить, что именно *руководитель* должен инициировать процесс поддержки принятия решения, ставя соответствующие вопросы. Система DSS существует для поддержки руководства и не подменяет собой руководителя. Если руководитель не сможет сформулировать нужные вопросы, то проблема останется нерешенной, и возможность ее решения будет упущена. Несмотря на весьма широкое распространение систем DSS, все же ключевым в DSS-технологиях является человеческий фактор.

Система DSS обычно состоит из четырех основных компонентов верхнего уровня: склад данных (data store), извлечение и фильтрация данных, инструментальные средства создания запроса конечным пользователем и средства представления данных конечному пользователю.

- *Склад данных* — это, как правило, база данных DSS. Склад данных содержит два основных типа данных: коммерческую информацию и данные о бизнес-модели. Коммерческая информация извлекается из операционной базы данных и из внешних источников данных; она представляет собой мгновенный снимок (snapshot) финансового положения компании. Коммерческая информация является не просто копией операционных данных. В ней операционные данные суммируются и упорядочиваются в структуры, оптимизированные для анализа информации и быстрой обработки запроса. Из внешних источников извлекаются данные, которые нельзя получить внутри организации, но которые имеют важное значение для финансовой деятельности, например, курс акций, показатели рынка, важная маркетинговая информация (например, демографическая) и информация о конкурентах. Бизнес-модели создаются при помощи специальных алгоритмов моделирования (линейной регрессии, линейного программирования, методов матричной оптимизации и т. д.) и предназначены для расширения понимания финансовой ситуации и возникающих проблем. Например, для определения связей между видами рекламы, издержками и продажами при прогнозировании рынка в системе DSS могут применяться некоторые типы регрессионного моделирования, результаты которого могут быть использованы для выполнения анализа временных рядов.
- *Извлечение и фильтрация данных* используются при получении и проверке данных, взятых из операционной БД и внешних источников информации. Например, для определения относительной доли акций на рынке по выбранной линей-

ке товаров в системе DSS могут потребоваться данные о конкурирующих товарах. Такие данные могут находиться во внешних базах данных, предоставляемых промышленными группами или компаниями, продающими такую информацию. Судя по названию, этот компонент DSS извлекает данные, фильтрует их, выбирая значимые записи, и упаковывает данные в должном формате для последующего добавления к складу данных. Позже будет показано, что информация в базе данных DSS обладает свойствами, отличающими ее от информации, содержащейся в операционных БД.

- ❑ *Инструментальные средства запросов конечного пользователя* используются аналитиками для создания запросов, позволяющих получить доступ к базе данных. В зависимости от реализации DSS инструментальные средства создания запросов позволяют обеспечить доступ либо к операционной БД, либо (чаще всего) к базе данных DSS. Этот компонент подскажет конечному пользователю, на каких данных остановить свой выбор и как построить надежную бизнес-модель.
- ❑ *Инструментальные средства представления данных конечному пользователю* используются аналитиками для структурирования и представления информации. Данный компонент помогает конечным пользователям выбрать наиболее приемлемое представление данных, например, в форме итогового отчета, карты, круговой диаграммы или гистограммы или смешанных диаграмм. Инструментальные средства запросов и представления являются внешним интерфейсом системы DSS.

Каждый из компонентов DSS, представленных на рис. 13.1, стал причиной быстрого роста рынка специализированных инструментальных средств DSS. И благодаря развитию клиент/серверных технологий эти компоненты могут взаимодействовать с другими компонентами, формируя истинно открытую архитектуру DSS.

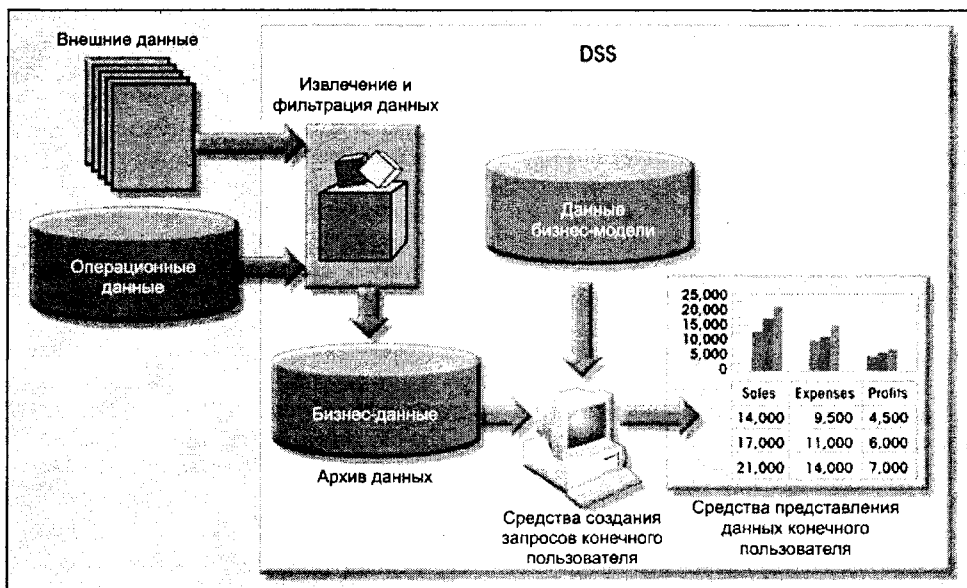


Рис. 13.1. Основные компоненты системы поддержки решений (DSS)

Несмотря на то что системы DSS используются на уровне разработки стратегии и тактики организации, *ее эффективность зависит от качества собранной информации на операционном уровне*. Операционные данные все еще редко применяются в задачах по принятию решений. В чем состоит отличие операционных данных от информации, необходимой для системы поддержки решений, мы рассмотрим в следующем разделе.

### 13.2.1. Операционные данные и данные системы поддержки решений

Операционные (эксплуатационные) данные и данные системы поддержки решений предназначены для разных целей. Поэтому не удивительно, что их формат и структура сильно отличаются.

Большая часть операционной информации хранится в реляционной базе данных, в которой структуры (таблицы) должны быть в достаточной степени нормализованы. Хранение операционных данных оптимизировано для поддержки транзакций, имеющих место при ежедневных операциях. Например, каждый раз при продаже товара необходимо сохранять данные об этой операции. Информация о клиенте, данные о наличии товара на складе и т. д. обновляются очень часто. Для того чтобы обеспечить оперативное обновление информации, операционные системы хранят данные в нескольких таблицах, в каждой из которых имеется минимум полей. Поэтому простая транзакция продажи представляется пятью или более различными таблицами (счет, строка счета, скидка, магазин, отдел и т. д.). Хотя такое представление прекрасно подходит для операционных баз данных, оно очень неудобно при формировании запросов. Например, для извлечения единственного счета необходимо соединить несколько таблиц. В то время как операционные данные фиксируют ежедневные транзакции, данные DSS придают операционным данным стратегический и тактический смысл. С точки зрения аналитика данные DSS отличаются от операционных данных по трем параметрам: временному диапазону, степени детализации и многомерности.

- *Временной диапазон (time span)*. Операционные данные представляют текущую (атомарную) транзакцию. Такие транзакции могут определять заказ на поставку, счет-фактуру, движение инвентаря и т. д.). В двух словах: операционные данные охватывают весьма короткий отрезок времени. В отличие от этого данные DSS, как правило, охватывают более протяженный отрезок времени. Руководители редко интересуются отдельным счетом-фактурой, выписанным клиенту X, их скорее заинтересует общий объем продаж за последний месяц, год или за последние пять лет. Вместо того чтобы интересоваться отдельной покупкой, сделанной клиентом, менеджеры интересуются тем, *что* обычно покупает данный клиент или группа клиентов. Иначе говоря, данные системы поддержки решений по своей природе относятся к некоторому историческому промежутку. То есть данные DSS представляют транзакции компании, начиная со вчерашнего дня, с прошлого месяца и т. д. вплоть до настоящего момента времени. Аналитики знают (или должны знать!), что счет, выписанный пару минут назад, нет смысла искать в базе данных DSS.
- *Степень детализации*. Операционные данные представляют определенные транзакции, имеющиеся в настоящий момент времени, например, покупку клиентом продукта X в магазине A. Данные DSS могут иметь различный уровень агрега-

ции — от укрупненной информации до почти атомарных данных. Это требование основано на том факте, что руководителям различного уровня требуется информация разных степеней детализации. Также возможно, что и для отдельной задачи потребуются данные различной степени детализации. Например, если руководитель должен проанализировать продажи по региону, то он должен получить доступ к данным, представляющим продажи по региону, по городам внутри региона, по магазинам в городе и т. д. В этом случае руководителю необходимо суммировать данные для сравнения продаж по нескольким регионам, но ему нужна такая структура данных, которая позволяет представить информацию в более детализированном виде (*drill down* — *детальный анализ*), т. е. иметь информацию на более низком уровне агрегации. Например, необходимо иметь возможность детально проанализировать магазины в данном регионе для того, чтобы сравнить эффективность работы магазинов по регионам. И наоборот, если вы *"сворачиваете"* (*roll up*), укрупняете детальную информацию, то можете анализировать данные на более высоком уровне (иначе говоря, *сворачивание данных* прямо противоположно их *детализации*).

- **Многомерность информации.** Возможно, это наиболее характерная черта данных систем DSS. С точки зрения аналитиков данные всегда связаны друг с другом различными способами. Например, когда анализируется продажа товара клиентам в течение данного промежутка времени, нам, вероятно, потребуется знать, сколько единиц товара X было продано клиенту Y за последние 6 месяцев. На самом деле, этот вопрос быстро разрастается, в него включаются все новые и новые данные. Например, может потребоваться информация о связи товара X с товаром Z в течение последних 6 месяцев по региону, штату, городу, магазину и клиенту. В этом случае для получения полной картины необходимо знать и время, и место операции. Фактически аналитики всегда заинтересованы в развертывании как можно более полной картины. Иначе говоря, аналитики стараются работать с данными различного рода; у них всегда многомерное представление о данных. Обратите внимание, как отличается их представление от представления о данных на уровне одиночной транзакции, типичное для операционных данных. Операционные данные связаны с атомарными транзакциями, а не с тем эффектом, который оказывают эти транзакции во времени.

На рис. 13.2 показано, как можно исследовать данные системы поддержки решения по разным аспектам (товар, регион, год и т. д.), используя различные фильтры для представления информации с различных сторон. Возможность анализа, извлечения и представления информации различными способами — одно из главных различий между данными системы поддержки решений и операционными данными текущей транзакции.

По этим трем характеристикам можно достаточно просто отличить операционные данные и данные DSS. Но проектировщиков баз данных DSS интересуют все тонкости этих различий. Для них полное выяснение свойств операционных данных и данных DSS жизненно необходимо. Рассмотрим различия между операционными данными и данными DSS с точки зрения проектировщиков систем DSS.

- Операционные данные представляют транзакции на момент их совершения в реальном времени. Данные DSS являются мгновенным снимком операционных данных на данный момент времени. Поэтому данные DSS относятся к некоторой предыстории транзакций, охватывают некоторый период времени.

- ❑ Операционные данные и данные DSS различаются по типу транзакций и объему транзакций. В то время как операционные данные связаны с транзакциями обновления, данные DSS в основном связаны с транзакциями запросов (только чтение). Данные DSS также требуют периодического обновления для загрузки новых сведений, полученных на основе операционных данных. Наконец, объем транзакции в операционных данных, как правило, очень высок по сравнению со средним или даже низким объемом транзакций данных DSS.
- ❑ Операционные данные обычно хранятся в нескольких таблицах, и хранимые данные представляют информацию только о данной транзакции. Данные DSS обычно хранятся в небольшом числе таблиц, содержащих информацию, полученную на основе операционных данных. Данные DSS не включают сведения о каждой операционной транзакции. Вместо этого данные DSS представляют собой некоторую сводку транзакций и, следовательно, в DSS хранятся данные, которые интегрированы, агрегированы и подытожены для обеспечения поддержки принятия решений.

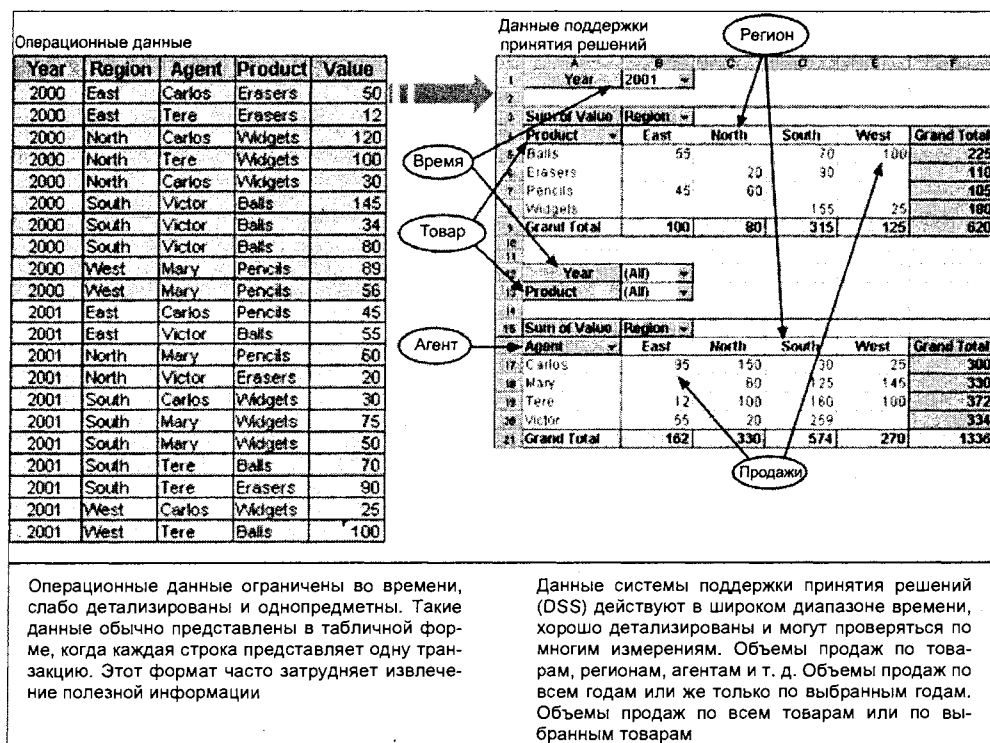


Рис. 13.2. Трансформация операционных данных в данные системы поддержки решений

- ❑ Степень обобщения данных в системе DSS очень высока по сравнению с операционными данными. Поэтому в базах данных DSS больше всего производных данных. Например, вместо того, чтобы хранить все 10 000 транзакций продаж в

данном магазине в данный день, в базе данных DSS может храниться просто общее число проданных единиц и общая сумма продажи в денежных единицах за определенный день. Данные DSS могут собираться для отслеживания таких итоговых результатов, как общая сумма продаж по данному магазину, по каждому товару и т. д. Такая обобщенная информация используется для выявления и оценки тенденций торговли, сравнения уровня продажи товаров и вообще для всего, что необходимо для принятия решений. (Насколько хорошо продается тот или иной товар? Не прекратить ли его выпускать? Сказалась ли проведенная рекламная компания на эффективности продажи товара?)

- Модели данных, используемые в операционных данных, и данных DSS различны. Повторяющиеся и быстро изменяющиеся данные в операционных БД могут стать причиной возникновения аномалии данных. Поэтому требования к данным в обычной реляционной (операционной) системе обычно требуют тщательной нормализации, что приводит к большому количеству таблиц, каждая из которых содержит минимальный набор атрибутов. В отличие от этого в базе данных DSS отсутствуют столь активные транзакции обновления, а основная цель — возможность обработки запросов. Поэтому базы данных DSS не стремятся нормализовать и включают в них небольшое число таблиц, каждая из которых содержит большое число атрибутов.
- Обработка запросов (частых и сложных) в операционных базах данных невысока, что позволяет выполнять большее количество транзакций обновления. Поэтому запросы к операционной базе данных небольшие по объему, несложные, но требуют быстрого ответа. В отличие от этого данные DSS существуют только для обслуживания запросов. Запросы к БД DSS обычно широкомасштабные, сложные, но не очень критичны к скорости ответа.
- Наконец, данным DSS свойственны очень большие объемы. Большие объемы данных появляются по следующим двум причинам: во-первых, данные хранятся в ненормализованном виде, что ведет к большой избыточности данных и явному дублированию, во-вторых, одни и те же данные можно рассматривать с разных позиций при получении различных мгновенных снимков системы. Например, данные по продажам могут храниться в связке с данными по товару, магазину, клиенту, региону, руководителям и т. д.

В табл. 13.2 приведены результаты сравнения операционных данных и данных DSS с точки зрения проектировщика.

**Таблица 13.2.** Сравнение операционных данных и данных DSS

| Свойства                       | Операционные данные       | Данные DSS                                                                  |
|--------------------------------|---------------------------|-----------------------------------------------------------------------------|
| Область распространения данных | Текущие операции          | Ретроспективные данные                                                      |
|                                | Данные в реальном времени | Мгновенный снимок данных компании<br>Временной компонент (неделя/месяц/год) |
| Степень детализации            | Высокая атомарность       | Укрупненная информация                                                      |

Таблица 13.2 (окончание)

| Свойства            | Операционные данные                                    | Данные DSS                                                                                                              |
|---------------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|
| Уровень укрупнения  | Низкий, возможность некоторого обобщения результатов   | Высокий, множество уровней обобщения                                                                                    |
| Модель данных       | Тщательная нормализация<br>В основном реляционные СУБД | Ненормализованные данные<br>Сложные структуры<br>Иногда реляционная модель, но в большинстве случаев — многомерные СУБД |
| Тип транзакции      | В основном обновление                                  | В основном запросы                                                                                                      |
| Объемы транзакций   | Большой объем обновлений                               | Периодическая загрузка и расчеты сумм                                                                                   |
| Скорость транзакций | Критична скорость обновления                           | Критично время поиска                                                                                                   |
| Активность запросов | Низкая или средняя                                     | Высокая                                                                                                                 |
| Масштаб запросов    | Небольшие запросы                                      | Широкомасштабные запросы                                                                                                |
| Сложность запросов  | Простые или средние                                    | Очень сложные                                                                                                           |
| Объем данных        | От сотен мегабайт до нескольких гигабайт               | От сотен гигабайт до нескольких терабайт                                                                                |

Различия операционных данных и данных системы поддержки решений определяют и требования к базе данных DSS. Если операционные данные не совсем соответствуют системе поддержки решений, то мы должны создать базы данных для хранения информации в формате, который наилучшим образом согласуется с системами поддержки решений. Итак, каким же требованиям должна удовлетворять СУБД, обеспечивающая хранение данных для системы поддержки решений? Ответ на этот вопрос будет дан в следующем разделе.

### 13.2.2. Требования к базе данных DSS

База данных DSS представляет собой специализированную СУБД, разработанную для быстрого получения ответа на очень сложные запросы. Упомянем четыре основных блока требований к базе данных DSS: схема базы данных, извлечение и загрузка данных, аналитический интерфейс конечного пользователя и требования к размеру базы данных.

#### Схема базы данных

Схема базы данных DSS должна обеспечивать работу со сложными (ненормализованными) данными. Как уже отмечалось, БД DSS должна содержать данные в агрегированном и укрупненном виде и, что более важно, она должна обеспечивать взаимосвязь со множеством других элементов данных. Кроме того, для выполнения

этих требований необходимо обеспечить условия, при которых запросы могут извлекать данные по заданным различным интервалам времени. Если применяется СУРБД, то это означает использование ненормализованных и даже дублирующих данных в реляционной БД DSS. Чтобы убедиться в этом, взглянем на реализацию хранения данных по объему продаж за 10 лет для одного магазина, имеющего один отдел. В этом случае данные полностью нормализованы и находятся в одной таблице, как это представлено в табл. 13.3.

**Таблица 13.3.** Десятилетняя история продаж для одного отдела (млн долларов)

| Год  | Объем продаж | Год  | Объем продаж |
|------|--------------|------|--------------|
| 1992 | 8227         | 1997 | 11 875       |
| 1993 | 9109         | 1998 | 12 699       |
| 1994 | 10 104       | 1999 | 14 875       |
| 1995 | 11 553       | 2000 | 16 301       |
| 1996 | 10 018       | 2001 | 19 986       |

Эта структура прекрасно работает, если у нас есть только один магазин с одним отделом. Однако маловероятно, что в таком простом случае есть необходимость использования системы поддержки решений. Скорее система DSS может потребоваться в ситуации, когда имеется более одного магазина, в каждом из которых несколько отделов. Для выполнения всех требований DSS база данных должна содержать информацию по всем магазинам и их отделам, а также поддерживать многомерные запросы, которые прослеживают продажи по магазинам, по отделам и по времени. Для простоты пусть имеется два магазина (А и В) и два отдела (1 и 2) в каждом магазине. Изменим и массив времени, чтобы включить данные по годам. В табл. 13.4 представлены объемы продаж по определенным условиям. Показаны полностью только годы 2001 и 1992, многоточия (...) говорят о том, что данные пропущены. По табл. 13.4 видно, что резко возросло число строк и атрибутов и что в таблице имеются избыточные данные.

**Таблица 13.4.** Ежегодный объем продаж  
(два магазина, два отдела в каждом магазине, млн долларов)

| Год  | Магазин | Отдел | Объем продаж |
|------|---------|-------|--------------|
| 1992 | А       | 1     | 1985         |
| 1992 | А       | 2     | 2401         |
| 1992 | В       | 1     | 1879         |
| 1992 | В       | 2     | 1962         |
| ...  | ...     | ...   | ...          |
| 1996 | А       | 1     | 3912         |
| 1996 | А       | 2     | 4158         |



Таблица 13.4 (окончание)

| Год  | Магазин | Отдел | Объем продаж |
|------|---------|-------|--------------|
| 1996 | В       | 1     | 3426         |
| 1996 | В       | 2     | 1203         |
| ...  | ...     | ...   | ...          |
| 2001 | А       | 1     | 7683         |
| 2001 | А       | 2     | 6912         |
| 2001 | В       | 1     | 3768         |
| 2001 | В       | 2     | 1623         |

Теперь предположим, что имеется 10 отделов в каждом магазине и 20 магазинов по стране. При этом необходимо получить доступ к ежегодным объемам продаж. Тогда мы будем иметь дело с 200 строками и 12 атрибутами для каждой строки (фактически 13 атрибутов на строку, если мы добавим итоговую сумму продаж по каждому году). Для базы данных DSS в этом случае можно использовать структуру, подобную представленной в табл. 13.5.

Итоговая информация, представленная в табл. 13.5, — это простой пример типовой структуры данных, имеющейся в базе данных DSS. В БД DSS также, вероятно, должны содержаться статистические данные, например, средний объем продаж за последнюю неделю, последний месяц, предыдущий год и предыдущий месяц.

При изучении представленных табличных структур помните, что схема базы данных DSS должна быть оптимизирована на выполнение поисковых запросов (только чтение). Для оптимизации времени выполнения запроса в СУБД должны иметься такие функции, как индексация по битовой карте (bitmap indexes) и декомпозиция данных (data partitioning) для ускорения доступа. Кроме того, оптимизатор запросов СУБД должен поддерживать ненормализованные сложные структуры, имеющиеся в базах данных системы поддержки решений.

## Извлечение и фильтрация данных

База данных DSS создается в основном путем извлечения данных из операционной БД и частично импортом данных из внешних источников. Поэтому в составе СУБД должны иметься функции извлечения и фильтрации данных. Функции извлечения данных должны поддерживать различные источники данных: плоские файлы, иерархические, сетевые и реляционные базы данных, а также различных поставщиков. Функции *фильтрации данных* должны поддерживать возможность контролировать целостность данных и обеспечивать выполнение правил проверки информации. Наконец, СУБД должна поддерживать дополнительные функции по интеграции данных, агрегации и классификации для того, чтобы фильтровать и интегрировать операционные данные в базу данных DSS. Помните, что данные в БД DSS обычно включают в себя информацию из различных внешних источников. При использовании информации из внешних источников, как правило, потребуется разрешать конфликты форматов данных.

Таблица 13.5. Итоговые ежегодные объемы продаж (20 магазинов, 10 отделов на магазин, млн долларов)

| Год  | Мага-<br>зин | Отдел 1 | Отдел 2 | Отдел 3 | Отдел 4 | Отдел 5 | Отдел 6 | Отдел 7 | Отдел 8 | Отдел 9 | Отдел 10 | Объем<br>продаж |
|------|--------------|---------|---------|---------|---------|---------|---------|---------|---------|---------|----------|-----------------|
| 1992 | A            | 1985    | 2401    | 1220    | 1401    | 1792    | 985     | 1118    | 1950    | 2541    | 1736     | 17 129          |
| 1992 | B            | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| 1992 | C            | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| 1992 | T            | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| 2001 | A            | 7683    | 6912    | 4002    | 8297    | 6509    | 5001    | 6183    | 8554    | 9989    | 4955     | 68 085          |
| 2001 | B            | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| 2001 | C            | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| 2001 | D            | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| ..   | ...          | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...     | ...      | ...             |
| 2001 | T            | 7109    | 9125    | 5506    | 6251    | 7618    | 6321    | 5194    | 8728    | 6227    | 9884     | 71 963          |

Например, в номере социального страхования могут встречаться даты в разных форматах, качественные характеристики (измерения) могут быть основаны на разных шкалах, одни и те же элементы данных могут иметь различные имена и т. д. Короче говоря, данные нужно очистить и отфильтровать, чтобы обеспечить хранение в БД DSS только необходимой и значимой информации в стандартном формате.

## Аналитический интерфейс конечного пользователя

СУБД DSS должна обеспечивать поддержку современных моделей данных и инструментов представления данных. Использование таких средств помогает аналитикам определить природу и масштабы проблем предприятия. После того как проблема выявлена, СУБД DSS должна инициировать необходимые запросы для извлечения соответствующих данных из БД DSS. При необходимости результаты запроса можно оценить средствами анализа данных, поддерживаемыми СУБД DSS. Поскольку запросы содержат критически важную информацию для ответственных лиц, их нужно оптимизировать для быстрой обработки. То есть СУБД DSS должна обеспечивать следующие функциональные возможности.

- ☐ Инструментальные средства моделирования и представления данных.
- ☐ Инструментальные средства для аналитиков.
- ☐ Генераторы запросов и средства оптимизации.

Интерфейс анализа данных конечного пользователя — один из важнейших компонентов СУБД DSS. При должной реализации аналитический интерфейс помогает пользователю осуществлять навигацию по данным, упрощая и ускоряя процесс принятия решения.

## Требования к размеру базы данных

База данных DSS имеет тенденцию к быстрому росту; объемы в сотни гигабайт и в несколько терабайт не являются чем-то необычным (как уже отмечалось, в базе данных DSS обычно содержатся избыточные и дублированные данные для упрощения извлечения информации). Поэтому СУБД должны обеспечивать поддержку *баз данных большого размера (VLDB, very large database)*. Для того чтобы корректно поддерживать VLDB, СУБД могут потребоваться современное оборудование, например, множество дисковых массивов и, что более важно, поддержка многопроцессорных технологий, таких как симметричная многопроцессорная обработка (SMP, Symmetric Multiprocessing) или массовая параллельная обработка (MPP, Massive Parallel Processors).

Как и в случае с операционными базами данных, которые мы исследовали в предыдущих главах, информационные требования определяют сферу действия компонента склада данных DSS. Поэтому компонент данных DSS разработан с целью предоставления аналитикам специфической информации.

Сложные информационные требования и постоянно растущий спрос на все более изощренный анализ информации послужили толчком к созданию нового типа информационного архива. Такой архив содержит информацию в форматах, облегчающих извлечение данных, их анализ и принятие решений. Информационный архив, дружественный системе поддержки решений (DSS), который получил название хранилища данных (data warehouse), стал основой нового поколения систем поддержки решений.

## 13.3. Хранилище данных

Билл Инмон (Bill Inmon), общепризнанный основатель *хранилища данных* (data warehouse), определяет его как *интегрированную предметно-ориентированную нестационарную долговременную базу данных* (курсив использован для подчеркивания значимости определений), которая обеспечивает поддержку принятия решений<sup>1</sup>. Для того чтобы глубже понять это определение, разберем подробно все его составляющие.

- ❑ **Интегрированность.** Хранилище данных представляет собой централизованную, консолидированную базу данных, объединяющую данные, полученные от всей организации. Поэтому хранилище данных консолидирует данные из множества разнообразных источников различных форматов. Интеграция данных предполагает проведение целого ряда мероприятий для выявления и централизации всех элементов данных. Эти мероприятия могут отнять достаточно много времени, но после их выполнения вы получите унифицированное представление обо всей ситуации на предприятии. Интеграция данных расширяет возможности принятия решений и помогает руководству лучше разобраться в работе предприятия. Это понимание поможет выявить стратегические возможности компании.
- ❑ **Предметная ориентированность.** Хранилище данных организовано и оптимизировано для получения ответов на вопросы, поступающие из различных функциональных областей в масштабах предприятия. Поэтому в хранилище данных информация организована и суммирована по различной тематике, например, объемы продаж, маркетинг, экономика, распространение товара и транспортировка. Для каждой из этих тем в хранилище данных имеется специфический раздел — товары, клиенты, отделы, регионы, движение товара и т. д. Обратите внимание, что такая форма организации данных отличается от более функциональной, ориентированной на процессы типичной организации транзакционных систем.
- ❑ **Нестационарность.** Мы уже отмечали, что данные DSS включают в себя время. В отличие от операционных данных, привязанных к текущим транзакциям, хранилище данных представляет собой поток данных во времени. Хранилище данных может даже содержать прогнозируемую информацию, полученную на основе статистических или каких-либо других моделей. Она также изменяется во времени, в том смысле, что по мере обновления данных в хранилище все зависящие от времени компоненты вычисляются заново. Например, когда в хранилище данных обновляется информация об объеме продаж за предыдущую неделю, то обновляются еженедельные, ежемесячные, ежегодные и другие зависящие от времени агрегатные данные по продуктам, клиентам, магазинам и другим переменным.
- ❑ **Долговременность.** После того как данные записаны в хранилище данных, их нельзя удалить оттуда. Поскольку данные в хранилище представляют всю историю деятельности компании, операционные данные, представляющие самые недавние сведения, также заносятся в него. Так как данные никогда не удаляются из хранилища, оно постоянно растет. Вот почему СУБД DSS должны обеспечивать поддержку многогигабайтных и даже многотерабайтных баз данных в многопроцессорных средах.

---

<sup>1</sup> Bill Inmon и Chuck Kelley, "The Twelve Rules of Data Warehouse for Client/Server World", *Data Management Review*, 4(5), май 1994, стр. 6—16

Приведенные четыре требования к хранилищу данных составляют ключевые элементы архива данных DSS. Для правильного функционирования базы данных DSS хранилище данных должно удовлетворять этим требованиям.

Данные в хранилище должны интегрироваться так, чтобы обеспечивать целостное представление обо всех организационных компонентах. Интеграция данных подразумевает, что все объекты бизнеса, элементы данных, свойства данных и системы показателей бизнеса *описаны одинаковым способом на всем предприятии*. Хотя это выглядит как требование к логической организации данных, вы удивитесь, узнав, как много существует критериев оценки эффективности продаж внутри одного предприятия; то же самое касается и других сторон бизнеса. И это только один маленький пример из числа проблем, с которыми сталкиваются профессиональные разработчики БД DSS в процессе создания хранилища данных. Кроме того, используя различные определения при описании одной и той же темы, разные отделы могут пользоваться различными способами вычислений одних и тех же элементов. Например, статус заказа в одном подразделении обозначается текстовой меткой, например, "открыт", "получено", "отменен" или "закрит", а в другом цифровой меткой типа "1", "2", "3" и "4". Статус студента в расчетном отделе может обозначаться как "freshman" (первокурсник), "sophomore" (второкурсник), "junior" (третьекурсник) или "senior" (выпускник), а на факультете компьютерных информационных систем (CIS) — как "FR", "SO" "JR" или "SR". Чтобы избежать возможных проблем, сведения в информационном хранилище должны храниться в едином формате, принятом во всех подразделениях предприятия.

Помните, что системы операционных БД ориентированы на обеспечение функциональности; т. е. главная цель таких проектов — удобство модификации данных. Например, проектировщик системы обслуживания счетов, помещая компоненты счетов в две таблицы INVOICE (счет) и INVLIN (строка счета), ставит своей целью нормализацию этих реляционных таблицы для надлежащего обслуживания бизнес-процессов. В отличие от этого хранилище данных предметно ориентировано. (Термин "предмет" относится к сущностям или элементам операционных данных, имеющим отношение к аналитикам или ко всей организации). Проектировщики хранилищ сосредотачиваются на данных как таковых, а не на процессах их модификации. (Данные в хранилище не так уж часто обновляются и изменяются, и уж тем более не в реальном времени!). Поэтому вместо хранения счетов в хранилище данных размещаются товары и сведения о клиенте, поскольку работа системы поддержки решений требует извлечения из хранилища итоговых сведений по товарам и клиентам.

Поскольку информация в хранилище данных представляет собой мгновенный снимок предыстории всей компании по всем направлениям ее деятельности, составляющая времени является критически важной. Поэтому в хранилище обязательно хранится *идентификатор времени (time ID)*, позволяющий организовать информацию по временным интервалам. Идентификатор времени используется также при формировании соответствующих итоговых результатов и вычислении агрегатных значений. Другими словами, временной компонент позволяет аналитикам добавлять при анализе решений еще одну координату — время, что позволяет оценить изменения

показателя за неделю, месяц, квартал и год. После занесения данных в хранилище им присваивается идентификатор времени, который изменить нельзя.

В табл. 13.6 продемонстрировано различие между хранилищем данных и операционной БД.

**Таблица 13.6.** Сравнение свойств хранилища данных и операционной БД

| Свойство              | Операционная база данных                                                                                                                                                                                                                                                                                             | Хранилище данных                                                                                                                                                                                                                                 |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Интеграция            | Схожие данные могут иметь разное внешнее представление или смысл. Например, номер социального страхования может храниться как ###-##-#### или как #####, условие может помечаться как T/F (true/false) или 0/1, или Y/N, или Да/Нет. Объем продаж может быть представлен в тысячах долларов или в миллионах долларов | Обеспечивает унифицированное представление для всех элементов данных с помощью общих определений и представлений для всех подразделений предприятия                                                                                              |
| Предметная ориентация | Хранение данных ориентировано на обработку. Например, данные могут предназначаться для счетов, платежей, кредитных операций и т. д.                                                                                                                                                                                  | Данные хранятся с ориентацией на конкретный предмет, что обеспечивает множественное представление данных и упрощает процесс принятия решений. Например, объемы продаж могут быть записаны по виду товара, по отделу, по менеджеру или по региону |
| Нестационарность      | Данные записываются для текущей транзакции. Например, данные продажи могут представлять собой продажу данного вида товара в данный день, например, \$342,78 на 12-АВГ-1999                                                                                                                                           | Данные записываются с учетом ретроспективы. Поэтому в распоряжение аналитиков предоставляется еще одна координата — время, и они имеют возможность сравнения параметров во времени                                                               |
| Долговременность      | Данные обновляются часто и в полном объеме. Например, складской запас обновляется после каждой продажи. Поэтому информационная среда очень подвижна                                                                                                                                                                  | Данные нельзя изменять. Данные только периодически добавляются из систем сбора статистических данных. После надлежащего размещения данных невозможны никакие изменения. Поэтому информационная среда относительно статична                       |

В итоге хранилище данных обычно представляет собой базу данных, предназначенную только для чтения и оптимизированную для анализа данных и обработки запросов. Обычно данные извлекаются из различных источников, а затем трансфор-

мируются и интегрируются — иначе говоря, фильтруются, — перед тем как их окончательно поместить в хранилище. Пользователи получают доступ к хранилищу данных с помощью интерфейсных программ и/или пользовательских прикладных программ, извлекающих данные в приемлемом формате. На рис. 13.3 представлен процесс создания хранилища данных на основе информации, содержащейся в операционной базе данных.

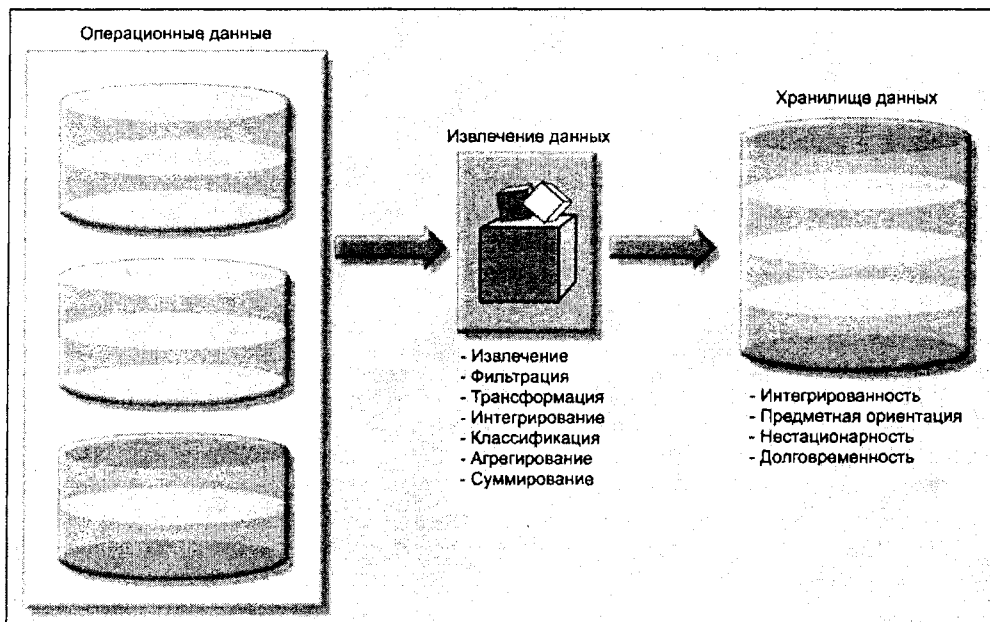


Рис. 13.3. Создание хранилища данных

Создать хорошее хранилище данных непросто, поскольку хранящаяся в нем информация, как правило, получена из разных источников, где используются различные форматы и структуры. Например, даты в одном источнике могут храниться в виде строки, в другом источнике — в юлианском формате; и даже строковый формат может быть разным. Точно так же номер социального страхования может храниться в одной базе данных в виде строки с дефисами (xxx-xx-xxxx), а в другой базе — в виде строки без дефисов (xxxxxxxxx). Кроме того, данные могут извлекаться из различных типов файлов и баз данных, а также поступать с различных платформ. Поэтому интеграция источников, типов данных и форматов в однородную среду хранилища данных может стать реальной проблемой.

Чтобы получить представление о проблемах, с которыми приходится сталкиваться при интеграции данных из множества источников, мы продемонстрируем проблемы противоречивости информации, обнаруженные в процессе проектирования университетской компьютерной лаборатории (см. гл. 7 и 8). В этом случае для формирования бюджета и выравнивания финансирования специальных проектов нам были необхо-

димы сведения данных о рейтинге использования компьютерной лаборатории различными студентами и факультетами. При этом имелись два внешних ограничения.

- Данные извлекались из вновь установленной системы считывания магнитных карт университета. Эта система функционировала на компьютере VAX, расположенном где-то в кампусе университета. Система использовала плоские файлы и содержала только информацию о дате доступа, времени, а также идентификатор студента. Система не предоставляла данные, на основе которых можно было бы получить информацию о классификации студентов или факультетов.
- Классификация студентов и данные по факультетам были размещены в другой базе данных, также основанной на плоских файлах и расположенной на другом компьютере. Стандарты университета обусловили использование различных сегментов информации этих баз данных.

В соответствии с законом Мэрфи ("все, что может испортиться, — портится") устройство считывания карт использовало структуры данных, отличные от структур студенческой базы данных. Иначе говоря, при создании хранилища данных мы столкнулись с обычным делом — различие в типах данных: информация, которую необходимо использовать при формировании бюджета, поступала из различных источников и была представлена в разных форматах. Следовательно, необходимо было создать отдельную базу данных (пока реляционную!), в которой мы могли бы легко интегрировать информацию из этих источников. Эта работа оказалась непростой. Во-первых, мы столкнулись с целым рядом бюрократических проблем. Затем мы должны были получать данные из различных систем, импортировать их в настольную базу данных, фильтровать информацию для очистки данных, интегрировать данные и, наконец, выпускать необходимые отчеты. Работа принесла свои плоды, поскольку мы получили необходимое финансирование для обеспечения деятельности лаборатории. Но мы не ожидали, что будет затрачено столько времени и сил.

Действия при создании хранилища данных лаборатории типичны для организации любого хранилища. Термин *организация хранилища данных (data warehousing)* используется для описания всех действий по созданию и обслуживанию хранилища данных.

Хотя пример с лабораторией демонстрирует, что централизованное и интегрированное хранилище данных может быть весьма привлекательным решением, дающим множество преимуществ, руководство, однако, может отказаться от такой стратегии. Создание хранилища данных требует времени, средств и значительных организационных усилий. Поэтому неудивительно, что многие компании начали организацию хранилища данных с создания более управляемых наборов данных, предназначенных для небольших групп в рамках организации. Эти небольшие архивы данных называются витринами данных (*data mart*). *Витрина данных* представляет собой небольшое, ориентированное на одну предметную область, подмножество хранилища данных, которое обеспечивает поддержку решений для небольшой группы людей.

Витрина данных привлекательна для некоторых организаций не только невысокой стоимостью и быстротой реализации, но и сравнительно невысокими требованиями к использованию технических средств и людских ресурсов. Мощные компьютеры позволяют нам обеспечить организацию систем поддержки решений для небольших групп способами, которыми централизованная система просто не обладает. К тому



же обстановка, сложившаяся в компании, может предрасположить людей сопротивляться радикальным изменениям, но совершенно благосклонно отнестись к небольшим изменениям, позволяющим облегчить принятие важных решений. Кроме того, людям на различных организационных уровнях, скорее всего, необходимы данные с различной степенью обобщения и в разной форме представления. Витрины данных могут сыграть роль тестового инструмента для компаний, которые исследуют потенциальные преимущества хранилища данных. Двигаясь по такой схеме — от витрин данных к хранилищу данных — руководители могут решить проблему создания систем поддержки решений за вполне приемлемое время (от 6 месяцев до года), по сравнению с временем, которое требуется на разработку информационного хранилища (от одного года до трех лет). Отделу информационных технологий (IT) также выгодно использовать именно такой подход, поскольку его персонал получает возможность изучить проблемы и развить профессиональные навыки, необходимые для создания полновесного хранилища данных.

Помните, что различие между витриной данных и хранилищем данных состоит только в размерах и рамках решаемых проблем, в то время как постановка задачи и требования к данным в них абсолютно одинаковы. К счастью, благодаря клиент/серверным технологиям компоненты оборудования и программного обеспечения могут также быть одинаковыми, как для витрины данных, так и для хранилища данных, что делает переход от более низкой ступени организации данных (витрина) к более высокой (хранилище) не слишком обременительным. Теперь мы рассмотрим влияние клиент/серверных технологий на развитие хранилищ данных.

### 13.3.1. Стили архитектуры DSS

В настоящее время имеется несколько стилей архитектуры базы данных DSS. Эти архитектуры обеспечивают дополнительные возможности для системы поддержки решений, некоторые из них предоставляют многомерный анализ информации. Инструментальные средства DSS, использующие многомерный анализ информации, называются *системами оперативного анализа данных (OLAP, Online Analytical Processing)* (мы будем обсуждать OLAP подробно далее в этой главе). В табл. 13.7 приведены свойства основных стилей архитектуры, которые чаще всего встречаются в среде баз данных DSS (источником везде являются операционные данные).

Может показаться, что хранилище данных это просто большая, итоговая база данных. Однако наши предыдущие рассуждения говорят о том, что это далеко не так. Полная архитектура хранилища данных включает в себя поддержку склада данных для DSS, извлечение данных и интегрирующие фильтры, а также специализированные интерфейсы представления информации. Чтобы хранилище данных стало приносить практическую пользу, в нем должна иметься унифицированная база данных в качестве основы архитектуры системы поддержки решений. То есть данные хранилища должны соответствовать единым структурам и форматам, что позволит избежать конфликта данных. На самом деле, до того, как БД DSS можно будет считать истинным хранилищем данных, ее надо привести в соответствие правилам, представленным в следующем разделе.

Таблица 13.7. Стили архитектуры DSS

| Тип системы                                                                               | Извлечение данных/процесс интеграции                                                                                                                                                                        | Архив данных DSS                                                                                                                                     | Инструментарий конечного пользователя                                                                  | Инструментарий представления данных конечного пользователя                                                          |
|-------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| Традиционная система, основанная на мэйн-фрейме                                           | Отсутствуют                                                                                                                                                                                                 | Отсутствуют                                                                                                                                          | Номинальные                                                                                            | Номинальные                                                                                                         |
|                                                                                           | Отчеты, считывание и суммирование данных напрямую из операционных данных                                                                                                                                    | Временные файлы, используемые для отчетов                                                                                                            | Предопределенные формы отчетов, базовая сортировка, суммирование и усреднение                          | Предопределенные отчеты в форме меню, только в текстовой и числовой форме                                           |
| Управленческие информационные системы с языками программирования третьего поколения (3GL) | Базовые способы извлечения и агрегирования                                                                                                                                                                  | Небольшие возможности агрегирования данных в СУРБД                                                                                                   | Все, что указано выше, плюс нерегламентированные отчеты с помощью SQL                                  | Все, что указано выше, плюс нерегламентированные запросы в виде столбцов                                            |
| Первое поколение систем DSS подразделений                                                 | Процесс извлечения и интеграции данных для заполнения архива DSS. Выполняется периодически                                                                                                                  | Первое поколение баз данных DSS<br>Обычно СУРБД                                                                                                      | Средства запросов с некоторыми возможными аналитическими запросами и предопределенными запросами       | Расширенные средства представления данных с графическими возможностями                                              |
| Первое поколение хранения данных предпочтения                                             | Улучшенные средства извлечения и интеграции данных<br>Сюда относятся: доступ к различным источникам данных, преобразование, фильтры, агрегатные функции, классификация, планирование, разрешение конфликтов | Хранилище данных интегрирует БД DSS для поддержки всего предприятия<br>Использует технологию СУРБД, оптимизированную для запросов<br>Модель "звезда" | Все, что указано выше, плюс поддержка улучшенных функций запроса и анализа                             | Все, что указано выше, плюс дополнительные инструменты многомерного представления с возможностью детального анализа |
| Второе поколение хранения данных с использованием многомерных СУБД (МСУБД)                | Все, что указано выше                                                                                                                                                                                       | В хранилище данные помещаются с помощью технологии МСУБД, которая основана на многомерных структурах данных, которые называются "кубами"             | Все, что указано выше, но используется другой интерфейс запросов для доступа к МСУБД (запатентованный) | Все, что указано выше, но используются "кубы" и многомерные матрицы; ограничено измерениями куба                    |

### 13.3.2. Двенадцать правил для хранилища данных

В 1994 году Уильям Инмон (William H. Inmon) и Чак Келли (Chuck Kelley) разработали список из 12 правил, определяющих хранилище данных, в которых нашли отражение многие обсуждавшиеся нами ранее вопросы.

1. Хранилище данных и операционная среда должны быть разделены.
2. Данные в хранилище должны интегрироваться.
3. В хранилище содержатся данные за длительный период времени.
4. Данные в хранилище представляют собой мгновенный снимок данных, полученный в данный момент времени.
5. Данные в хранилище предметно ориентированы.
6. Данные в хранилище в основном предназначены только для чтения с периодическим обновлением их на основе операционных данных. Не допускается оперативное обновление информации в хранилище.
7. Жизненный цикл разработки хранилища данных отличается от классической информационной системы. Во главу угла в хранилище поставлены данные, а в классической БД — процессы.
8. В хранилище данных содержится информация с несколькими уровнями детализации: текущая детализированная информация, старая детализированная информация, слабо обобщенная информация, информация высокой степени обобщения.
9. Среда хранилища данных характеризуется транзакциями, выполняющими только чтение больших наборов данных. Среда операционных баз данных характеризуется большим объемом транзакций обновления нескольких сущностей за один раз.
10. Хранилище данных имеет в своем составе систему, которая отслеживает источники данных, преобразование и хранение.
11. Метаданные хранилища данных являются важнейшим компонентом этой инфраструктуры. Метаданные описывают и определяют все элементы данных. Метаданные описывают источник, преобразование, интеграцию, хранение, использование, связи и историю каждого элемента данных.
12. В хранилище данных должен иметься механизм использования ресурсов, который обеспечивает оптимальное использование данных конечным пользователем.

Обратите внимание, как эти 12 правил охватывают весь процесс становления хранилища данных от его появления как сущности, отличной от операционного архива, до процесса определения его функций, компонентов и способов управления. Последнее поколение OLAP-систем предоставляет законченную инфраструктуру для проектирования, разработки, реализации и использования систем поддержки решений в рамках предприятия. Далее мы опишем эти сравнительно недавно появившиеся системы более подробно.

## 13.4. Оперативный анализ данных (OLAP)

Необходимость интенсивной поддержки принятия решений стала причиной появления инструментов нового поколения. Эти инструментальные средства, получившие название системы оперативного анализа данных (OLAP), создают новую среду анализа данных для обеспечения поддержки принятия решений, моделирования бизнеса и исследования операций. Все системы OLAP имеют четыре основных характеристики.

- В OLAP используется технология многомерного анализа данных.
- Все системы OLAP обеспечивают расширенную поддержку баз данных.
- Системы OLAP предоставляют конечному пользователю простой и удобный графический интерфейс.
- В системах OLAP используются клиент/серверные технологии.

Далее мы подробно исследуем каждое из этих свойств.

### Технология многомерного анализа данных

Это одно из наиболее характерных свойств систем OLAP. Многомерный анализ данных это такой способ обработки информации, при котором данные рассматриваются как часть многомерной структуры. (Мы упоминали многомерность данных как одно из требований к системам поддержки решений, см. *разд. 13.2.1.*) Интерес к многомерному анализу данных возникает оттого, что руководство предприятия обычно представляет данные с точки зрения перспектив бизнеса. То есть оно, как правило, рассматривает данные во взаимосвязи с другими данными предприятия.

Чтобы лучше понять эту точку зрения, рассмотрим, как аналитик предприятия исследует данные об объемах продаж. В этом случае он интересуется объемом продаж в его взаимосвязи с другими параметрами бизнеса, например, с клиентами и временем. Иначе говоря, клиенты и время рассматриваются как различные характеристики объема продаж. На рис. 13.4 показано, чем операционное (одномерное) представление отличается от многомерного представления.

Обратите внимание на рис. 13.4, что табличное представление объемов продаж не очень годится для поддержки принятия решений, поскольку связь INVOICE → LINE между сущностями INVOICE (счета) и LINE (строка счета) не может служить основой для выяснения перспектив бизнеса по объемам продаж. С другой стороны, представление конечного пользователя о данных по продажам с точки зрения перспектив бизнеса лучше всего описывать многомерным представлением, чем с помощью отдельных таблиц. Обратите также внимание, что многомерное представление позволяет конечным пользователям объединять или агрегировать данные на различных уровнях: общие данные объема продаж по клиентам и по датам. Наконец, многомерное представление данных позволяет аналитикам легко переключать качественные характеристики бизнеса (измерения) от данных продаж по клиентам к данным продаж по отделу, региону и т. д.

## Оперативное представление данных по объемам продаж

База данных: DW\_TEXT.MDB

Таблица: DW\_INVOICE

|   | INV_NUM | INV_DATE  | CUS_NAME    | INV_TOTAL  |
|---|---------|-----------|-------------|------------|
| + | 2034    | 15-May-02 | Dartonik    | \$1,400.00 |
| + | 2035    | 15-May-02 | Summer Lake | \$1,200.00 |
| + | 2036    | 16-May-02 | Dartonik    | \$1,350.00 |
| + | 2037    | 16-May-02 | Summer Lake | \$3,100.00 |
| + | 2038    | 16-May-02 | Trydon      | \$400.00   |

Таблица: DW\_LINE

|  | INV_NUM | LINE_NUM | PROD DESCRIPTION             | LINE PRICE | LINE QUANTITY | LINE AMOUNT |
|--|---------|----------|------------------------------|------------|---------------|-------------|
|  | 2034    | 1        | Serial Mouse                 | \$45.00    | 20            | \$900.00    |
|  | 2034    | 2        | 3.5" Floppy Drive            | \$50.00    | 10            | \$500.00    |
|  | 2035    | 1        | Everlast Hard Drive, 16.8 GB | \$200.00   | 6             | \$1,200.00  |
|  | 2036    | 1        | Serial Mouse                 | \$45.00    | 30            | \$1,350.00  |
|  | 2037    | 1        | Serial Mouse                 | \$45.00    | 10            | \$450.00    |
|  | 2037    | 2        | Roadster 56KB Ext. Modem     | \$120.00   | 5             | \$600.00    |
|  | 2037    | 3        | Everlast Hard Drive, 16.8 GB | \$205.00   | 10            | \$2,050.00  |
|  | 2038    | 1        | NoTech Speaker Set           | \$50.00    | 8             | \$400.00    |

## Многомерное представление данных по объемам продаж

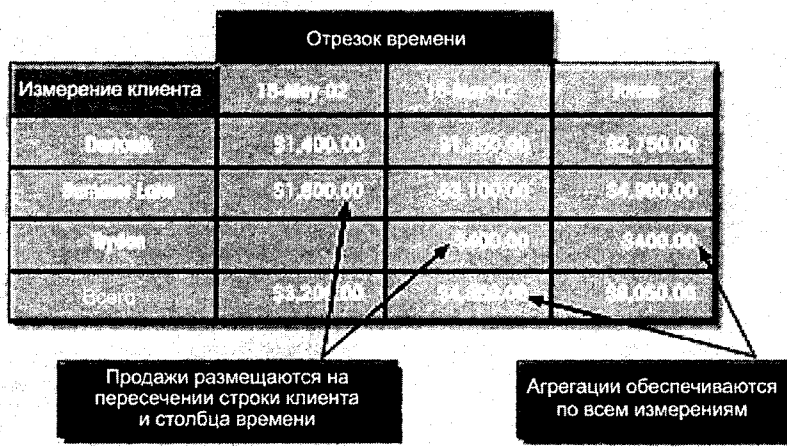


Рис. 13.4. Операционное и многомерное представления о продажах

Технология многомерного анализа данных обладает и дополнительными функциональными возможностями.

- ❑ Расширенные возможности представления данных: трехмерная графика, перекрестные таблицы, трехмерные кубы и т. д. Такие возможности представления дан-

ных хорошо согласуются с системами электронных таблиц, статистическими пакетами, а также пакетами по созданию запросов и отчетов.

- Развитые функции агрегации, объединения и классификации данных, которые позволяют аналитикам создавать несколько уровней, слоев и кубов агрегации данных, детальный и укрупненный анализ данных по различным характеристикам (измерениям) и уровням агрегации. Например, совокупность данных за неделю, квартал или год позволяет аналитикам бизнеса производить детализированный или укрупненный анализ информации.
- Развитые вычислительные функции: использование бизнес-переменных (курс акций, период сопоставлений, прибыль от продаж, товарная прибыль, относительные изменения и т. д.), экономические и бухгалтерские показатели (рентабельность, накладные расходы, распределение прямых затрат, поступления и т. д.), статистические функции и функции прогноза и т. п. Эти функции выполняются автоматически, и конечному пользователю не нужно переопределять их компоненты каждый раз при получении доступа к ним.
- Функции моделирования данных: поддержка сценариев оценки возможных вариантов (сценарии "what-if" ("что, если")), оценка переменных, влияние переменных на результат, линейное программирование и другие средства моделирования.

|    | A     | B | C      | D      | E      |
|----|-------|---|--------|--------|--------|
| 1  |       |   |        |        |        |
| 2  |       |   | Qtr1   | Qtr2   |        |
| 3  |       |   | Actual | Budget | Actual |
| 4  | East  |   | 2747   | 2880   | 3352   |
| 5  |       |   | 562    | 960    | 610    |
| 6  |       |   | 591    | 770    | 922    |
| 7  |       |   | 1480   | 1890   | 1815   |
| 8  |       |   | 555    | 620    | 652    |
| 9  |       |   | 5380   | 6500   | 6499   |
| 10 | West  |   | 1042   | 2350   | 849    |
| 11 |       |   | 2325   | 2570   | 2423   |
| 12 |       |   | 2363   | 2620   | 2739   |
| 13 |       |   | 1407   | 1420   | 1504   |
| 14 |       |   | 2025   | 2620   | 1975   |
| 15 |       |   | 7137   | 8960   | 7515   |
| 16 | South |   | 1051   | 1730   | 1198   |
| 17 |       |   | 1465   | 1640   | 1540   |
| 18 |       |   | 561    | 810    | 529    |

Рис. 13.5. Интеграция OLAP с электронными таблицами

Поскольку у многих функций анализа и представлений имеются аналоги в программах электронных таблиц, большинство поставщиков OLAP интегрируют свои систе-

мы с настольными системами электронных таблиц (например, Microsoft Excel и Lotus 1-2-3). С помощью графического интерфейса пользователя, например, Windows, меню OLAP просто встраивается в меню Lotus или Excel (рис. 13.5). Такая интеграция очень выгодна для OLAP-систем и поставщиков электронных таблиц, поскольку конечным пользователям удобно получать доступ к средствам анализа данных с помощью известных программ и знакомого интерфейса. При этом дополнительные затраты на обучение и разработку оказываются минимальными.

## Расширенная поддержка баз данных

Для обеспечения эффективной поддержки решений инструментарий OLAP должен иметь в своем составе расширенные средства доступа к данным. Сюда относятся следующие функциональные возможности.

- ☐ Доступ к различным типам СУБД, плоским файлам, а также к внутренним и внешним источникам данных.
- ☐ Доступ к агрегированной информации хранилищ данных, а также к операционным базам данных.
- ☐ Расширенные средства навигации с возможностью детального и укрупненного анализа информации.
- ☐ Быстрое и согласованное реагирование на запросы.
- ☐ Возможность адаптировать запросы пользователей, выраженные в терминах бизнеса или абстрактной модели, к соответствующему источнику данных, а затем к соответствующему языку запросов (обычно SQL). Код запроса должен оптимизироваться для определенного источника данных независимо от того, является источник хранилищем данных или операционной базой данных.
- ☐ Поддержка очень больших баз данных, VLDB. Как уже отмечалось, размер хранилища данных может очень быстро возрастать до нескольких гигабайт и даже терабайт.

Чтобы обеспечить удобный интерфейс, средства OLAP отображают на свои собственные словари данных элементы данных из хранилищ и операционных БД. Эти метаданные затем используются для трансляции аналитических запросов пользователей в соответствующие (оптимизированные) коды запросов, которые затем направляются на определенный источник (источники) данных.

## Простой интерфейс конечного пользователя

Расширенные средства OLAP принесут еще большую пользу, если доступ к ним будет достаточно простым. Поставщики инструментальных средств OLAP давно усвоили это и оснастили свои сложные средства извлечения и анализа данных простым графическим интерфейсом. Большая часть средств интерфейса позаимствована из уже знакомого пользователю предыдущего поколения инструментов анализа данных. Это упрощает изучение и применение инструментария OLAP.

## Клиент/серверная архитектура

Клиент/серверная архитектура предоставляет рабочую среду, в которой можно проектировать, разрабатывать и внедрять новые системы. (Если понадобится, вспомни-

те материал гл. 12, где представлены подробные сведения о свойствах клиент/серверной архитектуры.)

Клиент/серверная инфраструктура позволяет разделить OLAP-систему на несколько компонентов, которые и определяют ее архитектуру. Эти компоненты затем можно разместить на одном компьютере или распределить по нескольким машинам. При этом система OLAP будет обладать большой гибкостью и простотой использования.

### 13.4.1. Архитектура OLAP

Эксплуатационные характеристики OLAP можно разделить на три основные части.

- ☐ Графический интерфейс пользователя OLAP.
- ☐ Логика аналитической обработки OLAP.
- ☐ Логика обработки данных OLAP.

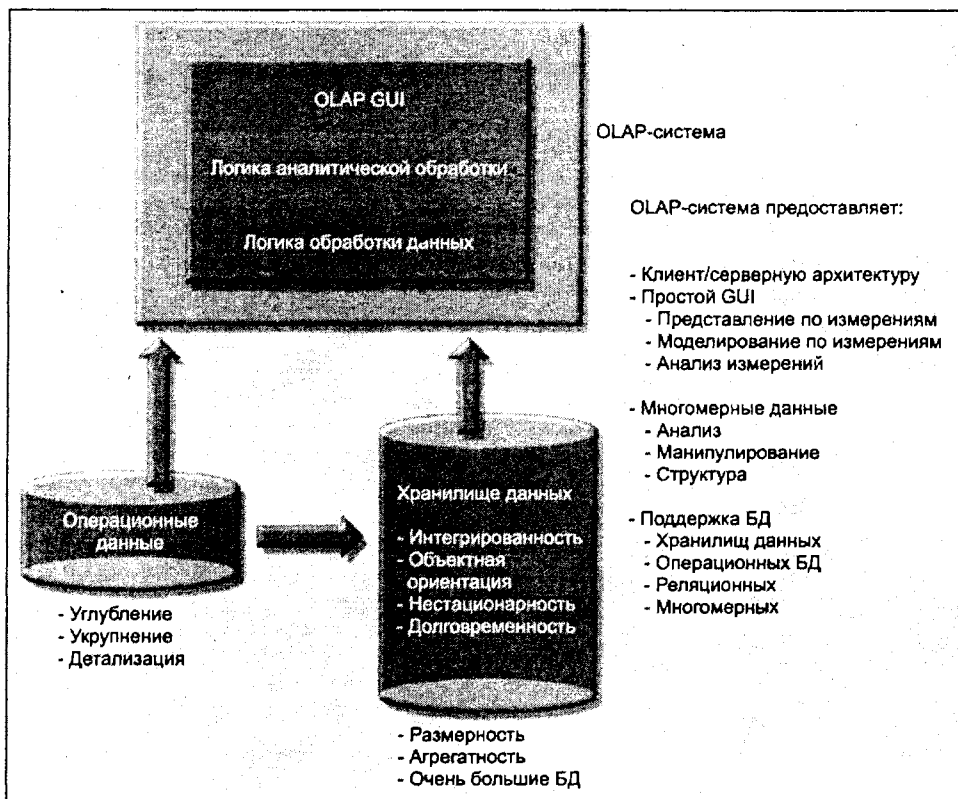


Рис. 13.6. Клиент/серверная архитектура OLAP

Эти три составные части OLAP, размещенные в клиент/серверной инфраструктуре, позволяют использовать свойства OLAP: многомерный анализ данных, расширен-



ную поддержку баз данных и простой интерфейс пользователя. Как показано на рис. 13.6, OLAP-системы спроектированы таким образом, чтобы использовать как операционные данные, так и информацию из хранилищ. Хотя на рис. 13.6 показано, что компоненты OLAP-системы расположены на отдельном компьютере, такой сценарий является лишь одним из многих. На самом деле, одна из имеющихся здесь проблем состоит в том, что каждый аналитик должен работать на достаточно мощном компьютере, позволяющем хранить OLAP-систему и выполнять всю обработку информации локально. Кроме того, каждый аналитик использует отдельную копию данных. Поэтому копии данных должны синхронизироваться так, чтобы гарантировать работу аналитиков с одними и теми же данными. Другими словами, каждый конечный пользователь должен иметь собственную "частную" копию данных и программ, что возвращает нас к проблеме *островков информации* (islands of information), обсужденной в гл. 12. Такой подход не дает каких-либо преимуществ перед единым представлением предприятия, совместно используемым несколькими пользователями.

Более общей и более практичной можно считать архитектуру, при которой OLAP GUI выполняется на клиентской рабочей станции, в то время как OLAP-машина или сервер, содержащий логику аналитической обработки OLAP и логику обработки данных OLAP, выполняется на совместно используемом компьютере. В этом случае OLAP-сервер является внешним интерфейсом для хранилища данных, обеспечивающего систему поддержки решений. Этот внешний интерфейс или промежуточный уровень (поскольку он находится между хранилищем и GUI конечного пользователя) получает и обрабатывает запросы, инициируемые различными инструментами конечных пользователей. GUI конечного пользователя должен представлять собой специально разработанную для клиента программу, точнее, подключаемый модуль (plug-in), интегрированный с Lotus 1-2-3, Microsoft Excel или какой-либо другой программой анализа данных и создания запросов. На рис. 13.7 представлена такая схема.

На рис. 13.7 следует обратить внимание, что хранилище данных создается и обслуживается процессом программного обеспечения, независимого от OLAP-системы. Независимое программное обеспечение выполняет извлечение данных, фильтрацию и интеграцию, необходимые для преобразования операционных данных в хранилище. Такой сценарий отражает тот факт, что в большинстве случаев организация хранилищ данных и анализ данных выполняются раздельно.

Здесь может возникнуть вопрос — а зачем нам вообще нужно хранилище данных, если OLAP-система обеспечивает необходимый многомерный анализ операционных данных? Ответ заключен в самом определении OLAP. Мы определили OLAP как "расширенную среду анализа информации, обеспечивающую поддержку систем принятия решений, бизнес-моделирования и научных исследований". Ключевое слово в этом определении — "среда", куда включена клиент/серверная технология. Понятие *среда* определяется как "окружение или атмосфера". А атмосфера окружает некое ядро. В данном случае ядро состоит из всех деловых операций внутри предприятия, представленных операционными данными. Точно так же, как в атмосфере есть несколько слоев (уровней), есть и несколько уровней обработки данных, причем каждый следующий внешний уровень отражает более укрупненный взгляд на информацию. OLAP-система может получать доступ или к обоим типам хранения информации (операционные БД или хранилища данных), или только к одному из них. Это зависит от реализации выбранной вами системы. В любом случае, многомерный

анализ данных предполагает наличие некоторого типа многомерного представления данных, обычно предоставляемого OLAP-машиной.

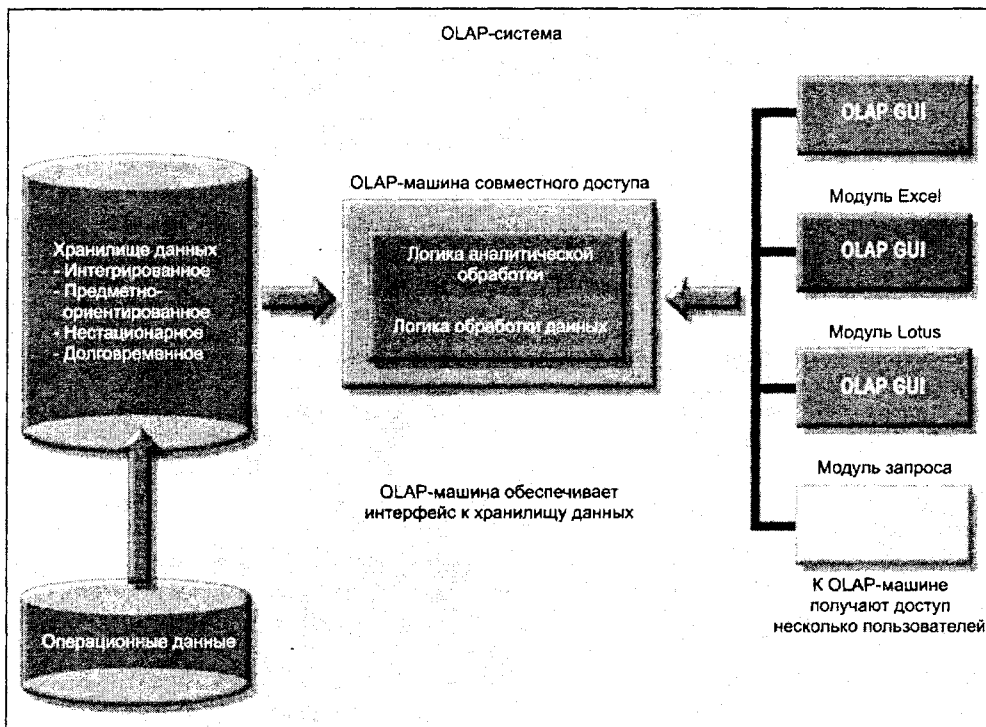


Рис. 13.7. Схема с OLAP-сервером

В большинстве реализаций хранилище данных и OLAP являются взаимосвязанными и дополняющими друг друга инфраструктурами. В то время как хранилище данных это интегрированные, предметно-ориентированные, нестационарные и долговременные данные для системы поддержки решений, OLAP-система предоставляет внешний интерфейс, с помощью которого конечные пользователи получают доступ к информации и могут ее анализировать. Кроме того, OLAP-система позволяет получать прямой доступ к операционным данным, преобразуя их и храня в многомерной структуре. Иначе говоря, OLAP-система предоставляет компонент многомерного хранения данных, как это показано на рис. 13.8.

На рис. 13.8 представлен сценарий, в котором OLAP-машина извлекает данные из операционной БД, а затем хранит их в многомерной структуре для дальнейшего анализа. Процесс извлечения следует тем же соглашениям, которые используются в хранилище данных. Поэтому OLAP-система представляет собой мини-компонент хранилища, очень похожий на витрину данных, о которой мы говорили в предыдущих разделах этой главы. В этом случае OLAP-машина берет на себя все функции по извлечению, фильтрации, классификации и агрегации данных, что обычно является

функцией хранилища данных. Фактически при правильной реализации вместо того, чтобы передавать эти функции OLAP, хранилище данных должно выполнять все рутинные операции подготовки данных, тем самым избегая дублирования функций. Более того, хранилище данных обрабатывает данные более эффективно, чем OLAP, так что можно оценить все выгоды наличия централизованного хранилища, которое служит большой базой данных для системы поддержки решений предприятия.

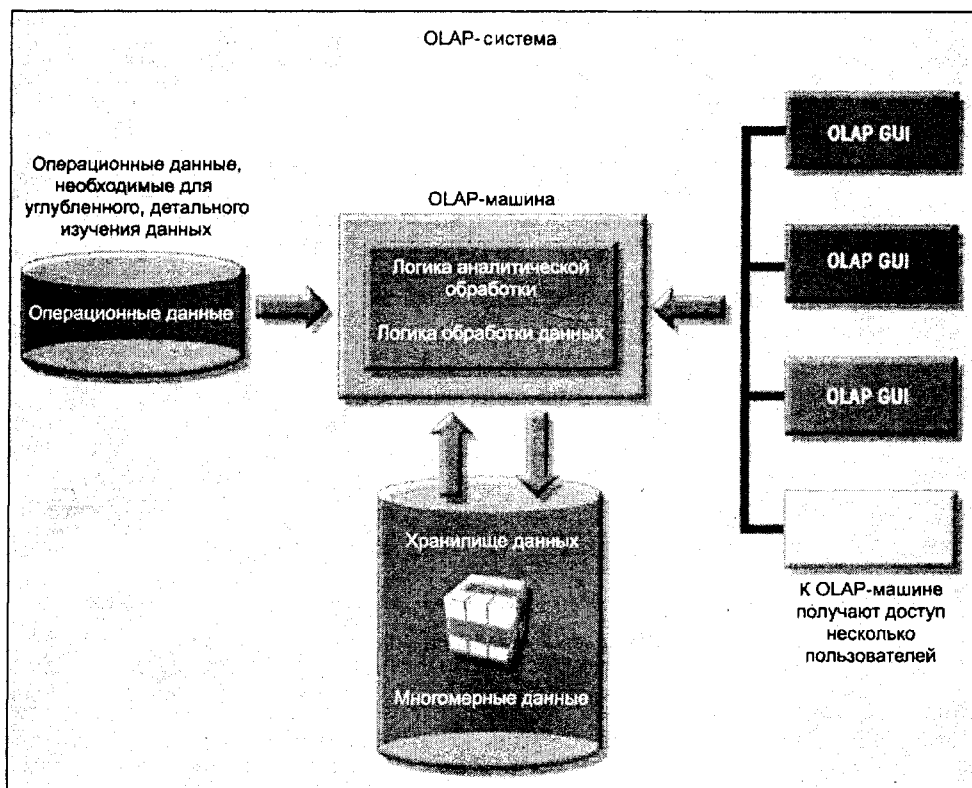


Рис. 13.8. OLAP-сервер со схемой многомерного хранения данных

Для лучшей производительности в некоторых OLAP-системах применяется смешанная идеология хранилища данных и витрины данных, когда на рабочей станции конечного пользователя хранится небольшое извлечение данных из хранилища. Цель этого — ускорение доступа к информации и визуализация данных (графическое представление данных и их свойств). Логика такой технологии заключается в предположении, что большинство конечных пользователей, как правило, работают с относительно небольшим устойчивым подмножеством данных хранилища. Например, аналитики сбыта, как правило, имеют дело с данными по продажам, в то время как анализ клиентов связан с данными по клиентам и т. д. Такой сценарий представлен на рис. 13.9.

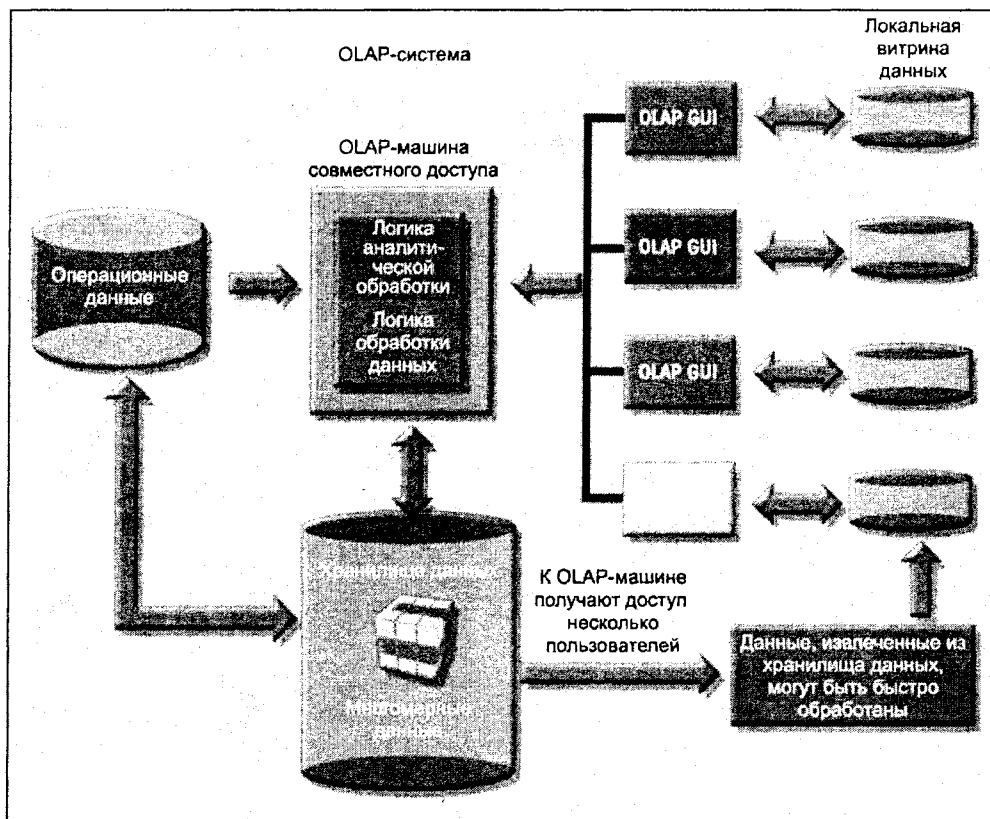


Рис. 13.9. OLAP-сервер с локальной мини-витриной данных

Теперь необходимо резюмировать рассмотренные нами основные свойства архитектуры OLAP-систем. Независимо от схемы размещения компонентов, в OLAP-системах обязательно используются многомерные данные. Но как эти многомерные данные хранить наилучшим образом и как ими управлять? Сторонники систем OLAP явно разделились: часть из них предпочитает использовать для хранения многомерных данных реляционные БД, в то время как другие выдвигают аргументы в пользу преимущественного использования для этих целей специализированных многомерных баз данных. Далее мы рассмотрим основные элементы каждого из этих подходов.

### 13.4.2. Реляционные OLAP-системы

*Реляционные системы оперативного анализа данных (relational online analytical processing, ROLAP)* обеспечивают функциональные возможности OLAP при помощи средств реляционных БД и общеизвестных средств создания реляционных запросов при хранении и анализе многомерной информации.

Такой подход строится на имеющихся реляционных технологиях и представляет собой их естественное расширение для компаний, использующих реляционные СУБД. ROLAP дополняет традиционные технологии СУРБД следующими возможностями:

- поддержка многомерных схем данных в рамках СУРБД;
- оптимизация языка доступа к данным и запросов для многомерных данных;
- поддержка очень больших баз данных (very large database, VLDB).

## Поддержка схем многомерных данных в СУРБД

Реляционная технология использует для хранения данных нормализованные таблицы. Применение нормализации в качестве базовой методологии проектирования реляционных БД выглядит как препятствие к их использованию в OLAP-системах. Для получения нормализованных таблиц объекты бизнеса разбиваются на более мелкие составляющие. Например, компоненты данных о сбыте могут храниться в четырех или пяти различных таблицах. Причина использования нормализованных таблиц — уменьшение избыточности данных, что позволяет избежать аномалий и упростить обновление данных. К сожалению, в системах поддержки решений данные проще исследовать, когда мы их видим в соотношении с другими данными (см. пример на рис. 13.4). При таком представлении среды данных необходимо иметь в виду, что данные для системы поддержки решений, как правило, дублированы, ненормализованы и предварительно рассортированы. Эти свойства, на первый взгляд, препятствуют использованию стандартных технологий проектирования реляционных СУРБД в качестве основы для работы с многомерными данными.

К счастью для тех, кто делал большие инвестиции в реляционные технологии, в ROLAP используется специальная технология поддержки многомерного представления данных. Эта специальная техника проектирования называется *схемой "звезда"* (*star schema*), которую мы подробно будем рассматривать далее в этой главе. В сущности, схема "звезда" создает некий эквивалент многомерной схемы БД на основе существующей реляционной базы данных.

Новая схема "звезда" разработана скорее для оптимизации операций запроса данных, чем для операций обновления. Естественно, изменение основ проектирования означает, что инструментарий, используемый для доступа к подобным данным, тоже необходимо изменить. Конечные пользователи, знакомые с традиционными средствами реляционных запросов, обнаружат, что эти средства не будут эффективно работать в новой схеме "звезда". Однако в ROLAP-системе проблема решается поддержкой использования в схеме "звезда" знакомых средств создания запросов. ROLAP-система предоставляет дополнительные функции анализа данных и улучшает методы оптимизации запросов и визуализации.

## Оптимизация языка доступа к данным и эффективности запросов для многомерных данных

Другой недостаток реляционных баз данных состоит в том, что язык SQL, использующийся в СУРБД, не подходит для выполнения расширенного анализа данных. В большинстве систем поддержки решений необходимо использовать многопроходные SQL-запросы или вложенные SQL-операторы. Чтобы избавиться от этого не-

достатка, ROLAP-система расширяет SQL таким образом, чтобы его можно было по-разному использовать для данных хранилища (на основе схемы "звезда") и для операционных данных (нормализованные таблицы). В этом случае ROLAP-система будет должным образом генерировать SQL-код, необходимый для доступа к данным, хранящимся по схеме "звезда".

Эффективность запросов тоже повышается, поскольку оптимизатор запросов модифицирован таким образом, что он может определять цели SQL-кода. Например, если целью запроса является хранилище данных, оптимизатор передаст запрос в хранилище. Однако если конечный пользователь выполняет углубленные запросы к операционным данным, оптимизатор идентифицирует эту операцию и должным образом оптимизирует SQL-запросы, перед тем как передать их операционной СУБД.

Другой источник улучшения выполнения запросов — использование улучшенной техники индексирования, например, индексация по битовой карте (bitmap indexes) в реляционных БД. Судя по названию, индексация по битовой карте основана на битовом представлении (0 и 1) какого-то условия. Например, если атрибут REGION на рис. 13.2 имеет только четыре значения — North (север), South (юг), East (восток) и West (запад), то различные варианты могут быть представлены так, как это показано в табл. 13.8. (В табл. 13.8 представлены только первые 10 строк из рис. 13.2. "1" представляет собой "бит включения", а "0" — "бит выключения". Например, для представления строки с атрибутом REGION = "East" будет "включен" только бит "East". Обратите внимание, что каждая строка должна быть представлена в индексной таблице.)

**Таблица 13.8.** Битовое представление значений атрибута REGION

| North | South | East | West |
|-------|-------|------|------|
| 0     | 0     | 1    | 0    |
| 0     | 0     | 1    | 0    |
| 1     | 0     | 0    | 0    |
| 1     | 0     | 0    | 0    |
| 1     | 0     | 0    | 0    |
| 0     | 1     | 0    | 0    |
| 0     | 1     | 0    | 0    |
| 0     | 1     | 0    | 0    |
| 0     | 0     | 0    | 1    |
| 0     | 0     | 0    | 1    |

Изучая табл. 13.8, обратите внимание, что битовый индекс занимает минимум места. Поэтому индексация по битовой карте более эффективна и может обрабатывать большие массивы данных, чем индексы, используемые в большинстве реляционных БД. Однако необходимо помнить, что индексация по битовой карте, главным образом, используется в том случае, когда число возможных значений атрибута (т. е. домен атрибута) сравнительно невелико. Например, атрибут REGION в нашем примере имеет

только четыре значения. Семейное положение (женат, холост, вдовец, разведен) — еще один кандидат на использование битовой индексации, как и пол (М и Ж).

Инструментарий ROLAP в основном является трехзвенным клиент/серверным продуктом, в котором интерфейс конечного пользователя, анализ и обработка данных выполняются на различных компьютерах. На рис. 13.10 представлено взаимодействие компонентов трехзвенной ROLAP-системы.

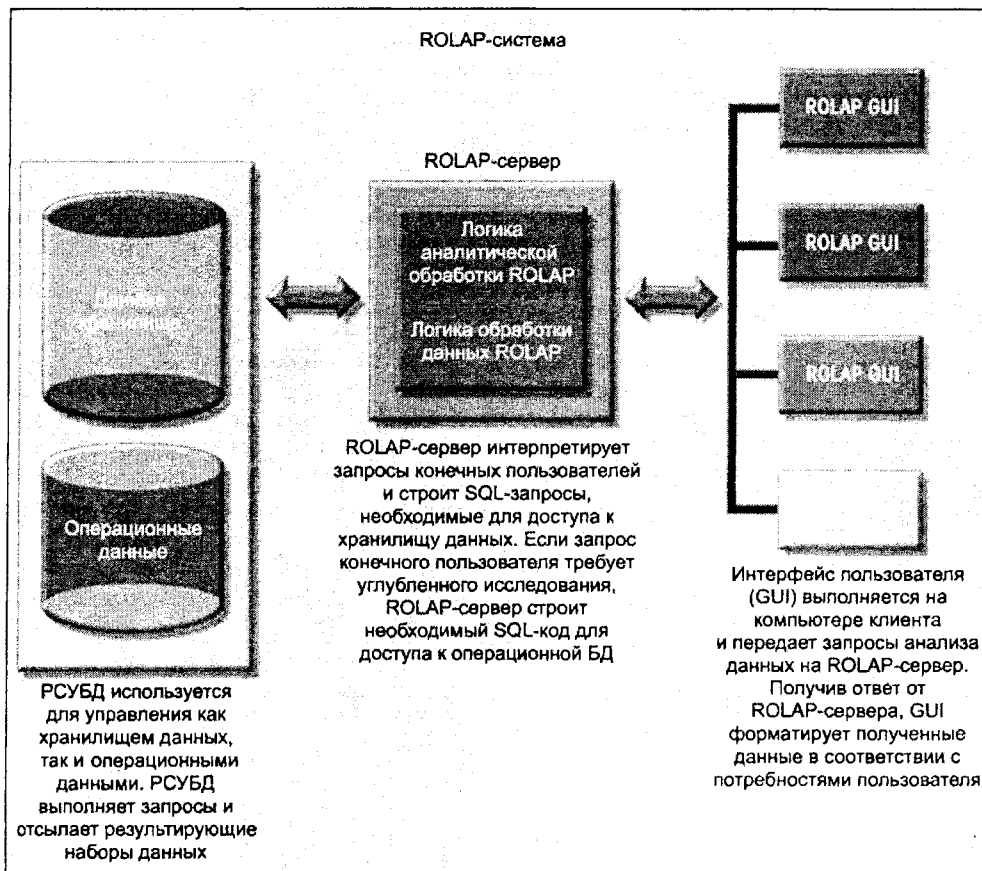


Рис. 13.10. Типичная архитектура клиент/серверной ROLAP-системы

## Поддержка очень больших баз данных (VLDB)

Ранее отмечалось, что поддержка VLDB (very large database, очень большая база данных) является обязательным требованием к системам DSS. Поэтому если в роли DSS выступает реляционная БД, то она должна обеспечивать возможность хранения больших массивов данных. Емкость БД и процесс загрузки данных в нее являются очень важными характеристиками. Данные, относящиеся к системе поддержки ре-

шений, обычно загружаются в пакетном режиме из операционных БД. Поэтому СУРБД должны иметь соответствующие средства импорта, интеграции и заполнения хранилища данными из операционной БД. Большинство реляционных средств загрузки данных выполняют загрузку в пакетном режиме. Однако пакетные операции блокируют источник и приемник данных на время операции. Скорость операции по загрузке данных имеет существенное значение, особенно когда операционные системы функционируют круглосуточно в течение года. Поэтому на выполнение пакетных процедур отводится очень небольшое время, как правило, это делается в период простоя системы.

Принимая во внимание открытость клиент/серверной архитектуры, система ROLAP предоставляет расширенные возможности для систем поддержки решений, которые могут быть распространены на все предприятие. Очевидно, что ROLAP-система является естественным выбором для тех компаний, которые уже используют реляционные БД для операционных данных. Учитывая широкий рынок реляционных БД, вряд ли стоит удивляться тому, что разработчики современных реляционных СУРБД модернизируют свои продукты для поддержки хранилищ данных.

### 13.4.3. Многомерные OLAP-системы

*Многомерная система оперативного анализа данных (Multidimensional Online Analytical Processing, MOLAP)* расширяет возможности OLAP до поддержки *многомерных систем управления базами данных (МСУБД, Multidimensional Database Management System, MDBMS)*. (В МСУБД используются специальные технологии для хранения данных в матричных  $n$ -мерных массивах). В системах MOLAP предполагается, что многомерные БД лучше всего подходят для управления, хранения и анализа многомерных данных. Большинство технологических приемов, используемых в МСУБД, берет свое начало в таких областях инженерии, как САПР (системы автоматизированного проектирования), САУ (системы автоматизированного управления) и геоинформационные системы (ГИС, geographic information system, GIS).

Концептуально для конечных пользователей МСУБД представляют хранящиеся данные в виде трехмерного куба, называемого *кубом данных (data cube)*. Положение каждого значения данных в кубе данных определяется координатами по осям  $X$ ,  $Y$  и  $Z$  в трехмерном пространстве. Координаты  $X$ ,  $Y$  и  $Z$  представляют собой качественные характеристики (измерения) данных. С помощью кубов данных можно отображать и  $n$ -мерные представления, кубы при этом становятся *гиперкубами*. Кубы данных создаются при помощи извлечения данных из операционных БД или из хранилищ данных. Одна из важнейших характеристик кубов данных — их статичность; т. е. они не подлежат изменению и должны создаваться до того, как их можно будет использовать. Иначе говоря, кубы данных не могут создаваться при помощи нерегламентированных запросов. Вместо этого запрос обращается к уже созданному кубу данных с определенными осями (например, куб данных объемов продаж имеет следующие характеристики: товар, местоположение и время). Поэтому процесс создания куба данных очень важен и требует тщательного проектирования интерфейса и глубокого анализа информации. Такая тщательная работа над интерфейсом определяет хорошую скорость работы баз данных MOLAP по сравнению с базами данных ROLAP, особенно в тех случаях, когда мы имеем дело со сравнительно небольшими наборами данных. Для того чтобы ускорить доступ к данным,



кубы данных обычно хранятся в участке оперативной памяти — *кэше куба* (*cube cash*). (Куб данных представляет собой только окно в predetermined набор данных БД. То есть куб данных и база данных это не одно и то же.) Поскольку системы MOLAP обладают всеми преимуществами клиент/серверных систем, кэш куба может размещаться на сервере MOLAP, на клиенте MOLAP или и на сервере, и на клиенте. На рис. 13.11 представлены основные компоненты архитектуры MOLAP.

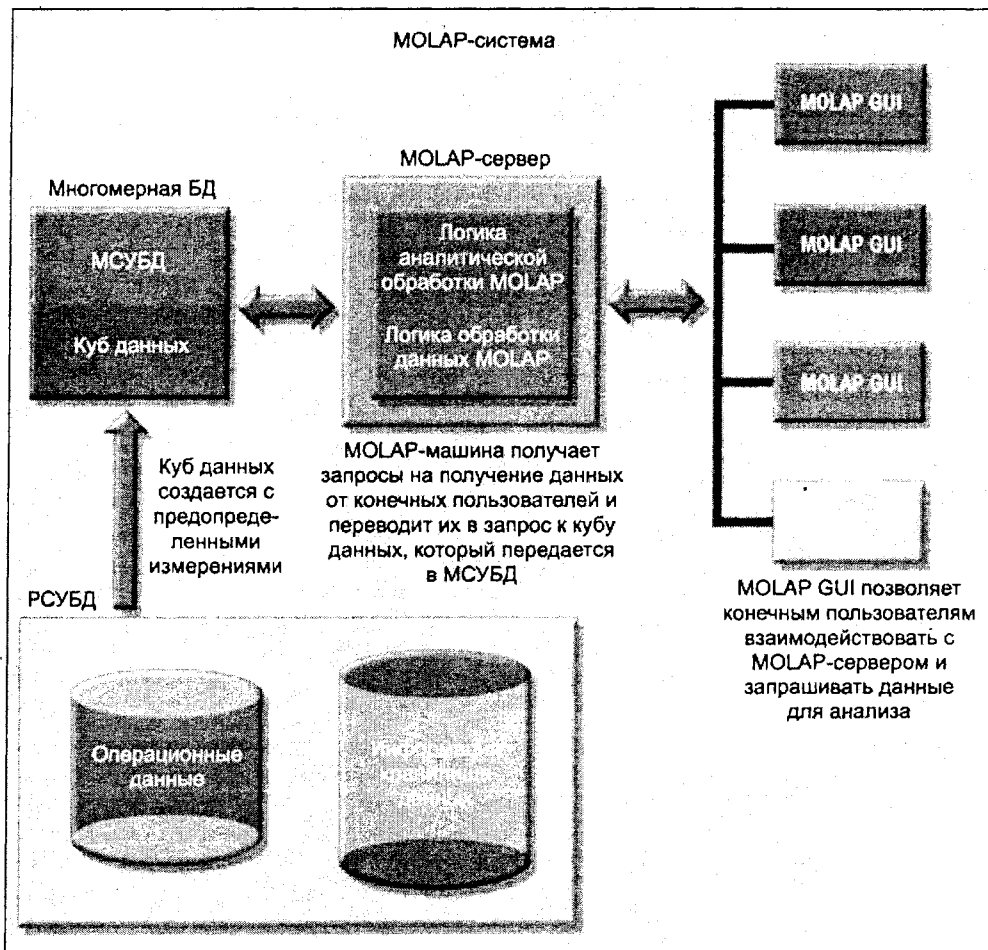


Рис. 13.11. Клиент/серверная архитектура MOLAP

Существует новый тип СУБД, называемый *системой управления БД, размещенной в оперативной памяти* (*Main Memory Database, MMDB*). Поскольку MMDB обслуживает все запросы непосредственно из памяти, она работает примерно в тысячу раз быстрее, чем традиционные реляционные БД, завязанные на обмен данными с диском. Даже корпорация Microsoft включила в Windows 2000 малоизвестную СУБД, дейст-

вующую непосредственно в ОЗУ, — IMDB (более подробно с такими типами СУБД можно ознакомиться на сайте [www.zdnet.com](http://www.zdnet.com). О технологии In-Memory Database можно узнать на сайте [www.timesten.com](http://www.timesten.com)).

Возможность размещать куб данных в памяти обеспечивает быстрое время отклика, но также требует больших информационных ресурсов для МСУБД (объем памяти и тактовая частота процессора) по сравнению с реляционными СУБД. Кроме того, приверженцы ROLAP-систем утверждают, что идея куба данных ограничивает гибкость, масштабируемость и простоту интеграции систем.

Поскольку в кубе данных заранее определен некоторый набор качественных характеристик, добавление нового измерения приведет к перестройке всего куба данных. Эта перестройка требует определенного времени. Поэтому, если куб данных создается достаточно часто, то МСУБД теряет свои преимущества в скорости по сравнению с реляционными СУБД. И хотя МСУБД имеет определенные достоинства по сравнению с реляционными БД, она все же лучше всего годится для небольших и средних наборов данных. Масштабируемость МСУБД действительно несколько ограничена, поскольку куб данных большого размера занимает много места в оперативной памяти, что ухудшает производительность операционной системы и прикладных программ. Кроме того, в МСУБД используется специальная технология хранения данных, которая в свою очередь предполагает применение соответствующих методов доступа к данным с использованием многомерных языков запроса.

На анализ многомерных данных оказывает влияние и то, как СУБД обрабатывает *разряженность* (*sparsity*). Разряженность — это степень плотности данных, хранимых в кубе данных. Разряженность рассчитывается делением общего числа параметров в кубе на общее число ячеек куба. Поскольку измерения куба данных предопределены, заполняются не все ячейки куба (некоторые ячейки пусты). Возвращаясь к нашему примеру с объемами продаж, можно сказать, что есть множество товаров, которые не были проданы в течение данного периода времени в данном месте (нулевые продажи). На самом деле, в ряде случаев заполняется менее 50% всех ячеек куба. В любом случае многомерные базы данных должны эффективно справляться с разряженностью для того, чтобы уменьшить накладные расходы на обработку и требования к ресурсам.

Сторонники реляционного подхода также указывают, что использование готовых решений затрудняет интеграцию МСУБД с другими источниками данных и инструментальными средствами предприятия. Все же несмотря на то что требуется достаточно много времени и усилий на интеграцию новой технологии и существующей информационной архитектуры, MOLAP может быть вполне приемлемым решением для магазинов, где нормой являются небольшие и средние базы данных, а скорость доступа к приложению очень важна.

#### **13.4.4. Сравнение реляционных и многомерных систем OLAP**

В табл. 13.9 суммируются некоторые преимущества и недостатки ROLAP и MOLAP. Однако необходимо подчеркнуть, что, возможно, некоторые из приведенных преимуществ по мере развития технологий придется пересмотреть. Например, быстрые процессоры и мощные компьютеры могут снять проблемы скорости и размера в

системах OLAP. Выбор того или иного подхода часто зависит от того, что берется за основу при оценке системы. Например, полная оценка OLAP-системы должна включать в себя стоимость разработки, поддержку аппаратных платформ, совместимость с существующими СУБД, требования к программированию, производительность и доступность административных средств. Тем не менее, данные, приведенные в табл. 13.9, можно использовать в качестве отправной точки при сравнении OLAP-систем.

**Таблица 13.9.** Сравнение реляционных и многомерных систем OLAP

| Свойства              | ROLAP                                                                                              | MOLAP                                                                              |
|-----------------------|----------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| Схема                 | Использует схему "звезда"<br>Дополнительные характеристики (измерения) можно добавлять динамически | Использует кубы данных<br>Дополнительные измерения требуют перестройки куба данных |
| Размер базы данных    | От средних до больших                                                                              | Небольшие и средние                                                                |
| Архитектура           | Клиент/серверная<br>Стандартная<br>Открытая                                                        | Клиент/серверная<br>Специфическая разработка                                       |
| Доступ                | Поддержка нерегламентированных запросов<br>Неограниченность измерений                              | Ограничен predetermined измерениями                                                |
| Требования к ресурсам | Высокие                                                                                            | Очень высокие                                                                      |
| Гибкость              | Высокая                                                                                            | Низкая                                                                             |
| Масштабируемость      | Высокая                                                                                            | Низкая                                                                             |
| Скорость              | Хорошая с небольшими наборами данных; средняя для средних и больших наборов                        | Быстрая для небольших и средних наборов данных, средняя для больших наборов        |

Поставщики ROLAP и MOLAP совместно работают над интеграцией своих решений для унификации среды системы поддержки решений. По мере добавления к продуктам дополнительных функциональных возможностей различия между ними стираются. Весьма вероятно, что будут найдены точки соприкосновения сторонников СУРБД и МСУБД, что приведет к появлению нового типа СУБД, в котором будут использованы лучшие возможности ROLAP и MOLAP. Возможно, эта новая СУБД позволит обрабатывать табличные и многомерные данные с одинаковой легкостью. А пока в реляционных базах данных для обработки многомерной информации успешно используется схема "звезда", а их высокие ставки на рынке программных средств вряд ли в скором времени снизят их популярность.

## 13.5. Схема "звезда"

Как уже ранее указывалось в этой главе, схема "звезда" представляет собой технику моделирования данных, используемую для отображения многомерных данных систем поддержки решений на реляционную схему. Необходимость разработки схемы "звезда" обусловлена тем, что существующие технологии реляционного моделирования, ER-моделирования и нормализации не позволяют использовать структуру БД для расширенного анализа данных.

Схема "звезда" реализует простую модель анализа многомерных данных, в то же время сохраняя реляционную структуру, в которой моделируются операционные данные. Базовая схема "звезда" состоит из четырех компонентов: факты, качественные характеристики (измерения), атрибуты и иерархии атрибутов. Далее мы детально рассмотрим каждый из этих компонентов.

### 13.5.1. Факты

*Фактами* называются числовые параметры (значения), представляющие специфическую сторону деятельности предприятия. Например, данные о продажах являются числовой мерой, характеризующей продажу данного товара или услуги. Факты, которые, как правило, используются в бизнес-данных, представляют собой предметы, расходы, цены и доходы. Факты обычно хранятся в таблице фактов, расположенной в центре схемы "звезда". В *таблице фактов (fact table)* содержатся факты, связанные через свои качественные характеристики — измерения (мы говорили об измерениях в *разд. 13.2.1* и будем исследовать их подробно в следующем разделе).

Факты могут также рассчитываться и получаться во время работы системы. Такие расчетные или полученные факты иногда называют *метриками (metrics)*, для того чтобы отличать их от хранимых фактов. Таблица фактов периодически обновляется (ежедневно, еженедельно, ежемесячно и т. д.) данными из операционных БД.

### 13.5.2. Качественные характеристики (измерения)

*Измерения* представляют собой установленные свойства, которые добавляют фактам дополнительные перспективы. В *разд. 13.2.1* мы установили, что измерения представляют для нас большой интерес, поскольку *данные систем DSS почти всегда рассматриваются по отношению к другим данным*. Например, объемы продаж можно сравнивать по товару от региона к региону или по разным периодам времени. Одна из задач, которую обычно решают в DSS, может звучать так: "Сравнить объемы продаж позиции X по региону Y в первом квартале с 1990 по 1999 год". В этом примере объем продаж имеет следующие измерения: товар, местоположение и время. Такие качественные характеристики обычно хранятся в *таблицах измерений (dimension table)*. На рис. 13.12 представлена схема "звезда" по объемам продаж для товара, местоположения и по времени.

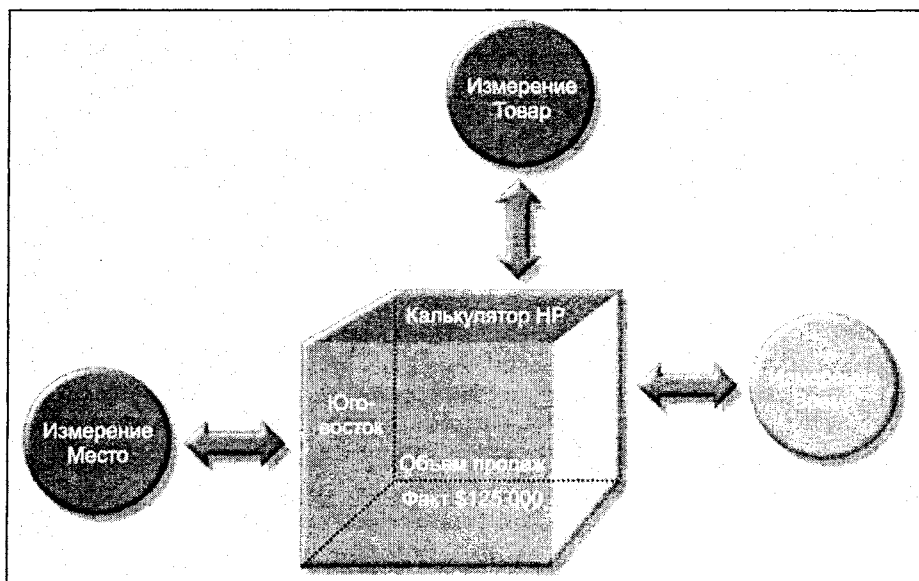


Рис. 13.12. Простейшая схема "звезда"

### 13.5.3. Атрибуты

Каждая таблица измерений содержит атрибуты. Атрибуты часто используются для поиска, фильтрации или классифицирования фактов. *Измерения наглядно представляют свойства о фактах через их атрибуты.* Поэтому проектировщик хранилища данных должен определить общие бизнес-атрибуты, которые будут использоваться аналитиками для точного поиска, группирования или описания измерений. Используя пример с объемами продаж, мы можем определить некоторые возможные атрибуты для каждой характеристики (измерения).

- Измерение "товар": идентификатор товара, описание, тип товара, производитель.
- Измерение "местоположение": регион, штат, город и номер магазина.
- Измерение "время": год, квартал, месяц, неделя и день.

Измерения "товар", "местоположение" и "время", представленные в табл. 13.10, добавляют определенную глубину к фактам объемов продаж. Аналитик теперь может группировать данные объема продаж по данному товару, в данном регионе и в данное время. Схема "звезда" с помощью этих фактов и измерений позволяет получать необходимую информацию в нужном формате и при этом не требуется избыточных данных (таких как номер заказа, номер счета, статус и т. д.), которые обычно присутствуют в операционной БД. На самом деле, измерения представляют собой как бы волшебное стекло, сквозь которое мы изучаем факты.

Таблица 13.10. Возможные атрибуты измерений объема продаж

| Измерение      | Описание                                                                                                                                                       | Возможные атрибуты                                                                   |
|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| Местоположение | Все, что является описанием местоположения. Примеры: Нэшвилл, Магазин №101, Южный регион, TN (Теннесси) и т. д.                                                | Регион, штат, город, магазин и т. д.                                                 |
| Товар          | Все, что является описанием продажи товара. Например, товары по уходу за волосами, шампунь, натуральный экстракт, 5,5 унций, бутылка, голубая жидкость и т. д. | Тип товара, идентификатор товара, марка, упаковка, внешний вид, цвет, размер и т. д. |
| Время          | Все, что представляет собой временной интервал по фактам продажи. Например, 1999 год, июль месяц, дата 29.07.1999, время 16:46                                 | Год, квартал, месяц, неделя, день, время или дата и т. д.                            |

С концептуальной точки зрения многомерную модель данных по объемам продаж лучше всего можно представить в виде трехмерного куба. Конечно, не следует полагать, что имеются какие-то ограничения на число измерений, которые могут быть связаны с таблицей фактов, — с математической точки зрения таких ограничений нет. Тем не менее, использование трехмерной модели упрощает визуализацию задачи. В этом трехмерном примере, используя терминологию многомерного анализа данных, куб, изображенный на рис. 13.13, — это представление об объемах продаж по измерениям "товар", "местоположение" и "время".

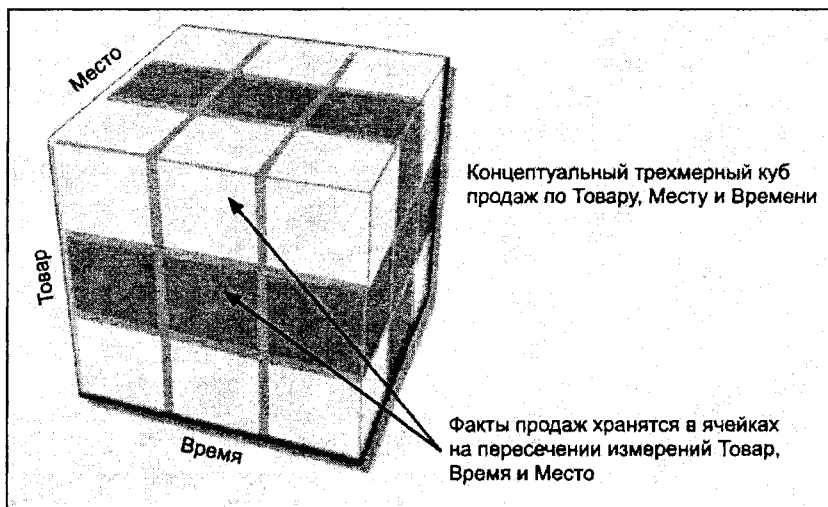
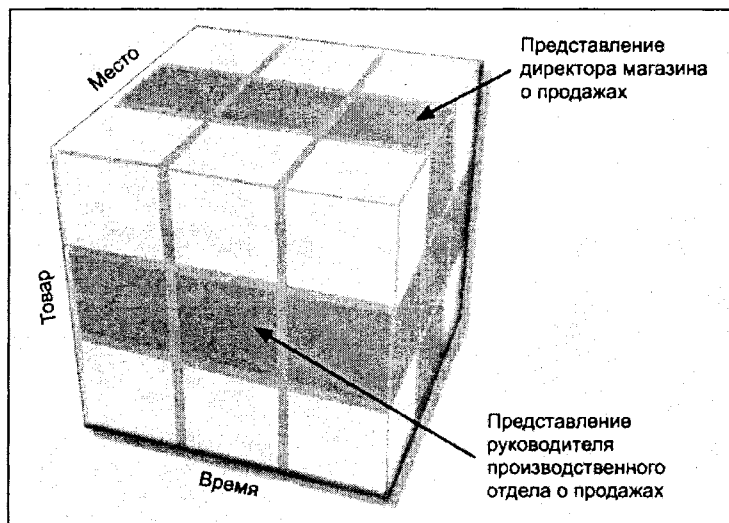


Рис. 13.13. Трехмерное представление об объемах продаж

На рис. 13.13 обратите внимание на то, что каждое значение объема продаж в этом кубе связано с местоположением, товаром и временем. Однако нужно помнить, что этот куб представляет собой только *концептуальное* представление многомерных данных и не показывает, каким образом данные физически размещаются в хранилище. Машина ROLAP хранит данные в СУРБД и использует собственную логику анализа данных и графический интерфейс пользователя для выполнения многомерного анализа. MOLAP-система хранит данные в МСУБД с помощью технологии матричного представления и массивов для моделирования такого куба.

Вне зависимости от выбранной технологии БД, одна из главных особенностей многомерного анализа — способность сфокусироваться на определенных слоях куба. Например, руководитель производственного отдела может быть заинтересован в исследовании общего объема продаж товара, в то время как директор магазина — в исследовании объема продаж товара в данном магазине. В терминах многомерного анализа возможность вырезать определенные слои из куба для выполнения детального анализа называется *исследованием "вдоль и поперек"* (*slice and dice*). Рис. 13.14 иллюстрирует эту концепцию. На рис. 13.14 можно видеть, что срезы куба образуют слои (*slice*). При поперечной нарезке слоев получают небольшие кубики (*dice*), которые представляют собой элементы исследования "вдоль и поперек".



**Рис. 13.14.** Исследование объемов продаж по методу "вдоль и поперек"

Для исследования по методу "вдоль и поперек" мы должны уметь идентифицировать каждый слой и кубик. Это выполняется с помощью значений каждого атрибута в данном измерении. Например, для использования измерения по местоположению мы должны определить атрибут `STORE_ID`, чтобы сфокусироваться на отдельном магазине.

С учетом этих требований к значениям атрибута в слое и кубике снова взглянем на табл. 13.10. Обратите внимание, что каждый атрибут получает дополнительную глубину к фактам объема продаж, т. е. появляются новые способы поиска, классификации и возможного агрегирования информации. Например, измерение по местоположению добавляет географическую перспективу для продаж: регион, штат, город и магазин. Все атрибуты выбраны с целью обеспечения системы поддержки решений для конечных пользователей так, чтобы они могли изучить объемы продаж по всем атрибутам данного измерения.

Необходимо помнить, что время — одно из важнейших измерений. Время предоставляет структуру, относительно которой можно анализировать объемы продаж и, возможно, предсказывать их. Также время играет важную роль, когда аналитик интересуется агрегатными значениями продаж (за неделю, месяц, квартал и т. д.). Учитывая важность и универсальность времени с точки зрения аналитика, многие поставщики используют автоматический контроль времени в своих продуктах организации хранилища данных.

#### 13.5.4. Иерархии атрибутов

Атрибуты внутри измерений можно упорядочивать в строгую иерархию атрибута. Иерархия атрибута представляет собой нисходящую организацию данных, используемую в двух целях: агрегирование информации и углубленный/укрупненный анализ данных. Например, на рис. 13.15 показано, как атрибуты измерения местоположения можно организовать в иерархию по региону, штату, городу и магазину.

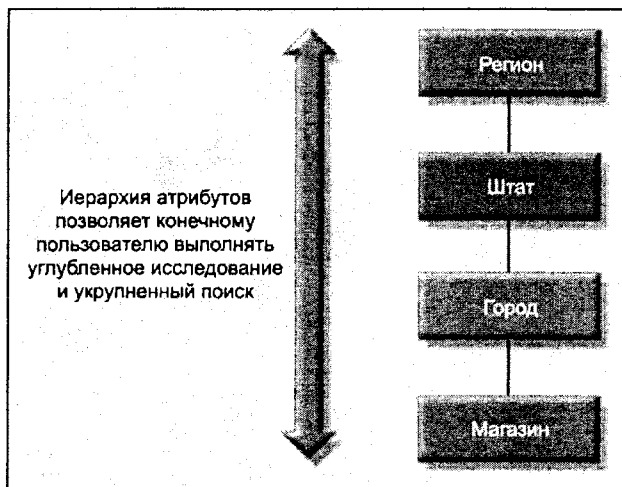


Рис. 13.15. Иерархия атрибута местоположения

Иерархия атрибута обеспечивает возможность выполнять в хранилище данных углубленное и укрупненное исследование. Предположим, что аналитику необходимо



получить ответ на вопрос "Как выглядят в сравнении объемы ежемесячных продаж в 2001 и в 2002 году?" Аналитик определяет резкое снижение продаж в марте 2002. Ему может потребоваться углубленное изучение данных по марту месяца, чтобы сравнить объемы продаж по регионам с предыдущим годом. Поступая так, аналитик может определить, происходит ли спад уровня продаж в марте во всех регионах или только в отдельном регионе. Такой вид углубленного анализа можно даже расширить, когда аналитику требуется определить магазин, продажи в котором ниже нормы.

Только что описанный сценарий возможен, поскольку иерархия атрибутов позволяет определить в хранилище данных и в OLAP-системах путь, который будет определять способ разбиения и агрегирования данных для операций углубленного исследования и укрупненных операций. Нет необходимости включать каждый атрибут в иерархию: некоторые атрибуты существуют сами по себе, обеспечивая подробное описание и измерение. Но необходимо помнить, что атрибуты из различных измерений могут группироваться в иерархии. Например, после углубленного исследования от города до магазина может потребоваться углубленное исследование по измерению "товар", чтобы выявить низкую производительность магазина. Измерение "товар" может быть основано на группе товаров (молочные продукты, мясо и т. д.) или на торговой марке (марка А, марка В и т. д.).

На рис. 13.16 представлен сценарий, в котором аналитик изучает факты объемов продаж при помощи измерений "товар", "время" и "местоположение". В этом примере измерение "товар" установлено как "Все товары", и это означает, что по оси Y вы будете видеть все товары. Измерение "время" (ось X) установлено как "Квартал", и это значит, что данные агрегируются поквартально (например, общий объем продаж товаров А, В и С в кварталах Q1, Q2, Q3 и Q4). Наконец, измерение "местоположение" изначально установлено как "Регион", это гарантирует, что каждая ячейка содержит общий объем продаж по региону для данного продукта в данном квартале.

Простейший сценарий анализа данных, показанный на рис. 13.16, предоставляет аналитику три различных информационных пути. По измерению "товар" (ось Y) аналитик может просмотреть все товары, сгруппированные по типу, или только один товар. По измерению "время" (ось X) аналитик может опросить изменяющиеся во времени данные на различных уровнях агрегации: год, квартал, месяц или неделя. Каждое значение объема продаж изначально показывает общий объем продаж по региону для каждого товара. С помощью графического интерфейса, щелкая кнопкой мыши на ячейке соответствующего региона, аналитик может углубленно исследовать объемы продаж по штатам внутри данного региона. Щелкнув кнопкой мыши еще раз на одном из штатов, можно посмотреть информацию об объеме продаж по каждому городу в штате и т. д.

Как показано в предыдущих примерах, иерархии атрибута определяют, каким образом информация извлекается из хранилища данных и как она представляется пользователю. Информация об иерархии атрибутов хранится в словаре данных СУБД и используется OLAP-системой для обеспечения доступа к хранилищу данных. Как только такой доступ обеспечен, инструментальные средства запроса должны тесно интегрироваться с метаданными хранилища и обеспечивать мощные аналитические возможности.

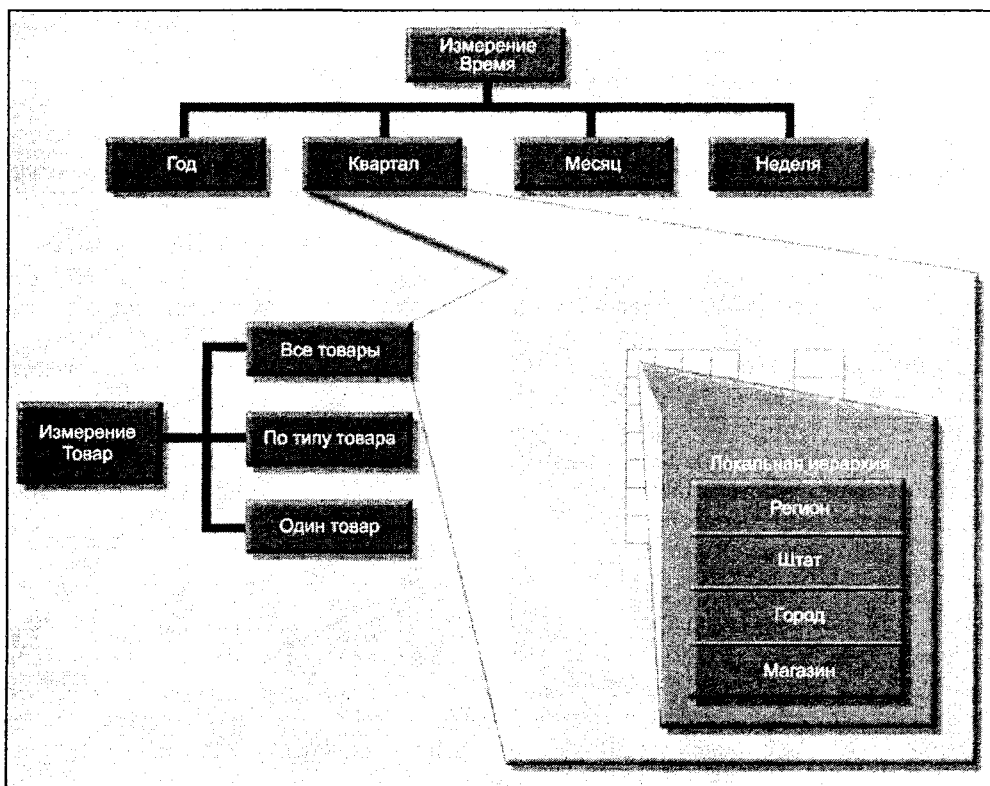


Рис. 13.16. Иерархии атрибутов в многомерном анализе

### 13.5.5. Представление в виде схемы "звезда"

Факты и измерения обычно представлены физическими таблицами в БД хранилища данных. Таблица фактов связана с каждой таблицей измерений связью М:1 (многие-к-одному). Иначе говоря, с каждой строкой измерения связано множество строк фактов. На примере с объемами продаж можно видеть, что каждый товар появляется несколько раз в таблице фактов.

Таблицы фактов и измерений связаны через внешние ключи и на них действуют уже знакомые нам ограничения на первичный и внешний ключ. Первичный ключ на стороне "один" таблицы измерений хранится как часть первичного ключа стороны "многие" таблицы фактов. Поскольку таблица фактов связана с несколькими таблицами измерений, первичный ключ таблицы фактов является составным первичным ключом. На рис. 13.17 представлены связи между таблицей фактов объемов продаж и таблицами измерений "товар", "местоположение" и "время". Чтобы показать, как можно без труда расширить схему "звезда", добавим в нее измерение CUSTOMER (клиент). Для добавления измерения CUSTOMER необходимо просто

включить идентификатор CUST\_ID в таблицу фактов SALES и добавить в базу данных таблицу CUSTOMER.

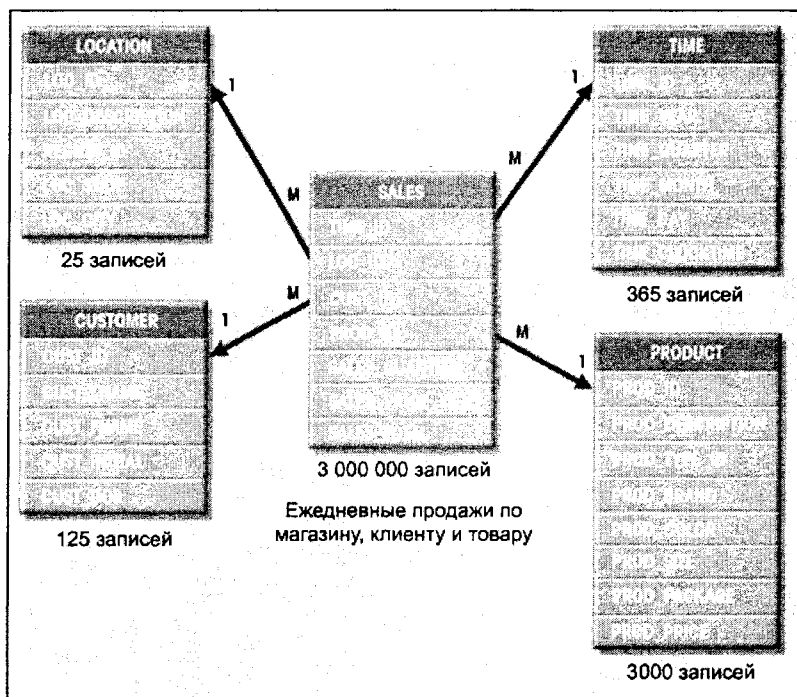


Рис. 13.17. Схема "звезда" для объемов продаж

В составной первичный ключ для таблицы фактов по объему продаж входят идентификаторы TIME\_ID (идентификатор времени), LOCATION\_ID (идентификатор местоположения), CUST\_ID (идентификатор клиента) и PRODUCT\_ID (идентификатор товара). Каждая запись в таблице фактов по объему продаж уникально идентифицируется комбинацией значений каждого из внешних ключей таблицы фактов. По умолчанию первичный ключ таблицы фактов всегда составляется из внешних ключей, указывающих на каждую таблицу измерений, с которой он связан<sup>2</sup>. В данном случае каждая запись по объему продаж представляет каждый товар, проданный отдельному клиенту в определенное время и в определенном месте. В этой схеме таблица измерения по времени представлена по дням, поэтому таблица фактов по объему продаж представляет ежедневные продажи, агрегированные по товару и по клиенту. Поскольку таблицы фактов содержат фактические значения, используемые в процессе поддержки решения, эти значения повторяются в таблице фактов не-

<sup>2</sup> Возможен специальный случай схемы "звезда", который называется *схемой с множеством звезд*, в том случае, если первичный ключ таблицы фактов не определяет уникальное значение. Подробнее см. Vidette Poe, "Building a Data Warehouse for Decision Support", Prentice Hall, 1996.

сколько раз. Поэтому таблицы фактов всегда являются самыми большими в схеме "звезда". Поскольку таблицы измерений содержат только неповторяющуюся информацию (все продавцы уникальны, все товары уникальны и т. д.), то эти таблицы всегда меньше, чем таблицы фактов.

В типичной схеме "звезда" каждая запись измерения связана с тысячами записей фактов. Например, "спойлер" появляется только один раз в измерении "товар", но может появляться тысячи раз в таблице фактов продаж. Это свойство схемы "звезда" упрощает функции извлечения данных, поскольку большую часть времени аналитики тратят на поиск фактов по атрибутам измерения. Поэтому СУБД, оптимизированная под систему поддержки решений хранилища данных, прежде всего, ищет наименьшую таблицу измерений, прежде чем получить доступ к большим таблицам фактов.

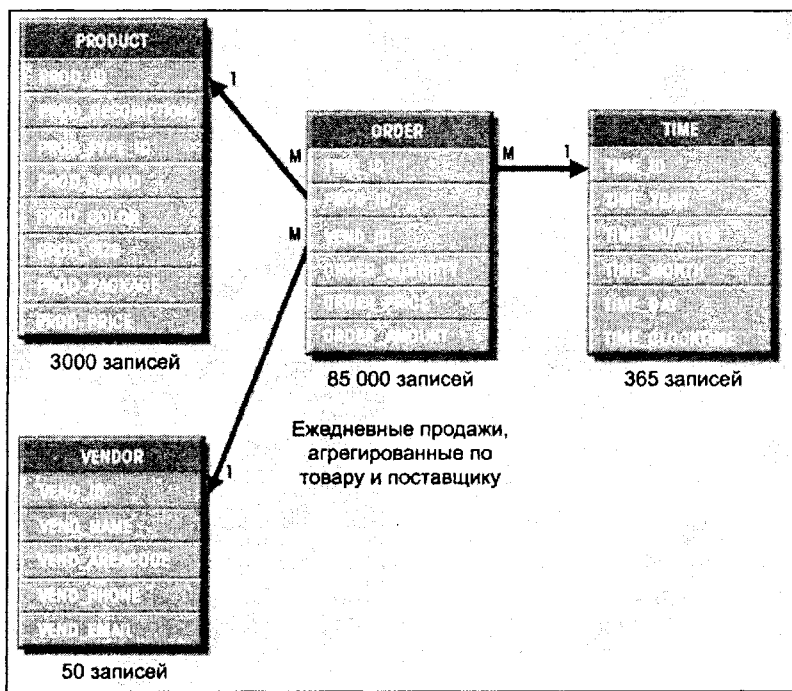


Рис. 13.18. Схема "звезда" для заказов

В хранилище данных обычно имеется множество таблиц фактов. Каждая таблица фактов разрабатывается для специфической проблемы системы поддержки решений. Предположим, что нас интересуют новые заказы и в то же время нам требуется и изначальная информация об объемах продаж. В этом случае необходимо иметь и таблицу фактов заказа, и таблицу фактов объема продаж в одном и том же хранилище данных. Если заказы являются ключевым интересом предприятия, таблица фактов заказа должна быть центральной в схеме "звезда" и должна иметь измерения

"поставщик", "товар" и "время". Тогда интерес в информации о поставщике потребует нового измерения "поставщик", представленного новой таблицей **VENDOR** в базе данных. Измерение "товар" представлено той же таблицей **PRODUCT**, которую мы использовали в исходной схеме "звезда" для объемов продаж. Однако с учетом необходимости получения информации о заказах, а также о продажах, на измерение "время" в этом случае надо обратить особое внимание. Если отдел заказов использует те же отчетные периоды времени, что и отдел сбыта, то время может быть представлено той же самой таблицей. Если же используются различные периоды времени, то мы должны создать другую таблицу времени, возможно, с названием **ORDER\_TIME**, чтобы представлять периоды времени, используемые отделом заказов. На рис. 13.18 в схеме "звезда" для заказов совместно используются измерения "товар", "поставщик" и "время".

Несколько таблиц фактов могут создаваться и с целью повышения производительности и улучшения семантики. В следующем разделе будут обсуждаться некоторые технические приемы по улучшению производительности, используемые в схеме "звезда".

### 13.5.6. Технические приемы повышения производительности

Создание базы данных, обеспечивающей быстрый и точный доступ к запросам анализа данных, является одной из основных целей проектирования хранилища данных. Поэтому действия по повышению эффективности могут быть нацелены на упрощение SQL-кода, а также на улучшение семантического представления измерений предприятия. Чаще всего для оптимизации проекта хранилища данных используются четыре технических приема.

- Нормализация таблиц измерений.
- Представление различных уровней агрегации несколькими таблицами фактов.
- Денормализация таблиц фактов.
- Разбиение и репликация таблиц.

#### Нормализация таблиц измерений

*Нормализация* выполняется для упрощения навигации пользователя по измерениям. Например, если таблица измерения по местоположению содержит транзитивные зависимости между регионами, штатами и городами, то мы можем привести эти отношения к 3НФ (третья нормальная форма), как показано на рис. 13.19. (При необходимости обратитесь к гл. 4, где техника нормализации рассматривается подробно.)

Нормализуя таблицы измерений, мы упрощаем операцию фильтрования данных, связанных с измерениями. В данном примере регион, штат, город и местоположение содержат очень немного по сравнению с таблицей фактов продаж. Только таблица местоположения напрямую связана с таблицей фактов продаж.

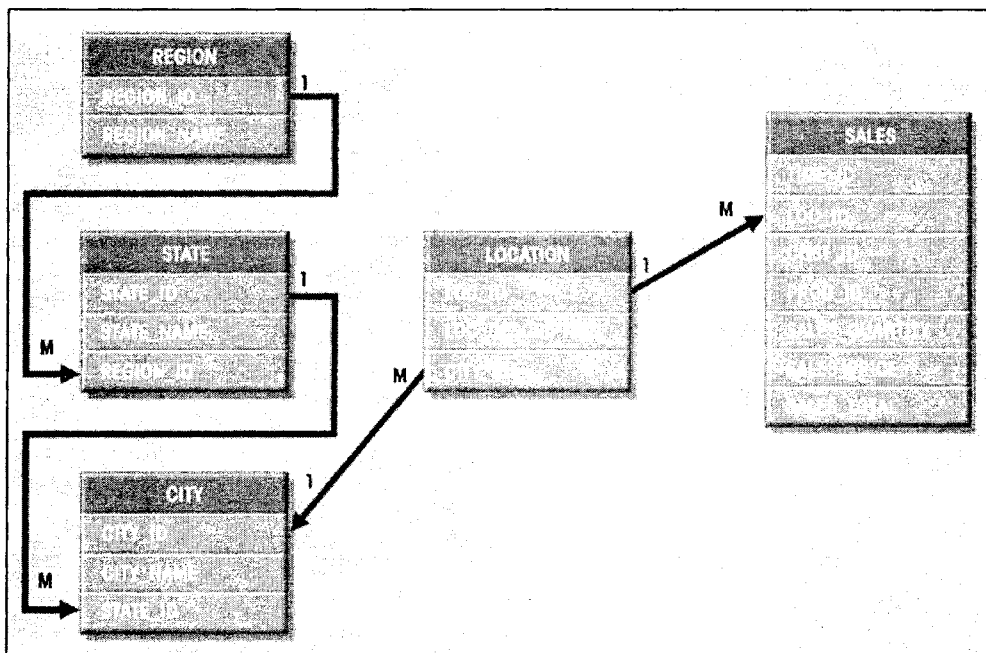


Рис. 13.19. Нормализованная таблица измерений

## Представление различных уровней агрегации несколькими таблицами фактов

Мы можем увеличить скорость выполнения запросов, создавая и поддерживая несколько таблиц фактов на каждом уровне агрегации (регион, штат и город) по измерению "местоположение". Эти агрегатные таблицы рассчитываются предварительно, на этапе загрузки данных, а не во время работы системы. Цель этого технического приема состоит в экономии времени центрального процессора во время работы системы, что увеличит скорость анализа данных. С помощью инструментария запросов конечного пользователя, оптимизированного для анализа решений, затем осуществляется доступ к итоговым таблицам фактов, вместо того, чтобы рассчитывать значения, обращаясь к таблицам фактов нижнего уровня. Этот технический пример представлен на рис. 13.20. Используя начальный пример с объемом продаж, мы добавляем агрегатные таблицы фактов для региона, штата и города.

Проектировщик хранилища данных должен определить, какой уровень агрегации необходимо выполнить на предварительном этапе и хранить его затем в БД. Множество агрегатных таблиц фактов обновляются при каждом процессе загрузки в пакетном режиме. И поскольку цель состоит в минимизации времени доступа и обработки, проектировщик хранилища данных должен выбрать агрегатные таблицы фактов в соответствии с ожидаемой частотой использования и временем обработки, необходимым для расчета данного уровня агрегации во время работы системы.

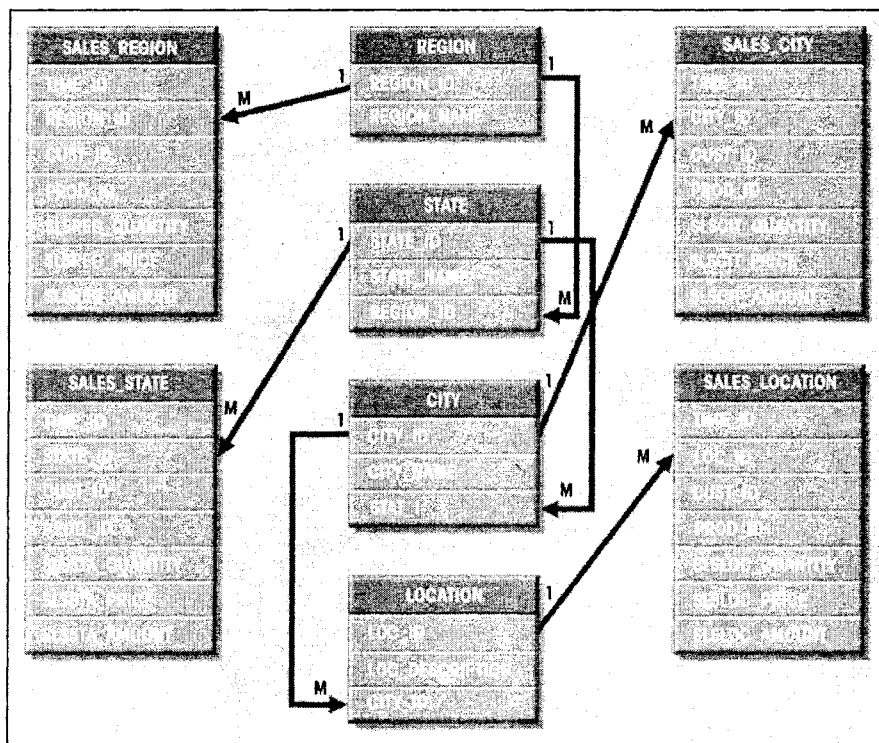


Рис. 13.20. Несколько таблиц фактов

## Денормализация таблиц фактов

Денормализация таблиц фактов повышает эффективность доступа к данным и экономит пространство для хранения данных. Последняя проблема (экономия места хранения) в настоящее время практически снята с повестки дня. Стоимость хранения информации уменьшается с каждым днем, и ограничения СУБД на размер базы данных и таблиц, размер записи и максимальное число записей в одной таблице сдерживают больше, чем затраты на оперативную память.

Денормализация повышает производительность, поскольку при этом данные, которые обычно хранятся в нескольких записях, будут храниться в одной записи. Например, чтобы рассчитать общий объем продаж по всем товарам во всех регионах, мы должны получить доступ к объему продаж по регионам, агрегировать и суммировать все записи в этой таблице. Если мы имеем дело с 300 000 продаж, то нам придется просуммировать, по крайней мере, 300 000 строк. Хотя это и не такая дорогостоящая операция для СУБД, но выполнение подобной операции, скажем, за десятилетний период может просто привести к краху системы. В таких случаях полезно иметь специальные денормализованные агрегатные таблицы. Например, таблица **YEAR\_TOTALS** (годовые итоги) может содержать следующие поля: **YEAR\_ID**,

MONTH\_1, MONTH\_2, ..., MONTH\_12 и сумму по каждому году. Такие таблицы можно легко использовать для выполнения сравнения по годам на уровне месяца, квартала или года. Здесь снова критерий проекта, например, частота обращения или требования производительности, должен оцениваться по возможной перегрузке СУБД при управлении денормализованными отношениями.

## Разбиение и репликация таблиц

Поскольку мы уже подробно рассматривали эти технические приемы в гл. 10, здесь мы их обсудим только по отношению к хранилищу данных. Разбиение и репликация таблиц особенно важны при реализации DSS в геоинформационных системах. При *разбиении* таблицы расчленяются на подмножества строк или столбцов, и эти подмножества размещаются как можно ближе к клиентскому компьютеру с тем, чтобы уменьшить время доступа к данным. При *репликации* копии таблицы размещаются в разных местах, что также позволяет уменьшить время доступа к данным.

Вне зависимости от того, какая используется схема повышения производительности, время является одним из самых важных измерений при анализе данных. Поэтому, как правило, имеется одна таблица фактов для каждого уровня агрегации, определенного по измерению "время". Например, в примере с объемами продаж можно создать пять агрегированных таблиц фактов продаж: ежедневные продажи, еженедельные, ежемесячные, квартальные и ежегодные. Эти таблицы фактов должны иметь явную или неявную периодичность. *Периодичность (periodicity)* обычно выражается только текущим годом, предыдущими годами или всеми годами, предоставляя информацию о диапазонах времени для данных, хранящихся в таблице.

В конце каждого года ежедневные продажи текущего года перемещаются в другую таблицу, в которой хранятся только данные по ежедневным продажам за предыдущий год. Эта таблица фактически содержит все записи продаж с начала проведения таких операций, за исключением текущего года. Данные в таблицах по текущему году и предыдущим годам, таким образом, представляют собой полную историю продаж компании. Можно реплицировать таблицы с данными по продажам за предыдущие годы по нескольким местам, чтобы избежать удаленного доступа к ретроспективным данным, который будет выполняться медленно. Предполагаемый размер этой таблицы достаточен, чтобы запугать всех, кроме бесстрашного оптимизатора запросов. Это именно тот случай, когда денормализация очень важна.

## 13.6. Реализация хранилища данных

На разработку информационной системы организации накладывается много ограничений. Часть из них связана с финансированием, другие ограничения зависят от того, как руководство представляет себе роль отдела информационных систем, и от уровня требований к информации. Поэтому вместо того, чтобы изучать методологию проектирования и реализации специфического отдельного хранилища данных, мы определим несколько факторов, по-видимому, являющихся общими для всех хранилищ данных.



### 13.6.1. Хранилище данных как активная среда поддержки решений

Может быть, первое, о чем надо помнить, — это то, что хранилище данных не является статичной базой данных. Наоборот, хранилище — это динамичная среда поддержки решений, которая, почти по определению, постоянно развивается. Поскольку хранилище данных является основой современных систем DSS, при его проектировании и реализации мы включаемся в процесс проектирования и реализации всей инфраструктуры базы данных для обеспечения системы поддержки решений в данной компании. Хотя проще всего основное внимание уделить хранилищу данных как централизованному складу данных для системы DSS, необходимо помнить, что инфраструктура поддержки решений включает в себя еще и аппаратное обеспечение, программное обеспечение, людей и процедуры, а также данные. Считать, что только хранилище данных, как самая важная составляющая, делает успешной систему DSS, — такое же заблуждение, как полагать, что для жизнедеятельности человека нужны только сердце и мозг. Хранилище данных, безусловно, является важнейшим компонентом современных систем DSS, но далеко не единственным. Поэтому проектирование и реализацию необходимо исследовать в свете всей инфраструктуры.

### 13.6.2. Усилия всего предприятия с привлечением пользователей и согласованием на всех уровнях

Очевидно, что предприятие в целом должно получить определенные выгоды от хранилища данных как основного компонента инфраструктуры системы поддержки решений. Приступить к проектированию хранилища данных означает задаться целью помочь разработать интегрированную модель данных, охватывающую данные, которые являются важнейшими для всего предприятия, как с точки зрения конечного пользователя, так и со стороны перспектив предприятия. Информация хранилища данных выходит за рамки предприятия и географические границы. Поскольку хранилище данных представляет собой попытку моделирования всех данных предприятия, скорее всего, обнаружится, что компоненты предприятия (подразделения, отделы, группы поддержки и т. д.) зачастую преследуют противоположные цели, обычно это приводит к противоречивости информации и к ее избыточности. Информация — это власть, управление ее источниками и использование, вероятно, может привести к распрям, сопротивлению конечных пользователей и к напряженной борьбе на всех уровнях. Разработка отличного хранилища данных состоит не только в понимании того, как создать схему "звезда", это требует еще и организаторского мастерства, умения разрешать конфликты, использования посреднических усилий и проведения разбирательств. Короче говоря, проектировщик должен:

- ☐ привлекать к процессу проектирования конечных пользователей;
- ☐ с самого начала стремиться выполнять обязательства перед конечными пользователями;
- ☐ наладить постоянный обмен мнениями с конечным пользователем;
- ☐ учитывать ожидания конечных пользователей;
- ☐ определить процедуры разрешения конфликтов.

### 13.6.3. Данные, анализ и пользователи

Одни высокие организаторские способности не гарантируют успеха. Необходимо учитывать и технические аспекты хранилища данных. Старая схема "ввод-процесс-вывод" здесь применима в полной мере. Проект хранилища данных должен удовлетворять следующим критериям:

- ☐ интеграция данных и критерии загрузки;
- ☐ возможность анализа данных и приемлемая эффективность запросов;
- ☐ потребности пользователя в анализе данных.

Передовые технические решения при реализации хранилища данных должны обеспечивать поддержку решений для конечного пользователя расширенными возможностями анализа данных — в нужный момент, в нужном формате, на основе правильных данных и по приемлемой цене.

### 13.6.4. Использование процедур проектирования баз данных

Мы описали жизненный цикл базы данных и процесс проектирования базы данных в гл. 6, поэтому правильно будет начать с обзора традиционных процедур проектирования данных. Эти процедуры проектирования должны затем адаптироваться к требованиям хранилища данных. Поскольку хранилище данных получает данные из операционных баз данных, понятно, почему нужно иметь надежно спроектированную операционную базу данных. (Трудно получить хорошие данные в хранилище, если операционные данные некорректны.) На рис. 13.21 в упрощенном виде представлен процесс реализации хранилища данных.

Разработка хранилища данных это плод усилий всей компании, которые потребуют привлечения различных ресурсов: человеческих, финансовых и технических. Для обеспечения функционирования системы поддержки решений в рамках предприятия требуется высококачественная архитектура, основанная на опыте людей, технологии и организационных мероприятиях, которые зачастую трудно найти и реализовать. Например:

- ☐ огромные объемы данных системы поддержки решений, вероятно, потребуют использования самых последних программных и аппаратных средств, т. е. самые последние модели компьютеров с несколькими процессорами, передовые системы баз данных, устройства хранения больших объемов информации и т. д. Не так давно такие требования могли выполняться только в системах с мэйнфреймом. Сегодняшние клиент/серверные технологии предлагают различные решения по реализации хранилища данных;
- ☐ также важно иметь процедуры упорядочивания (оркестровки) потока данных от операционных БД к хранилищу данных. Управление потоком данных включает в себя извлечение данных, проверку и интеграцию;
- ☐ для реализации и поддержки архитектуры хранилища необходим персонал с соответствующими навыками работы по проектированию баз данных, интеграции программного обеспечения, а также с опытом управленческой работы.

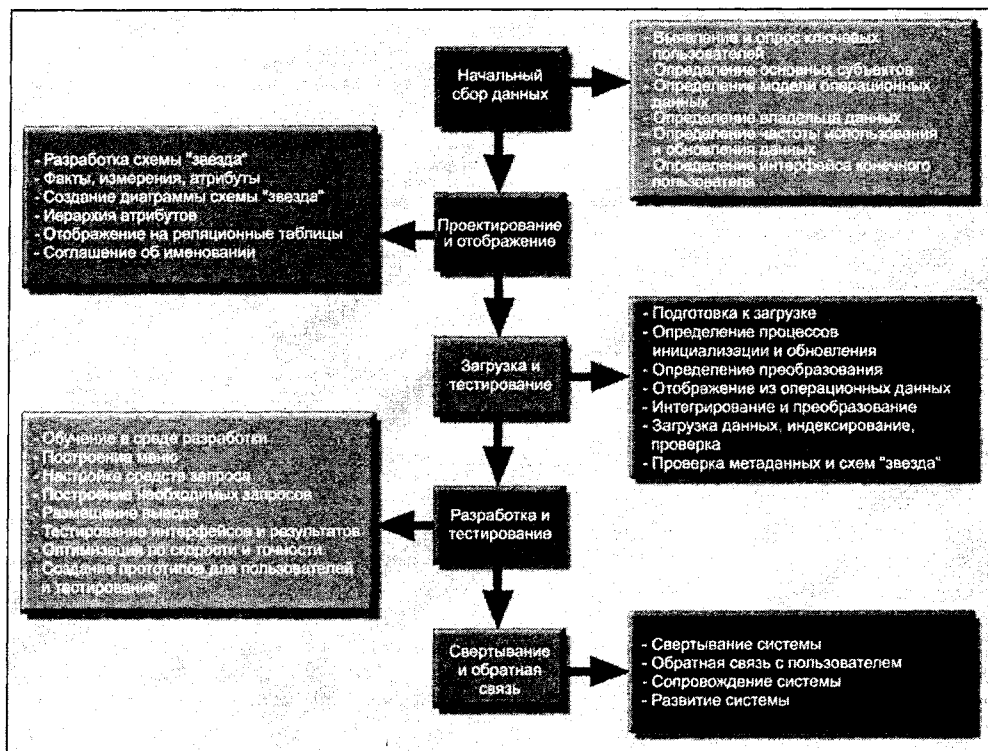


Рис. 13.21. Схема реализации хранилища данных

## 13.7. Добыча данных (data mining)

Цель анализа данных — выявить прежде неизвестные свойства, связи, зависимости или тенденции. Такие открытия затем становятся частью информационной среды, на которой строятся решения. *Типичные инструментальные средства анализа данных предполагают, что конечные пользователи ставят задачу, осуществляют выборку данных и инициируют анализ данных с целью получения информации, помогающей решить возникшие проблемы.* Иначе говоря, конечный пользователь реагирует на внешнее воздействие — на проблему как таковую. Если конечный пользователь не может выявить проблему, то никакие действия предприниматься не будут. Принимая во внимание эти ограничения, в некоторых современных системах DSS теперь поддерживаются различные типы автоматических извещений. Эти извещения представляют собой программные агенты, постоянно наблюдающие за некоторыми параметрами, например, за индикаторами продаж и уровнями запасов, и выполняющие специфические действия (отсылку электронной почты или выдачу предупреждающего сообщения, запуск программ и т. д.), когда контролируемые параметры достигают заданного критического уровня.

В отличие от традиционных (реактивных) инструментальных средств DSS, средства добычи данных являются превентивными (профилактическими). То есть вместо того, чтобы заставить пользователя выявлять проблему, выбирать данные и инструментарий для анализа этих данных, *инструменты добычи данных автоматически ищут в данных аномалии и возможные связи, тем самым выявляя проблемы, еще не выявленные конечным пользователем*. Другими словами, инструменты добычи данных (data mining) анализируют данные, выявляют проблемы или возможности, скрытые в связях данных, формируя компьютерные модели на основе найденных сведений, и затем используют эти модели для предсказания поведения бизнеса — при минимальном вмешательстве пользователя. Конечный пользователь может поэтому использовать обнаруженные сведения для получения знаний, которые могут обеспечить конкурентоспособные решения. Добыча данных представляет собой новое поколение специализированных средств систем поддержки решений, позволяющее автоматизировать анализ данных. Короче говоря, средства добычи данных — основанные на алгоритмах, формирующих функциональные блоки искусственного интеллекта, нейронных сетей, выводе правил по методу индукции и логики предикатов, инициируют анализ для получения знаний.

С учетом этого мы можем определить добычу данных как методологию, созданную для выполнения быстрого получения знаний на основе данных БД при минимальном участии конечного пользователя на этапе фактического получения знаний. Для наглядного представления этого определения используем пирамиду, изображенную на рис. 13.22, где представлены три ступени: неупорядоченные ("сырые") данные, упорядоченные данные и, наконец, знания. Неупорядоченные данные составляют основу пирамиды и представляют собой информацию, хранимую предприятием в операционных БД. Второй уровень содержит упорядоченные данные, которые представляют собой очищенную и обработанную информацию. Упорядоченная информация — основа принятия решений и осмысления бизнеса. Знания находятся на вершине пирамиды и представляют собой высоко специализированную информацию.

Трудно привести точный список характеристик средств добычи данных. С одной стороны, в современных поколениях средств добычи данных имеется множество различных приложений, удовлетворяющих всем требованиям добычи данных. Кроме того, существование множества разновидностей таких систем определяется отсутствием стандартов, управляющих созданием средств добычи данных. Говоря кратко, в каждом средстве добычи данных используются различные подходы, таким образом создаются семейства средств добычи данных, которые занимают соответствующие ниши на рынке (в области маркетинга, розничной продажи, финансов, здравоохранения, инвестиций, страхования и банковского дела). Внутри данной ниши средства добычи данных могут использовать определенные алгоритмы, а эти алгоритмы могут быть реализованы различными способами и применяться к различным данным.

Несмотря на отсутствие должной стандартизации, можно утверждать, что добыча данных состоит из четырех этапов:

1. Подготовка данных.
2. Анализ данных и их классификация.
3. Приобретение знаний.
4. Прогноз.

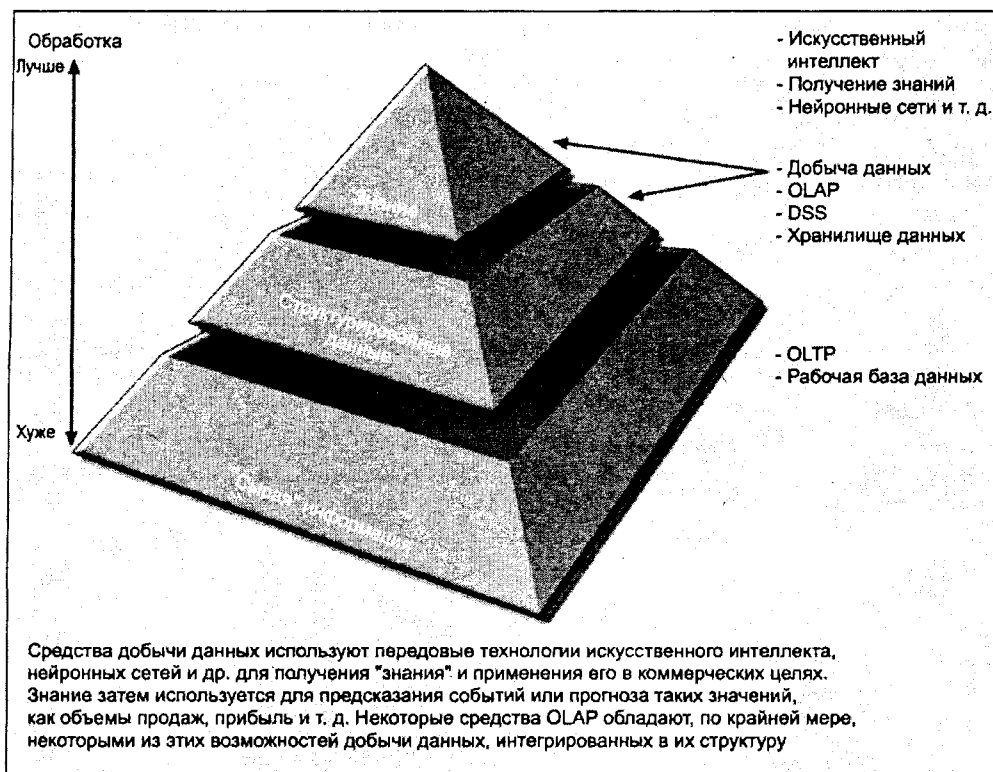


Рис. 13.22. Получение знаний из данных

На этапе *подготовки* основные наборы данных, которые будут использоваться в операции добычи данных, идентифицируются и очищаются от ненужных сведений. Поскольку информация в хранилище данных уже интегрирована и отфильтрована, операции добычи данных, как правило, базируются на информации хранилища.

На этапе *анализа и классификации* данные изучаются с целью определения общих характеристик данных или шаблонов. На этом этапе средства добычи данных используют специальные алгоритмы для выявления:

- ☐ группирования данных, классификаций, кластеров или последовательностей;
- ☐ зависимости и связи (links, relationships) данных;
- ☐ шаблонов данных, тенденций и отклонений.

На этапе *приобретения знаний* используются результаты этапа анализа и классификации данных. Здесь с помощью средств добычи данных (с возможным вмешательством конечного пользователя) выбираются подходящие алгоритмы моделирования и приобретения знаний. Наиболее типичные алгоритмы, используемые при добыче данных, основаны на нейронных сетях, деревьях решений, выводе правил методом индукции, генетических алгоритмах, классификационных и регрессионных деревьях, выводах путем сопоставления, визуализации ближайшего соседа и данных. В средст-

вах добычи данных для создания компьютерной модели, отражающей поведение наборов данных, многие из этих алгоритмов могут использоваться в различных комбинациях.

Хотя некоторые средства добычи данных заканчивают свою деятельность на этапе приобретения знаний, другие продолжают работать и на этапе *прогноза*. Здесь полученные сведения используются для предсказания последующего поведения и прогноза результатов бизнеса. Вот примеры сведений, полученных средствами добычи данных:

- ❑ 65% клиентов, которые не пользуются кредитной карточкой последние 6 месяцев, в 88% случаев закрывают свой счет;
- ❑ 82% клиентов, которые приобрели новый телевизор с 27-дюймовым (или большего размера) экраном, в 90% случаях в последующие 4 недели приобретают оборудование для домашнего кинотеатра;
- ❑ если возраст клиента меньше 30 лет, доход не выше 25 000 долларов, кредитный рейтинг ниже 3, а размер кредита более 25 000 долларов, то минимальный срок ссуды равен 10 годам.

Полный набор добытых сведений можно представить в виде дерева решений, нейронной сети, модели прогнозирования или визуального интерфейса, которые затем используются для проектирования последующих событий и результатов. Например, на этапе прогноза можно проектировать вероятные результаты свертывания выпуска нового товара или продвижения его на рынке. На рис. 13.23 представлены различные этапы технологии добычи данных.

Поскольку технология добычи данных находится на начальной стадии своего развития, некоторые из сведений, полученных методами добычи данных, могут быть неожиданными для руководства. Например, с помощью средств добычи данных может обнаружиться тесная связь между маркой газировки, которую предпочитает клиент, и маркой колес его автомобиля. Очевидно, такая связь не имеет никакого значения для торговых агентов. (В регрессионном анализе такие связи называют "идиотской корреляцией"). К счастью, добыча данных обычно дает более значимые результаты. На самом деле, добыча данных оказывается весьма полезной при поиске связей между данными, позволяя определить предпочтения клиентов, улучшить распространение и признание товара, уменьшая возможность нанесения вреда здоровью, помогая анализировать состояние рынка и т. д., как указано в следующем примечании.

### Примечание

Телефонная компания использовала средства добычи данных для анализа хранилища данных своих клиентов. С помощью добычи данных были найдены порядка 10 000 предположительно "резидентных" (местных) клиентов, платящих за телефон больше \$1000 долларов в месяц. При дальнейшем исследовании хранилища телефонная компания обнаружила несколько предприятий малого бизнеса, которые вообще пытались избежать оплаты (Cheryl D. Krivda, "Data-Mining Dynamite", *BYTE*, октябрь 1995, т. 20, № 10, стр. 203—205).

Технологии добычи данных обладают большим потенциалом и помогают преодолеть очередной рубеж в развитии баз данных. В идеальном случае ожидается появление баз данных, позволяющих не только хранить данные и различную статистическую информацию, но и извлекать знания из хранимой информации. Такие системы управления

базами данных, называемые также *индуктивными* или *интеллектуальными базами данных*, являются предметом исследований многих лабораторий. Хотя такие базы данных еще должны проложить себе путь на рынок, но и модули расширения и интегрированные в СУБД средства добычи данных уже получили широкое распространение на рынке разработки хранилищ данных. В табл. 13.11 указаны поставщики современного программного обеспечения хранилища данных и добычи данных.

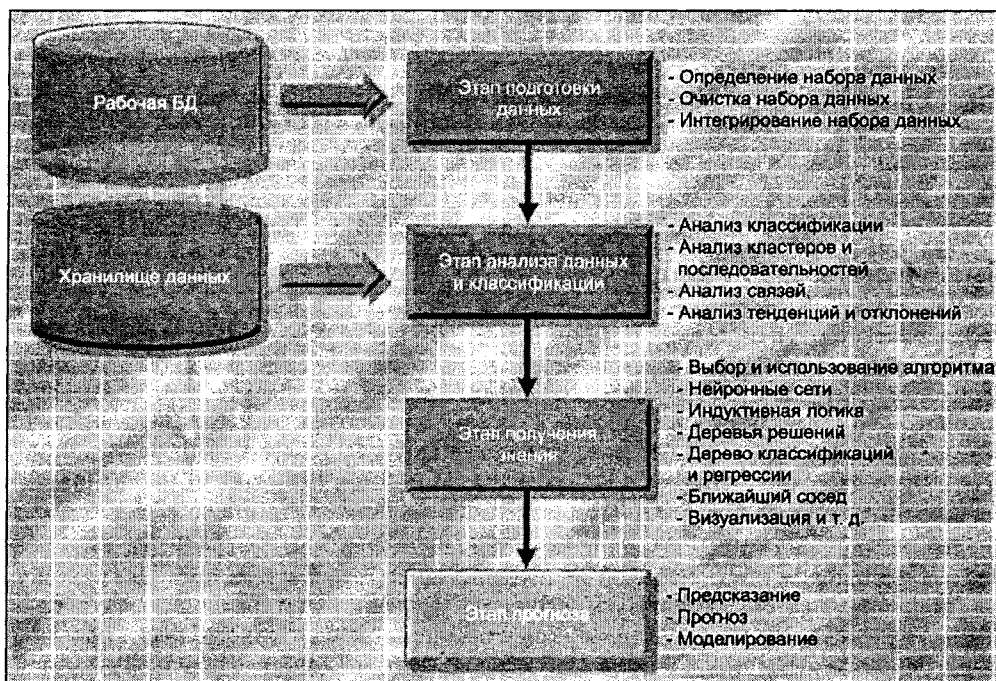


Рис. 13.23. Этапы добычи данных

Таблица 13.11. Поставщики современных программных средств создания хранилищ данных и добычи данных

| Поставщик        | Продукт                     | Web-адрес                                                            |
|------------------|-----------------------------|----------------------------------------------------------------------|
| SAS Institute    | SAS Warehouse Administrator | <a href="http://www.sas.com">www.sas.com</a>                         |
| IBM              | DB2 Warehouse Manager       | <a href="http://www.software.ibm.com">www.software.ibm.com</a>       |
| Informix         | MetaCube                    | <a href="http://www.informix.com">www.informix.com</a>               |
| IBM-Informix     | Red Brick Decision Server   | <a href="http://www.redbrick.com">www.redbrick.com</a>               |
| Brio Techlogy    | Brio.Enterprise             | <a href="http://www.brio.com">www.brio.com</a>                       |
| Business Objects | BusinessObjects             | <a href="http://www.businessobjects.com">www.businessobjects.com</a> |

Таблица 13.11 (окончание)

| Поставщик        | Продукт                      | Web-адрес                                                              |
|------------------|------------------------------|------------------------------------------------------------------------|
| Oracle           | Oracle Express               | <a href="http://www.oracle.com">www.oracle.com</a>                     |
| Cognos           | Cognos Impromptu             | <a href="http://www.cognos.com">www.cognos.com</a>                     |
| MicroStrategy    | MicroStrategy Agent          | <a href="http://www.microstrategy.com">www.microstrategy.com</a>       |
| Hyperion         | Hyperion Essbase Server      | <a href="http://www.hysoft.com">www.hysoft.com</a>                     |
| Crystal Decision | Holos                        | <a href="http://www.crystaldecisions.com">www.crystaldecisions.com</a> |
| Hyperion         | Hyperion Essbase OLAP Server | <a href="http://www.hysoft.com">www.hysoft.com</a>                     |

*Примечание:* ссылки (URL) могли измениться после слияния компаний. Для более точных сведений обратитесь на сайт [www.dwinfocenter.org](http://www.dwinfocenter.org).

## Резюме

Чтобы руководители могли принимать стратегические и тактические решения для выживания и процветания в динамической среде современного бизнеса, им требуется быстрый доступ к точной информации. Анализ данных используется для структурирования и интерпретации информации. Поэтому анализ данных играет важную роль (некоторые считают, что самую важную) при выработке успешного решения. Учитывая столь высокое значение анализа данных, многие бизнесмены организовали своих аналитиков в формальные группы "поддержки решений".

Поддержка решений связана с методологией (или серией методологий), разработанной для извлечения информации и использования ее в качестве основы для принятия решений. Системы поддержки решений (DSS-системы) представляют собой упорядоченные компьютеризованные средства, помогающие руководству предприятий принимать решения. Система DSS обычно состоит из четырех основных компонентов высокого уровня: склад данных, извлечение и фильтрация данных, средства создания запросов конечных пользователей и средства создания представлений данных для конечных пользователей. Системы DSS используются на уровне стратегии и тактики руководства организации, но польза их напрямую зависит от качества сбора информации на операционном уровне. Операционные данные не годятся для принятия решений. С точки зрения конечного пользователя данные DSS отличаются от операционных данных по трем основным параметрам: временному диапазону, степени детализации и наличию качественных характеристик (измерений).

Требования к СУБД DSS можно разделить на четыре основные категории: схема базы данных, извлечение и загрузка данных, аналитический интерфейс конечного пользователя и требования к размеру БД. Схема базы данных DSS должна поддерживать сложное (ненормализованное) представление данных. В БД DSS содержатся агрегированные и подытоженные данные. Хотя БД DSS может содержать данные из различных внешних источников, чаще всего их получают из операционных баз данных. СУБД должна поддерживать расширенные возможности извлечения и фильтрации данных. СУБД DSS предоставляет методы поддержки расширенного моделирования и представления данных.



Хранилище данных представляет собой интегрированную предметно-ориентированную долговременную базу данных, которая обеспечивает поддержку решений. Хранилище данных обычно представляет собой базу данных, предназначенную только для чтения и оптимизированную для анализа данных и обработки запросов. Аналитики бизнеса для извлечения данных в приемлемом формате получают доступ к базе данных посредством интерфейсных программ и пользовательских приложений. Витрина данных — это небольшое, ориентированное на одну предметную область подмножество хранилища данных, которое обеспечивает поддержку решений для небольших групп пользователей.

Система оперативного анализа данных (Online Analytical Processing, OLAP) связана со средой расширенного анализа данных, которая обеспечивает поддержку решений, бизнес-моделирование и исследование операций. OLAP-системы обладают четырьмя основными характеристиками: использование технологии многомерного анализа данных, расширенная поддержка баз данных, простой и понятный интерфейс конечного пользователя и клиент/серверная архитектура. OLAP-системы основаны на клиент/серверной технологии и состоят из трех основных модулей: графический интерфейс пользователя OLAP, логика аналитической обработки OLAP и логика обработки данных OLAP. Реляционные OLAP-системы (ROLAP) обеспечивают функциональные возможности OLAP с помощью реляционных баз данных и известных средств создания реляционных запросов для хранения и анализа многомерных данных. Такой подход основан на реляционных технологиях и представляет собой естественный выбор для тех организаций, которые уже используют СУРБД. Многомерные системы оперативной обработки данных (MOLAP) обеспечивают функциональные возможности OLAP-систем с помощью систем управления многомерными базами данных (МСУБД), позволяющих хранить и анализировать многомерные данные. В МСУБД используются специальные технологии для хранения данных в виде матричных  $n$ -мерных массивов.

Схема "звезда" представляет собой технику моделирования данных, используемую при отображении многомерных данных систем поддержки решений на реляционные БД с целью выполнения расширенного анализа информации. Схема "звезда" предоставляет удобную и понятную модель для анализа многомерных данных. В основе схемы "звезда" лежат четыре компонента: факты, измерения (качественные характеристики), атрибуты и иерархии атрибутов. Факты это числовые значения параметров, представляющих различные аспекты бизнеса или деятельности предприятия. Измерения это основные качественные показатели, дополнительно характеризующие данный факт. Каждая таблица измерений состоит из атрибутов, которые часто используются для поиска, фильтрации или классификации фактов. Концептуально многомерную модель данных можно представить в виде трехмерного куба. Атрибуты можно объединить в строгую иерархию атрибутов. Иерархия атрибутов организована в нисходящем порядке и используется в двух целях: объединение (агрегирование) данных и обеспечение возможности углубленного и укрупненного анализа данных. Факты и измерения обычно представлены в виде физических таблиц в БД хранилища. Таблица фактов связана с каждой таблицей измерений связью "многие-к-одному" (M:1). Первичный ключ таблицы фактов формируется путем комбинации внешних ключей каждой таблицы измерений, с которой она связана. Для оптимизации проекта хранилища данных используются четыре основные методологии: нормализация таблиц измерений, использование нескольких таблиц фактов на разных уровнях агрегации, денормализация таблиц измерений, а также разбиение и репликация таблиц,

Реализация информационной системы в любой компании всегда является причиной возникновения различного рода конфликтов в организационной и поведенческой сфере. Поскольку хранилище данных предназначено для моделирования всех совокупных данных предприятия — а многие его подразделения, возможно, имеют противоположные цели — возможно возникновение противоречивости и избыточности данных. Поэтому для минимизации конфликтов при разработке хранилища данных необходим обширный ввод данных от конечных пользователей. Чтобы разрешить возникающие конфликты, руководители должны обладать навыками разрешения споров и конфликтных ситуаций, уметь выступать в качестве посредников и арбитров.

Добыча данных — новое поколение специализированных инструментов обеспечения поддержки решений, позволяющее автоматизировать анализ операционных данных с целью поиска ранее неизвестных характеристик данных, зависимостей, связей и/или тенденций. Процесс добычи данных состоит из четырех этапов: подготовка данных, анализ и классификация данных, приобретение знаний и прогноз. На этапе подготовки данных очищаются и классифицируются основные наборы данных, которые будут использоваться в операциях добычи данных. На этапе анализа и классификации определяются общие характеристики данных с помощью сложных алгоритмов, выявляющих группы данных, кластеры или последовательности, зависимости данных и/или связи, а также шаблоны данных, тенденции и отклонения. Результаты этапа анализа и классификации используются на этапе приобретения знаний для разработки компьютерных моделей, перестраивающих поведение наборов данных. Результаты этапа приобретения знаний используются на этапе прогноза для предсказания дальнейшего поведения и прогноза результатов деятельности предприятия.

Хранилище данных — основное место хранения информации для системы поддержки решений, которое интегрируется с двумя новыми компонентами систем поддержки решений: системами оперативного анализа данных (OLAP) и системами добычи данных (data mining). Средства OLAP-систем получают доступ к хранилищу данных для обеспечения расширенного многомерного анализа данных. Средства добычи данных используют информацию хранилищ для извлечения необходимого набора данных. Затем применяются алгоритмы добычи данных для получения специализированной информации или знаний о деятельности предприятия. Совместное использование этих компонентов составляет основу поддержки тактических и стратегических решений.

## Основные термины

База данных, размещенная в оперативной памяти — main memory database (MMDB)

Витрина данных — data mart

Гиперкуб — hypercube

Детальный анализ — drill down

Добыча данных — data mining

Иерархия атрибута — attribute hierarchy

Извлечение данных — data extraction

Измерения, качественные характеристики — dimension

Исследование "вдоль и поперек" — slice and dice

Куб данных — data cube

Кэш куба — cube cache

Метрика — metrics

Многомерная система оперативного анализа данных — multidimensional online analytical processing (MOLAP)

Организация хранилища данных — data warehousing

Очень большая база данных — Very Large Database (VLDB)

Периодичность — periodicity

Поддержка решений — decision support

Разбиение — partitioning

Разряженность — sparsity

Реляционные системы оперативного анализа — Relational Online Analytical Processing (ROLAP)

Репликация — replication

Система оперативного анализа данных — Online Analytical Processing (OLAP)

Система поддержки решений — Decision Support System (DSS)

Система управления многомерной базой данных (МСУБД) — Multidimensional Database Management System (MDBMS)

Склад данных — data store

Средства запроса конечного пользователя — end-user query tool

Средства представления данных конечного пользователя — end-user presentation tool

Схема "звезда" — star schema

Таблица измерений — dimension table

Таблица фактов — fact table

Укрупненный анализ — roll up

Факты — facts

Фильтрация данных — data filtering

Хранилище данных — data warehouse

## Вопросы

1. Что представляют собой системы поддержки решений? Какова их роль в инфраструктуре бизнеса?
2. Поясните взаимодействие основных компонентов системы DSS.
3. В чем основное различие между операционными данными и данными для систем поддержки решений?

4. Что такое хранилище данных, каковы его основные характеристики?
5. Приведите три примера проблем, которые, скорее всего, возникнут при интеграции операционных данных в хранилище данных.

Используйте следующий сценарий для ответа на вопросы с 6 по 12

Как аналитика национальной торговой организации вас попросили участвовать в группе разработки проекта хранилища данных.

6. Подготовьте укрупненный анализ основных требований для оценки СУБД для хранилища данных.
7. Группа разработки проекта хранилища данных обсуждает прототип хранилища данных до начала его реализации. Члены группы особенно беспокоятся о необходимости получения некоторого опыта, перед тем как реализовать проект в рамках всей организации. Что вы можете порекомендовать? Поясните свои рекомендации.
8. Предположим, вы рекламируете идею хранилища данных вашим пользователям. Как вы объясните им, что такое многомерный анализ данных и в чем его преимущества?
9. Группа разработки проекта хранилища данных пригласила вас, чтобы представить обзор OLAP-системы, перед тем как принять решение. Членов группы особенно интересуют требования к клиент/серверной архитектуре OLAP и то, каким образом OLAP-система вписывается в имеющуюся архитектуру. Ваша работа состоит в том, чтобы объяснить основные компоненты клиент/серверной OLAP-системы и ее архитектуру.
10. Один из ваших поставщиков рекомендует использовать МСУБД. Как вы объясните свой выбор руководителю проекта?
11. Группа проекта готова сделать окончательный выбор между ROLAP и MOLAP. Что должно лежать в основе такого решения? Почему?
12. Проект хранилища данных находится в стадии разработки. Поясните вашим проектировщикам, как можно использовать в проекте схему "звезда".
13. Проследите эволюцию DSS от начала до современных средств анализа данных. Какие технологии оказали влияние на эту эволюцию?
14. Что такое OLAP-система и каковы ее основные характеристики?
15. Поясните, что такое ROLAP-система и почему рекомендуется ее использовать в среде реляционных баз данных.
16. Поясните использование фактов, измерений и атрибутов в схеме "звезда".
17. Что такое многомерные кубы, и как в эту модель вписывается техника исследования "вдоль и поперек" (slice and dice)?
18. Что такое иерархии атрибутов и уровни агрегации в контексте схемы "звезда" и каково их предназначение?
19. Обсудите наиболее общие технологии повышения эффективности обработки данных, применяемые в схеме "звезда"
20. Поясните некоторые из наиболее важных проблем, возникающих при реализации хранилища данных.

21. Что такое добыча данных и чем она отличается от традиционных средств DSS?
22. Как работает добыча данных? Опишите этапы процесса добычи данных.

## Задачи

### **Примечание для преподавателя<sup>3</sup>**

Базы данных для хранилища данных в задачах 1, 3 и 4 в онлайн-руководствах (Instructor's Manual) представлены в формате Microsoft Access 2000. Базы данных с именами DW-P1.MDB, DW-P3.MDB и DW-P4.MDB содержат информацию для задач 1, 3 и 4 соответственно. Поскольку размеры некоторых из этих баз данных очень велики, чтобы ваши студенты имели необходимую рабочую базу данных для заполнения хранилища данных в каждой из задач списка, мы предлагаем вам скопировать эти базы данных и сделать их доступными для студентов. Полное решение содержат файлы DW-P1sol.MDB, DW-P3sol.MDB и DW-P4sol.MDB.

Данные для задачи 2 хранятся в формате Microsoft Excel 2000, в файле DW-P2.xls (решение — в файле DW-P2sol.xls).

1. Директор Университетской компьютерной лаборатории отслеживает использование лаборатории, учитывая число студентов, пользующихся лабораторией. Эта функция имеет очень большое значение при формировании бюджета. Директор поручил вам задачу разработки хранилища данных, с помощью которого можно отслеживать статистику загрузки лаборатории. Основные требования к этой базе данных:
  - показать общее число пользователей по различным периодам времени;
  - показать загрузку по данному промежутку времени, по дисциплинам и по классификации студентов;
  - сравнивать загрузку по различным дисциплинам и различным семестрам.

Используйте базу данных DW-P1.MDB, в которую входят следующие таблицы:

- USELOG — содержит данные по доступу студентов в лабораторию;
- STUDENT — таблица измерений, содержащая данные по студентам.

С учетом требований (приведенных ниже) и используя данные DW-P1.MDB решите задачи 1a—1ж.

- а. Определите основные факты, которые необходимо анализировать (*Подсказка*: эти факты — источник проектирования таблицы фактов.)
- б. Определите и опишите необходимые измерения (*Подсказка*: эти измерения будут использоваться при проектировании таблиц измерений.)
- в. Изобразите схему "звезда" для загрузки лаборатории, используя структуры фактов и измерений, определенные в задачах 1а и 1б.
- г. Определите атрибуты для каждого из измерений задачи 1б.
- д. Дайте рекомендации по необходимым иерархиям атрибутов.

---

<sup>3</sup> Для доступа к дополнительным материалам для преподавателей нужно пройти авторизацию на сайте [www.course.com](http://www.course.com). — *Прим. ред.*

- е. Реализуйте проект хранилища данных с помощью схемы "звезда", созданной в задаче 1в, и атрибутов, определенных в задаче 1г.
- ж. Создайте отчеты, удовлетворяющие требованиям, перечисленным во введении к этой задаче.

2. Миссис Виктория Эфанор руководит небольшой компанией, поставляющей товары. Поскольку бизнес быстро развивается, миссис Эфанор решила, что пора организовать управление большими объемами информации, чтобы контролировать растущий бизнес. На предприятии миссис Эфанор, которая знакома с электронными таблицами, в настоящее время работают 4 продавца. Она попросила вас разработать прототип приложения хранилища данных, позволяющего изучить объемы продаж по годам, регионам, продавцам и товарам. (Этот прототип будет использоваться в качестве основы для занесения данных в хранилище.)

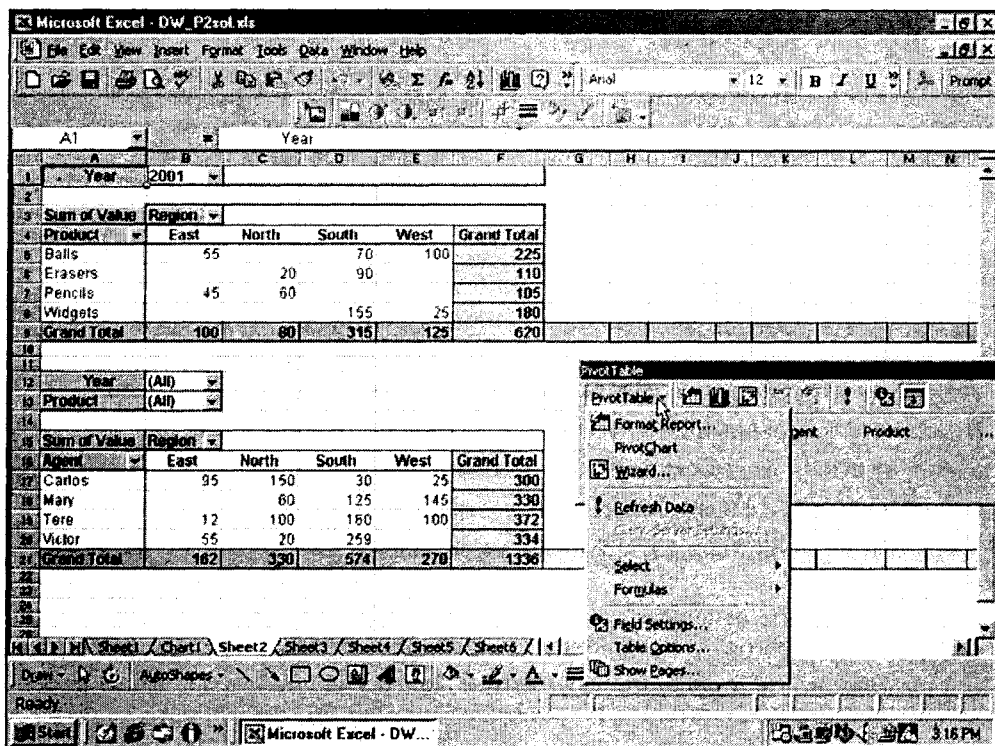


Рис. 13.24. Использование сводной таблицы

Используя данные, представленные в файле DW-P2.xls, решите следующие семь задач.

- а. Определите компоненты таблицы фактов.
- б. Определите таблицу измерений.

- в. Нарисуйте схему "звезда" для этого хранилища данных.
- г. Определите атрибуты для таблиц измерений, необходимые для решения этой задачи.
- д. Используя электронные таблицы Microsoft Excel (или любую систему, позволяющую создавать сводные таблицы), создайте сводную таблицу, отражающую объемы продаж товара по регионам. Конечный пользователь должен иметь возможность определять отображение объема продаж для каждого данного года. (Пример вывода представлен в первой сводной таблице на рис. 13.24.)
- е. Используя задачу 2д, добавьте вторую сводную таблицу (см. рис. 13.24), чтобы представить объемы продаж по продавцам и по регионам. Конечный пользователь должен иметь возможность определить продажи по данному году или по всем годам и для данного товара, или для всех товаров.
- ж. Создайте трехмерную диаграмму, чтобы показать объемы продаж по продавцам, по товарам и по регионам (пример вывода — на рис. 13.25).

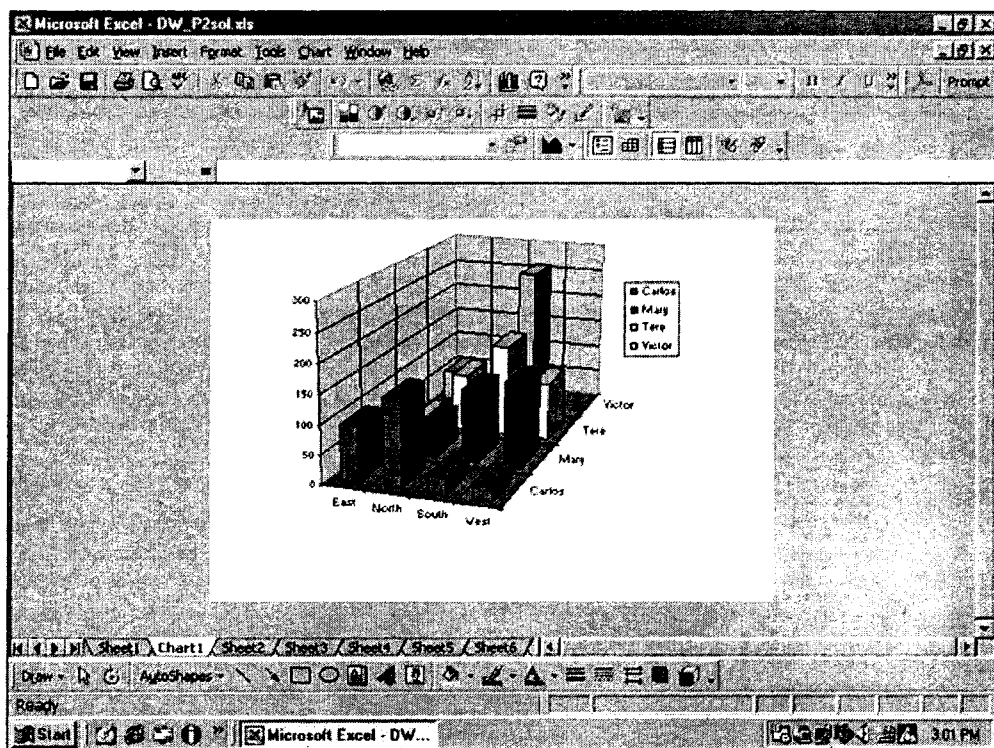


Рис. 13.25. Трехмерная диаграмма, показывающая связи между агентами, товарами и регионами

3. Мистер Дэвид Сакер, руководитель службы материально-технического снабжения в компании, занимающейся исследованием рынка, заинтересован в изучении использования материальных средств в различных отделах компании. Мистер Сакер слышал, что его подруга, миссис Эфанор, разработала небольшое хранилище данных на основе электронных таблиц (см. задачу 2), которое она использует для анализа данных объема продаж. Мистер Сакер заинтересован в разработке небольшой модели хранилища данных, похожей на имеющуюся у миссис Эфанор, для того, чтобы проанализировать заказы по подразделениям и по товарам. Он использует Microsoft Access в качестве СУБД хранилища данных и Microsoft Excel в качестве инструмента анализа.

- Разработайте схему "звезда".
- Определите необходимые атрибуты измерений.
- Определите иерархии атрибутов, необходимые для этой модели.
- Выполните отчет в табличной форме (в Microsoft Access), используя трехмерную диаграмму, чтобы показать заказы по товарам и по подразделениям (пример вывода представлен на рис. 13.26).

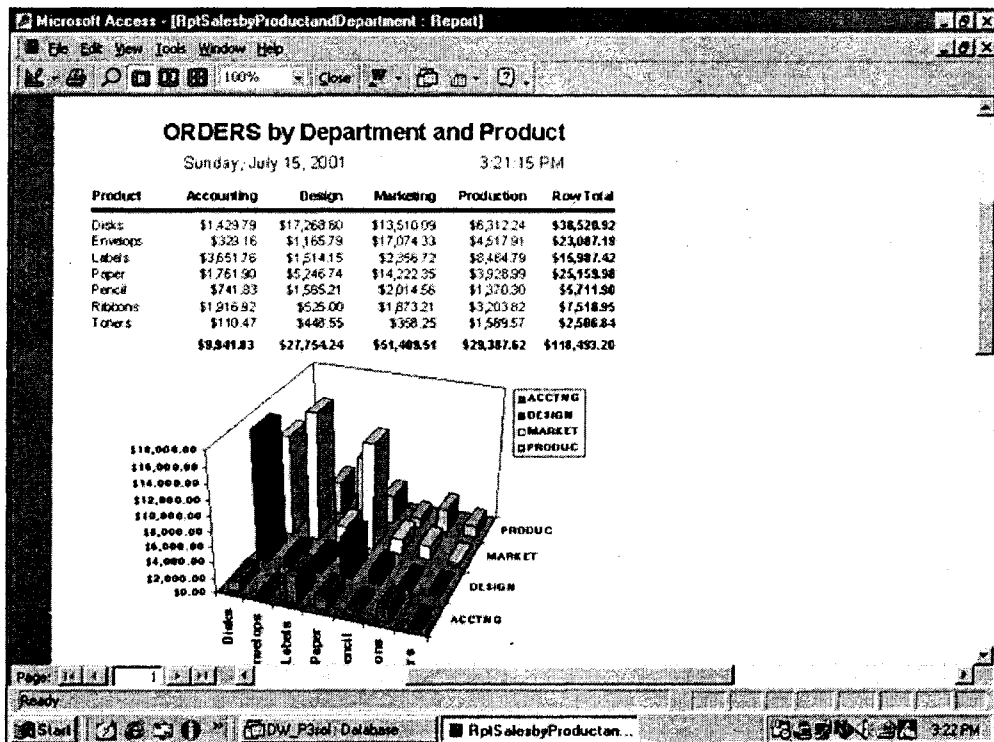
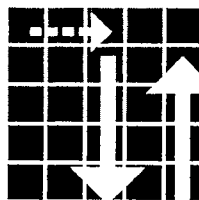


Рис. 13.26. Табличный отчет: заказы по подразделениям и по товарам



4. Корпорация ROBCOR, образцы данных для которой находятся в базе данных (файл DW-P4.MDB), предоставляет чартерные рейсы по требованию, используя различные типы самолетов. Поскольку корпорация ROBCOR быстро развивается, вас пригласили на работу в качестве администратора базы данных (база данных компании, разработанная сторонней консалтинговой организацией, уже включает в себя базу данных рейсов, помогающую управлять операциями компании). Ваше первое и самое важное действие состоит в разработке системы поддержки решений для анализа данных по чартерам (вспомните, пожалуйста, задачи 28—30 гл. 2, в которых были описаны операции этой компании). Руководителю полетов необходимо иметь возможность анализировать данные чартерных рейсов (стоимость, полетное время, расход топлива и выручку). Ему также требуется возможность углубленного изучения данных по пилотам, типу самолета и временным диапазонам.
5. С учетом этих требований выполните следующие задачи.
- а. Создайте схему "звезда" для данных по рейсам.
  - б. Определите измерения и атрибуты для схемы "звезда" чартерных операций.
  - в. Определите необходимые иерархии атрибутов.
  - г. Реализуйте проект хранилища данных, используя компоненты проекта, разработанные в задачах 4а—4в.
  - д. Выполните отчет, показывающий, что данные вашего хранилища могут соответствовать заданным информационным требованиям.

## Глава 14



# Базы данных в электронной коммерции

В этой главе мы изучим:

- ☐ что такое электронная коммерция (e-commerce, e-коммерция);
- ☐ влияние e-коммерции на современную экономику;
- ☐ различные стили e-коммерции;
- ☐ архитектурные компоненты Интернета, необходимые для ведения e-коммерции;
- ☐ проблемы проектирования и реализации баз данных для приложений e-коммерции;
- ☐ что такое расширяемый язык разметки (eXtensible Markup Language, XML) и каково его предназначение.

## Обзор

### ***Дополнительная информация***

Дополнительная информация по материалам этой главы имеется на сайте [www.course.com](http://www.course.com) (искать по ISBN данной книги 0-619-06269-X).

Если вы живете не на необитаемом острове, то, скорее всего, вы в той или иной степени соприкасались с Интернетом и глобальной Сетью World Wide Web. Интернет изменил деятельность организаций всех типов. Даже неискушенный человек может найти Web-адрес в рекламном проспекте практически любой организации. Покупать товары и пользоваться услугами с помощью Интернета стало привычным делом. Фактически компании всего мира используют Интернет так, как трудно было представить всего лишь несколько лет назад. Электронная коммерция (e-коммерция) позволяет предприятиям — независимо от того, являются они государственными или частными, — представлять свои товары и услуги на глобальном рынке для миллионов пользователей. Через Интернет — точнее, через Сеть — можно заказать авиабилет, номер в гостинице, купить акции, билеты в театр или на концерт, проверить состояние своего банковского счета и выполнить любое исследование. Например, заказывая книгу через [amazon.com](http://amazon.com) (один из самых популярных интернет-магазинов в США), вы как раз и участвовали в электронной коммерции.

Е-коммерция занимается не только продажей и покупкой товаров через Интернет: Интернет оказывает большое влияние на обработку информации в правительственных и некоммерческих организациях. Через Интернет можно обновить водительские права, заплатить налоги и записаться на курсы. С помощью Интернета адвокаты помогают миллионам людей решить проблемы. Вы планируете приобрести или продать дом? С помощью Интернета вы можете найти тысячи предложений с подробными планами жилищ, отражающих различные архитектурные стили. Заказать дом именно той планировки, о которой вы мечтали, можно щелкнув несколько раз кнопкой мыши. Вы ходите поменять место жительства? Легко отыскать сайты, которые предоставят вам всю необходимую информацию о том месте, куда вы хотите переехать, — от примерного уровня заработной платы до налоговых льгот, наличия школ, учреждений культуры и мест проведения досуга — чтобы облегчить вам переезд. Учитывая влияние Интернета на нашу сегодняшнюю жизнь, правительства практически всех стран мира уделяют очень много времени содействию е-коммерции и обеспечению гарантий того, что развитие этого рынка обеспечит всем равные права и возможности.

Компании, занимающиеся е-коммерцией, продают услуги и товары не только клиентам, но и другим компаниям. В Интернете появляются новые технологии, облегчающие обмен документами среди бизнес-партнеров. Компании используют Интернет для создания новых типов систем, интегрирующих информацию в целях повышения эффективности и уменьшения затрат.

### **Дополнительная информация**

На сайте [www.course.com](http://www.course.com) вы можете найти дополнительные сведения о влиянии Интернета на бизнес-операции, проектирование, реализацию и управление базами данных (ищите по ISBN 0-619-06269-X).

## **14.1. Что такое электронная коммерция?**

Е-коммерции (общепринятое сокращение термина "электронная коммерция") было дано много кратких определений. Онлайн-словарь *Webopedia* ([webopedia.com](http://webopedia.com)) определяет е-коммерцию как "ведение интерактивного бизнеса" (*conducting business on-line*). Однако даже беглое изучение публикаций со всей очевидностью показывает, что определение е-коммерции меняется в зависимости от того, кого вы об этом спрашиваете, — и, судя по всему, эти определения меняются с той же скоростью, что и современные технологии. В этой книге *электронная коммерция — е-коммерция* — определяется как использование электронной компьютерной технологии в следующих целях:

- ☐ поставка на рынок новых товаров, услуг или идей;
- ☐ поддержка и расширение спектра бизнес-операций (включая продажу товаров/услуг через Web).

Хотя предприятия всегда использовали различные виды электронной технологии, е-коммерция, как правило, идентифицируется с использованием Интернета в качестве средства обеспечения бизнеса (покупка, продажа товаров и услуг) и придания значимости предприятию. Поскольку Интернет, в особенности Web (глобальная

Сеть), играет такую важную роль в становлении и развитии е-коммерции, некоторые специалисты полагают, что е-коммерцию нужно называть *интернет-коммерцией (I-коммерцией)*.

Если рассматривать е-коммерцию только как расширение возможностей клиента, то полезно вспомнить, что большинство транзакций е-коммерции происходит все же между предприятиями. Компании используют Интернет для рационализации своей деятельности и для расширения внешних и внутренних операций. На самом деле, одно из главных достоинств е-коммерции — повышение конкурентоспособности предприятий. Поскольку операции е-коммерции так хорошо встраиваются в среду бизнеса, многие называют е-коммерцию *электронным бизнесом (е-бизнесом)*.

Внешний признак участия компании в е-коммерции — наличие Web-сайта. Web-сайт может быть простым, состоящим из нескольких статичных страниц, предоставляющих линейку товаров и контактную информацию, или сложным, с полным оперативным, основанным на базе данных, каталогом товаров с возможностью оперативного заказа и оплаты с помощью кредитных карт. В следующих разделах будут представлены различные стили е-коммерции и их влияние на различные секторы бизнеса.

Е-коммерция сейчас считается самым простым источником дохода. Если разместить ваш товар на рынке Сети, он будет доступен миллионам покупателей. Компании конкурируют за ресурсы сетевого рынка, привлекая клиентов на свои Web-сайты (и стараясь удержать их там!). Но сделать это не так-то просто — многие компании обнаруживают, что конкуренция на сетевом рынке очень напряженная и не ограничивается лишь созданием и использованием Web-страниц. То есть е-коммерция это нечто большее, чем просто еще один канал сбыта продукции; е-коммерция представляет собой ядро новой модели бизнеса, цель которого — быстрое представление товаров, идей и услуг на огромный рынок относительно недорогим способом с помощью интернет-технологий.

Е-коммерция не является самоцелью. Скорее, это путь к успешной конкуренции и выживанию бизнеса в XXI столетии. На этом пути интернет-технология сыграла и продолжает играть важнейшую роль. Чтобы знать, куда мы движемся, полезно знать, где мы находимся сейчас. Поэтому следующий раздел посвящен беглому обзору основных вех на пути развития е-коммерции.

## 14.2. Путь к электронной коммерции

Понимание и использование современных технологий — ключ к выживанию в условиях жесткой конкуренции. Бизнес использует технологии для расширения своей деятельности в течение многих десятилетий. Например, телефоны и факсимильные аппараты — пример технологий, уже давно используемых в бизнесе. Должны ли эти технологии рассматриваться как составная часть модели е-коммерции? Конечно, нет (мы не имеем в виду телефон, подключенный к модему компьютера)! В приведенных примерах проведение коммерческих сделок сильно зависит от участия человека. Фактически телефоны и факсимильные аппараты выполняют простые функции связи для отправителя и получателя информации. Получатель информации для завершения операции должен обработать данные вручную. Ключевым компонентом е-коммерции является использование Интернета.

### **Дополнительная информация**

На сайте [www.course.com](http://www.course.com) вы можете найти краткий исторический экскурс возникновения е-коммерции (ищите по ISBN 0-619-06269-X).

## **14.3. Влияние е-коммерции**

Е-коммерция изменила среду бизнеса в таком масштабе, который еще несколько лет назад невозможно было даже представить. Эти изменения на рынке, обусловленные компьютерными технологиями и Интернетом, экономисты называют "новой экономикой".

### **Дополнительная информация**

Об изменениях в современном бизнесе и о влиянии, которое они оказывают на проведение коммерческих сделок, можно почитать на сайте [www.course.com](http://www.course.com) (поиск по ISBN 0-619-06269-X).

Для отдела информационных систем применение интернет-технологий — новая возможность предоставления услуг клиентам, партнерам, сотрудникам и широким слоям населения. Интернет — это основа разработки нового поколения информационных систем. Для многих информационных приложений применение интернет-технологий облегчает совместное использование разнородных данных в среде, предоставляющей значительные выгоды посредством распределения затрат.

Именно в этой динамической среде созревает "новая экономика". Итак, каким же образом интернет-технологии (в особенности Web) влияют на традиционный рынок? Как е-коммерция влияет (положительно или отрицательно) на покупателей и продавцов? Поскольку е-коммерция основана на непрерывно меняющейся технологической инфраструктуре, мы не можем дать ответить определенно. Ответы на эти вопросы зависят от природы индустрии, вида рынка и его масштабов, структуры компании и т. д. Однако, по крайней мере, некоторые тенденции начинают проявляться. В следующих разделах главы мы исследуем некоторые преимущества и недостатки, которые е-коммерция имеет в глазах как покупателей, так и продавцов.

### **14.3.1. Преимущества е-коммерции**

К преимуществам е-коммерции относится возможность быстрого и удобного сравнительного поиска товара, что усиливает конкуренцию и тем самым снижает стоимость товара. Кроме того, возможность проведения множества коммерческих сделок на автоматизированной основе означает, что бизнес может функционировать 24 часа в сутки, 7 дней в неделю и 365 дней в году (иногда это называют  $24 \times 7 \times 365$ ). Глобальный доступ подразумевает потенциальное расширение рынка и снятие барьеров, препятствующих доступу к глобальному рынку. Наконец, среда е-коммерции стимулирует изучение рынка клиентами, что позволяет им принимать верные решения.

### **Дополнительная информация**

Используйте материал сайта [www.course.com](http://www.course.com) для получения подробных сведений о преимуществах е-коммерции (поиск по 0-619-06269-X).

### 14.3.2. Недостатки е-коммерции

Несмотря на столь впечатляющие достоинства е-коммерции, она далека от совершенства. Фактически некоторые из недостатков вызывают весьма сильное беспокойство, как со стороны клиентов, так и со стороны бизнесменов. Хотя обычно все говорят о значительной экономии е-коммерции, скрытые затраты быстро превратят это преимущество в недостаток. Кроме того, эта технология несовершенна: надежность сетей продолжает оставаться слабым звеном. А небольшой (становящийся все более незаметным) задел, который имеет е-коммерция, означает, что за место в этом бизнесе предстоит напряженная борьба. Может быть, самые большие проблемы кроются в сфере безопасности, утере секретности, низком уровне удаленного сервиса и в сложных правовых проблемах.

#### **Дополнительная информация**

Более подробные сведения о недостатках е-коммерции вы можете найти на сайте [www.course.com](http://www.course.com) (поиск по ISBN 0-619-06269-X).

### 14.4. Стили е-коммерции

Компании, которые осуществляют свои сделки через Интернет, могут использовать различные схемы реализации. Компании, использующие е-коммерцию, можно классифицировать следующим образом.

- ❑ *Чистые компании е-коммерции.* Транзакции таких компаний преследуют только одну цель: представить свои товары и услуги в Интернете. Обычно такая компания создается специально для онлайн-бизнеса. Поэтому вместо розничных точек продажи у них имеются "виртуальные витрины" в Интернете. Для таких компаний весь бизнес и архитектура транзакций по предоставлению товаров и услуг строятся на основе Интернета.
- ❑ *Компании традиционного бизнеса, которые решили расширить свою деятельность за счет онлайн-рынка.* В этом случае компании уже имеют устоявшийся рынок и определенный круг клиентуры, Интернет для них это просто еще один канал сбыта. Такие компании, как правило, имеют информационные системы с определенной архитектурой, которые необходимо расширить до интеграции с е-коммерцией.

Транзакции е-коммерции можно классифицировать по покупателям и продавцам. Выделим следующие основные стили е-коммерции.

- ❑ *Бизнес-бизнес (business-to-business, B2B):* электронная коммерция между коммерческими предприятиями.
- ❑ *Бизнес-клиент (business-to-consumer, B2C):* электронная коммерция между предприятием и клиентом.
- ❑ *Внутренний бизнес (Intra-business):* внутренние коммерческие операции, большая часть из которых — взаимоотношения между сотрудниками и нанимателями.

Хотя некоторые могут возразить, что сюда еще необходимо включить такие стили е-коммерции, как правительство-бизнес (government-to-business, G2B) и правитель-

ство-клиент (government-to-consumer, G2C), мы все же склонны рассматривать их как специальные случаи B2B и B2C. На рис. 14.1 представлены основные участники е-коммерции и их взаимодействие.

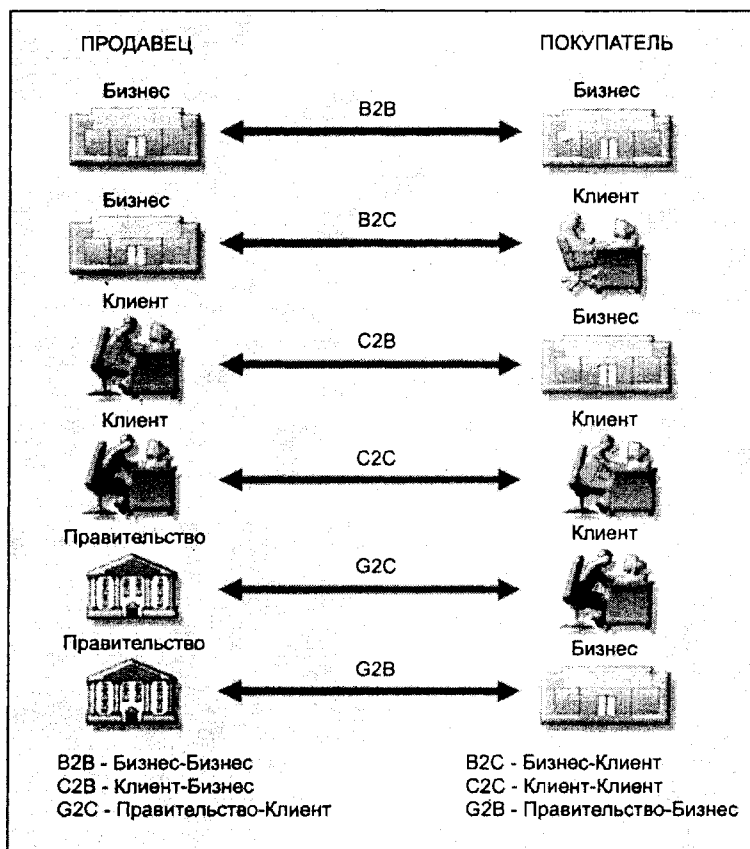


Рис. 14.1. Стили е-коммерции

### Дополнительная информация

Дополнительные сведения о различных стилях е-коммерции можно найти на сайте [www.course.com](http://www.course.com). Здесь приведено много разных примеров стилей е-коммерции (поиск по 0-619-06269-X).

## 14.5. Архитектура е-коммерции

У компаний е-коммерции возникают как организационные, так и технические проблемы. Организационные проблемы связаны как с установлением партнерских от-

ношений с поставщиками, дистрибьюторами и производителями, так и с проектированием и разработкой хорошо отлаженных бизнес-планов.

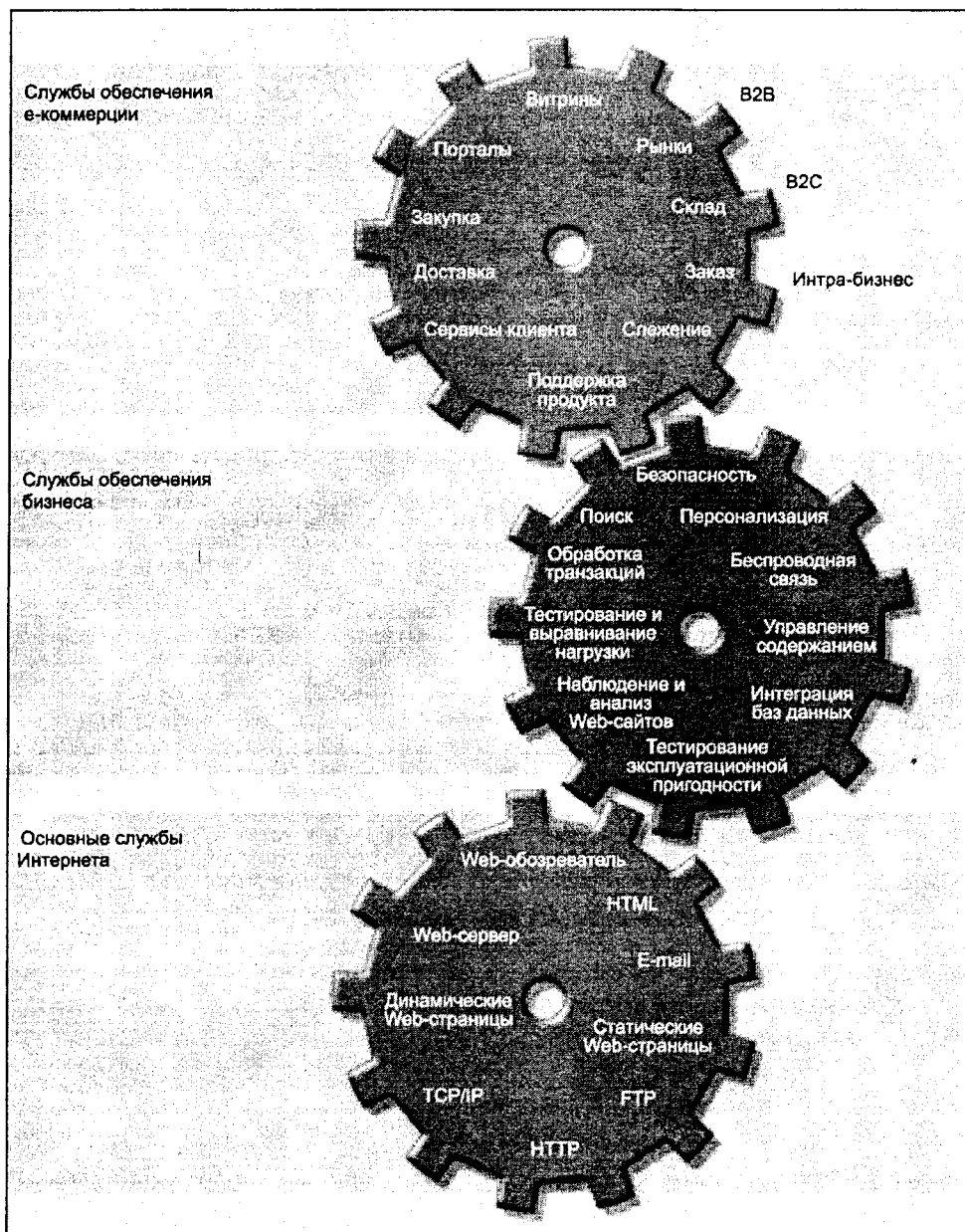


Рис. 14.2. Архитектура e-коммерции



Технические проблемы связаны с оборудованием и программным обеспечением, которые обеспечивают надежность транзакций е-коммерции. Независимо от способа и степени участия информационной структуры предприятия в е-коммерции, есть потребность в надлежащем проектировании, разработке и развертывании хорошо спланированной архитектуры, обеспечивающей как внутренние, так и внешние бизнес-транзакции. Решение этих проблем зависит от состояния архитектуры информационной системы и от стиля е-коммерции.

В этом разделе будут представлены основные архитектурные компоненты, необходимые для обеспечения транзакций е-коммерции<sup>1</sup>.

Для лучшего понимания архитектуры е-коммерции разделим ее на несколько уровней. Каждый уровень использует сервисы от нижерасположенных уровней. Ниже приводится список этих уровней.

- ☐ Основные службы Интернета.
- ☐ Службы обеспечения бизнеса.
- ☐ Службы обеспечения е-коммерции.

На рис. 14.2 представлен общий вид архитектуры е-коммерции.

## 14.5.1. Основные службы Интернета

Е-коммерция ведется в Интернете. Интернет предоставляет основные службы (сервисы), которые облегчают передачу информации между компьютерами. Термины "Интернет" и "World Wide Web" (Сеть) часто используются как взаимозаменяемые, но это не синонимы. World Wide Web — просто одна из многих служб Интернета. В табл. 14.1 описаны основные функциональные блоки и службы Интернета.

**Таблица 14.1. Функциональные элементы и основные службы Интернета**

| Функциональный блок | Определение                                                                                                                                                                                                                                                                                                                                                            |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Интернет            | Всемирная Сеть сетей — Интернет — действует как "суперсеть", связывающая воедино тысячи небольших сетей, разбросанных по всему миру. Интернет можно представлять себе как скоростную магистраль передачи данных, образно говоря, "супермагистраль информации"                                                                                                          |
| TCP/IP              | Основной сетевой протокол, определяющий правила, используемые при создании и маршрутизации пакетов данных между компьютерами в одной и той же сети или в разных сетях. Каждый компьютер, подключенный к Интернету, имеет уникальный TCP/IP-адрес (или IP-адрес). IP-адрес состоит из двух частей, которые используются для идентификации сети и компьютера (или хоста) |

<sup>1</sup> Хотя организационные проблемы очень важны для успешного ведения е-коммерции, они остаются за рамками рассмотрения этой книги. Здесь мы сосредоточимся на интернет-технологии.

Таблица 14.1 (продолжение)

| Функциональный блок                                                           | Определение                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Маршрутизатор</b>                                                          | Специальное аппаратно-программное оборудование, с помощью которого соединяются разнородные сети. Маршрутизатор отвечает за доставку пакетов данных от локальной сети в удаленную сеть. Маршрутизаторы, образно говоря, являются надсмотрщиками Интернета, наблюдающими за трафиком и передачей данных от одной сети к другой                                                                                                                   |
| <b>World Wide Web (WWW или Web, глобальная Сеть)</b>                          | Всемирная сетевая совокупность специальным образом форматированных и взаимосвязанных документов называется Web-страницами. Web — это только одна из услуг, предоставляемых Интернетом                                                                                                                                                                                                                                                          |
| <b>Web-страница</b>                                                           | Текстовый документ, содержащий текст и специальные команды (теги) на языке HTML (Hypertext Markup Language, язык разметки гипертекста). Web-страница может содержать текст, графику, видео- и аудиоинформацию и т. д.                                                                                                                                                                                                                          |
| <b>Язык разметки гипертекста, HTML (Hypertext Markup Language)</b>            | Стандартный язык разметки документов для Web-страниц. HTML помогает создавать связи между документами и представлять эти документы в Web-обозревателе стандартным способом                                                                                                                                                                                                                                                                     |
| <b>Гиперссылка (hyperlink)</b>                                                | Web-страницы связываются друг с другом (т. е. каждая Web-страница вызывает другие Web-страницы), создавая эффект паутины (web). Поскольку ссылка может связывать различные типы документов, например, текст, графику, анимированную графику, видео и аудио, она называется "гиперссылкой". В HTML-коде Web-страница-гиперссылка часто выражается с помощью URL (Universal Resource Locator, унифицированный указатель информационного ресурса) |
| <b>Унифицированный указатель информационного ресурса (URL), или Web-адрес</b> | URL уникально идентифицирует адрес ресурса в Интернете. Примеры URL: <b>www.dell.com</b> , <b>www.ford.com</b> , <b>www.amazon.com</b> , <b>www.bhv.ru</b> , <b>www.faa.gov</b> и <b>www.mtsu.edu</b>                                                                                                                                                                                                                                          |
| <b>Протокол передачи гипертекста (Hypertext Transfer Protocol, HTTP)</b>      | Стандартный протокол, используемый Web-обозревателем и Web-сервером для коммуникации, т. е. для обмена запросами и ответами между серверами и обозревателями. В HTTP для передачи данных между компьютерами, подключенными к Интернету, используется протокол TCP/IP                                                                                                                                                                           |
| <b>Web-обозреватель</b>                                                       | Приложение конечного пользователя, использующееся для просмотра Web-страниц в Интернете и для навигации по страницам. Обозреватель представляет собой графическое приложение, которое выполняется на клиентском компьютере, и его основной функцией является отображение Web-страниц. Клиент использует Web-обозреватель (например, Netscape Navigator, Microsoft Internet Explorer, Opera и т. д.) для запроса Web-страниц с Web-сервера      |

Таблица 14.1 (окончание)

| Функциональный блок                                           | Определение                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Web-сервер</b>                                             | Специализированное приложение, единственной функцией которого является ожидание запроса клиентов, их обработка и передача запрошенных Web-страниц на клиентский обозреватель. Web-сервер и Web-клиент связываются с помощью специального протокола — HTTP                                                                                                                                      |
| <b>Web-сайт</b>                                               | Этот термин используется для обозначения Web-сервера и набора Web-страниц, хранящихся на локальном жестком диске серверного компьютера или в доступном каталоге совместного использования                                                                                                                                                                                                      |
| <b>Статичная Web-страница</b>                                 | Web-страница, содержимое которой остается одним и тем же (при просмотре в обозревателе), пока она не будет отредактирована вручную. Примером статичной Web-страницы является стандартный прейскурант производителя товаров, предоставляемый покупателям                                                                                                                                        |
| <b>Динамичная Web-страница</b>                                | Web-страница, содержимое которой автоматически создается и оформляется каждый раз при запросе в соответствии с потребностями пользователя. Например, конечный пользователь может получить доступ к Web-странице, на которой отображена стоимость акций избранной компании                                                                                                                      |
| <b>Протокол передачи файлов (File Transfer Protocol, FTP)</b> | Используется для передачи файлов между компьютерами в Интернете. FTP-клиент запрашивает файл от FTP-сервера. FTP-сервер ожидает запросы клиента, обрабатывает их и отправляет запрошенные файлы клиенту                                                                                                                                                                                        |
| <b>Электронная почта (e-mail)</b>                             | Используется для передачи сообщений электронной почты между компьютерами в Интернете. Почтовый сервер хранит почтовые сообщения в почтовом ящике пользователя (mailbox). Почтовые клиенты извлекают сообщения из почтового сервера. Когда клиент посылает электронную почту (e-mail), она временно хранится на почтовом сервере, который в свою очередь доставляет почту по корректному адресу |
| <b>Службы новостей и дискуссионных групп по интересам</b>     | Специализированные сервисы (службы), позволяющие создавать "виртуальные объединения", в которых пользователи обмениваются сообщениями по специальным темам, например, авиация, спорт, компьютеры. Этот сервис позволяет конечным пользователям размещать информацию на электронных досках объявлений для публичного доступа к ней                                                              |

Именно глобальная Сеть (World Wide Web) реально открыла Интернет миру и позволила разработать новую экономику на базе e-коммерции. Каждый из функциональных блоков, приведенных в табл. 14.1, использовался вышестоящими уровнями для обеспечения поддержки транзакций e-коммерции. На рис. 14.3 показаны связи компонентов, описанных в табл. 14.1.

Обратите внимание на рис. 14.3, что клиент вводит Web-адрес или URL в Web-обозревателе. В свою очередь, Web-обозреватель отправляет запрос Web-страницы на Web-сервер. Этот запрос обрабатывается TCP/IP для передачи по сети. Протокол

TCP/IP преобразует запрос в пакеты и отправляет их на маршрутизатор, который в свою очередь отправляет его в Интернет. В Интернете TCP/IP-пакеты проходят несколько маршрутизаторов до достижения целевого сервера. Сервер получает запрос на Web-страницу, выбирает страницу и отправляет ее на клиентский Web-обозреватель тем же способом. Клиент получает Web-страницу в формате HTML и отображает ее на экране.

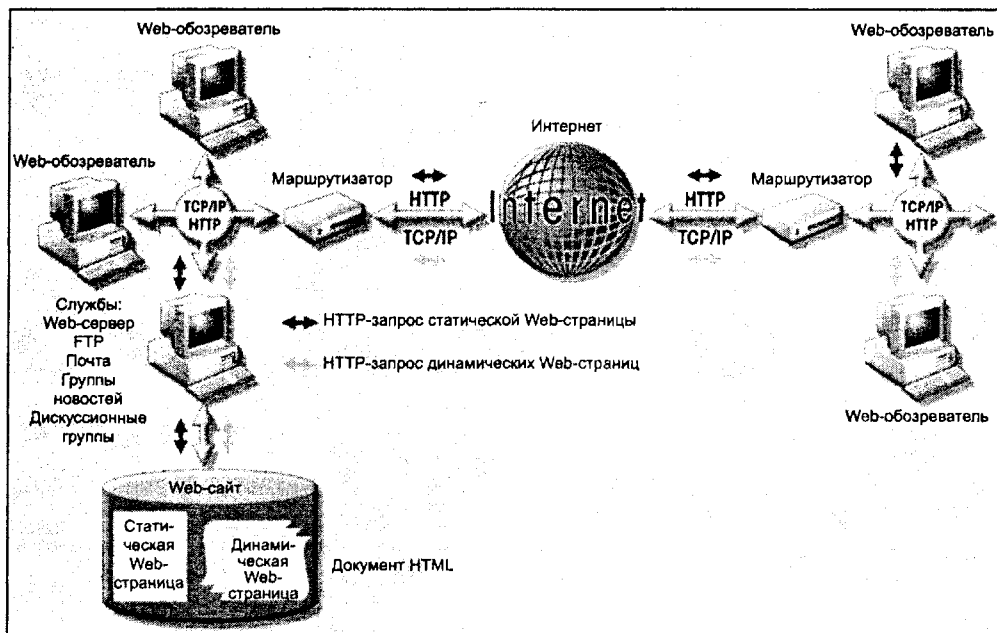


Рис. 14.3. Основные службы Интернета

Клиент и сервер могут быть расположены в одном здании или где угодно на планете. Это делает Web Великим Посредником по доставке информации через географические границы. Web-обозреватель интегрирует все службы, предоставляемые Web-сайтом. Конечному пользователю нет необходимости знать обо всех приложениях, выполняющихся на стороне сервера.

Web-страницы могут быть статичными и динамичными. *Статичные Web-страницы* используются для отображения информации, которая не изменяется в течение некоторого промежутка времени или вообще не меняется со временем. *Динамичные Web-страницы* используются в основном для страниц с содержимым, которое меняется со временем и его нельзя заранее предсказать, примером может служить онлайн-система заказов. Статичные Web-страницы адекватно отображают такую информацию, как каталоги товаров или контактные сведения; динамичные Web-страницы лучше приспособлены для приложений e-коммерции, например, для онлайн-системы заказов с возможностью выбора товара пользователем. Например, вы можете посетить сайт компании Dell, посвященный домашним компьютерам ([www.dell4me.com](http://www.dell4me.com)), и здесь

вам помогут сконфигурировать ваш компьютер. Динамические Web-страницы — основа большинства Web-транзакций B2B и B2C.

Интернет-технологии также облегчают создание сетей с управляемым доступом. Такие сети позволяют компаниям контролировать, какие компьютеры (внутри или вне организации) получают доступ к сети. С точки зрения технологии, *интранет* — это принадлежащий компании и управляемый компанией Интернет, доступ к которому может осуществляться только с компьютеров локальной сети компании. В интранет-сетях используются интернет-технологии для разработки решений e-коммерции и для улучшения деятельности компании, облегчая создание, сбор и распределение бизнес-информации, используемой для принятия решений на всех уровнях предприятия (e-коммерция интранет-бизнеса).

Если интранет распространяется за пределы одной корпорации, то он называется экстранетом. *Экстранет* расширяет интранет в *value chain*<sup>2</sup> предприятия. Экстранет облегчает интеграцию B2B. Например, большое предприятие розничной торговли может использовать экстранет для расширения информации о продукции и заказах для своих поставщиков и дистрибьюторов, помогая таким образом автоматизировать их деятельность. Существование экстранета расширяет возможности внутриорганизационного технологического процесса.

### 14.5.2. Службы обеспечения бизнеса

Служб Интернета, описанных в предыдущем разделе, достаточно для работы основного Web-сайта. Однако они не предоставляют средств, необходимых для проведения даже элементарных транзакций. Службы обеспечения бизнеса предлагают дополнительную поддержку коммерческих сделок. Эти добавочные службы реализуются аппаратными и программными компонентами, работающими совместно, обеспечивая дополнительные функциональные возможности, которые не могут быть предоставлены основными службами Интернета. В табл. 14.2 приведены службы, которые используются для расширения Web-сайтов, предоставляя возможность выполнения поиска, обеспечения аутентификации и безопасности деловой информации, управления содержимым Web-сайта и т. д. Список, представленный в табл. 14.2, нельзя считать полным: технологии продолжают развиваться, возникают новые службы, которые в свою очередь используются для создания новых служб.

Таблица 14.2. Службы обеспечения бизнеса

| Служба           | Определение                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Поисковые службы | Поисковые службы позволяют Web-сайтам выполнять поиск по их содержимому. Эти службы могут использоваться на сайтах внутреннего бизнеса (Intra-business) для поиска информации о платежных ведомостях, премиях, отпусках, контрактной информации и т. д. Web-сайт B2C может использовать эту функцию для поиска информации о возврате продукции, данных о клиентах и т. д. Поисковые службы являются обязательными для всех Web-сайтов e-коммерции |

<sup>2</sup> Value chain — последовательность наращивания стоимости в ходе производственного процесса (сырье-материалы-полуфабрикаты/детали/узлы-готовое изделие). Причем производственный процесс рассматривается целиком, а не только в границах отдельного предприятия.

Таблица 14.2 (продолжение)

| Служба                                                   | Определение                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Безопасность</b>                                      | Эти службы разработаны для обеспечения безопасности и секретности информации посредством шифрования, цифровых сертификатов, SSL, S-HTTP, брандмауэров и прокси-серверов                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <b>Наблюдение за сайтом и анализ данных</b>              | Предоставляет службы для обеспечения оптимальной работы Web-сервера. Наблюдает за основными показателями производительности системы и сети. В эту службу также включены компоненты, позволяющие выявлять узкие места сети, сильно снижающие скорость загрузки Web-страниц. Средства анализа позволяют изучать сетевой трафик для выявления страниц, чаще всего посещаемых пользователями. Такая обратная связь позволяет оценить эффективность страниц                                                                                                                                                                                  |
| <b>Проверка и выравнивание нагрузки; Web-кэширование</b> | Проверка нагрузки выполняется перед тем, как запустить сайт e-коммерции в эксплуатацию. Основная цель этой операции — обеспечение работы многих тысяч пользователей, получающих доступ к сайту. Если предполагаемая нагрузка слишком велика для одного сервера, то, возможно, потребуется несколько серверов. Выравнивание нагрузки обеспечивает равномерное распределение нагрузки между несколькими серверами. Эти службы обеспечивают высокую производительность системы и оптимальную скорость работы                                                                                                                               |
| <b>Эксплуатационные испытания</b>                        | В среде e-коммерции плохой спроектированный Web-сайт может быть даже хуже, чем его отсутствие. Эксплуатационное тестирование связано с определением того, насколько возможности и сервисы Web-сайта дружелюбны пользователю. Легко ли обнаружить на Web-сайте средства поиска? Не затрудняют ли цвета Web-сайта чтение информации на экране портативных устройств? Оправданы ли все представленные на сайте опции?                                                                                                                                                                                                                      |
| <b>Персонализация</b>                                    | Средства персонализации связаны с настройкой Web-страниц на индивидуального пользователя. Это средство особенно необходимо в новостных и развлекательных Web-порталах. Идея персонализации — сделать сайт дружелюбным и привлечь к нему как можно больше пользователей. Чтобы увидеть персонализацию в действии, зайдите на Web-сайт yahoo.com и нажмите мышью кнопку <b>My Yahoo!</b> . После этого вы можете задать сферу ваших интересов, ваш ZIP-код и настроить Web-страницу Yahoo! на информирование вас о погоде именно в вашей местности, местных новостях и т. д.                                                              |
| <b>Web-разработки</b>                                    | Приложения e-коммерции требуют встраивания в Web-сайты бизнес-логики. Web-разработчикам предоставляются средства добавления бизнес-логики на Web-страницы. Язык HTML не является языком программирования — это просто спецификация форматирования документов, созданная для надлежащего представления документов в Web-обозревателях. Web-сайты можно использовать в коммерческих целях, добавляя бизнес-логику с помощью одной из многих средств Web-программирования, таких как Java, JavaScript, VBScript и т. д. Такие программные среды позволяют создавать динамические Web-страницы, составляющие основу коммерческих Web-сайтов |

Таблица 14.2 (окончание)

| Служба                                  | Определение                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Интеграция с базами данных</b>       | Данные коммерческих транзакций обычно хранятся в базе данных. Интеграция базы данных предприятия с Web — <i>необходимое условие</i> успеха е-коммерции. Многие СУБД, такие как Oracle и Microsoft SQL Server, поставляются с готовой средой разработки, интегрирующей базу данных и Интернет. Сторонние поставщики представляют решения, позволяющие интегрировать корпоративные базы данных (включая старые данные) с сайтом компаний. Более подробно эта тема будет обсуждаться в <i>гл. 15</i> |
| <b>Обработка транзакций</b>             | Е-коммерция создает глобальное хранилище с тысячами подключенных клиентов, каждый из которых работает как "виртуальный кассир", подключенный к вашей системе. Что случится, если некоторые из них отключатся до момента завершения транзакции? В <i>гл. 9</i> мы изучали необходимость управления транзакциями. В среде е-коммерции эта задача становится еще более значимой, поскольку транзакции могут порождаться клиентами по всему миру                                                      |
| <b>Управление содержанием</b>           | Web-сайт может содержать тысячи связанных друг с другом страниц. Управление содержанием автоматизирует создание и контроль наполнения Web-сайтов. Управление содержанием предоставляет стандартный гибкий способ создания Web-страниц различными работниками и отделами предприятия. Управление содержанием очень важно для компаний, занимающихся поставкой коммерческой информации                                                                                                              |
| <b>Обмен сообщениями</b>                | Интернет представляет собой канал взаимодействия приложений. Приложение е-коммерции посылает сообщения от клиента серверу и наоборот. Эти сообщения могут варьироваться в зависимости от реализации приложений. Обмен сообщениями предоставляет сервис, гарантирующий надлежащую маршрутизацию и доставку сообщений, ориентированных на приложения, между различными службами                                                                                                                     |
| <b>Поддержка беспроводных устройств</b> | В нашей экономике постоянно появляются новые способы коммерции. Беспроводные или мобильные устройства по всей вероятности будут играть важную роль в е-коммерции в будущем. Используя устройства мобильной коммуникации, такие как, например, интеллектуальные телефоны (smart phones), пользователи по всему миру смогут проводить коммерческие операции, бронировать номера в гостинице, покупать товары и оплачивать счета                                                                     |

### 14.5.3. Службы обеспечения е-коммерции

Службы обеспечения е-коммерции формируют верхний уровень ее архитектуры. На этом уровне используются службы двух более нижних уровней для отображения бизнес-логики и для автоматизации коммерческих процессов. Автоматизация бизнес-процессов расширяет возможности подразделений и операций внутреннего бизнеса. На этом уровне службы обеспечения бизнеса (уровень 2) взаимодействуют с основными службами Интернета (уровень 1) для обеспечения поддержки интер-

фейсных служб е-коммерции. Обратите внимание, что бизнес-функции управляют операциями с низлежащими уровнями, а не наоборот. На реализацию этого уровня требуется много времени, поскольку разработчику нужно полностью разобраться в бизнесе, установить партнерские взаимоотношения и выяснить поведение клиентов.

После разработки и реализации интерфейсного Web-приложения необходимо принять решение о том, *какие* службы необходимо обеспечить и *как* их обеспечить. Для сайтов, поддерживающих стиль B2C е-коммерции, интерфейсное приложение может быть таким же простым, как онлайн-каталог товаров, или таким же сложным, как ориентированная на базу данных виртуальная витрина с трехмерным представлением доступных товаров и поддержкой электронной корзины, позволяющей пользователям заказывать отдельные позиции в процессе просмотра товаров.

Общие службы, предоставляемые Web-сайтами, включая автоматизацию *supply chain* (процесс движения товаров от производителя к потребителю и реакция на запросы потребителей) для целей B2B. Такая автоматизация поддерживает онлайн-доставку, оперативный склад, онлайн-заказ, отслеживание заказов, доставку товаров, товарный спрос, управление пользовательским спросом и т. д.

Один из сервисов, "обязательных" для всех сайтов е-коммерции, — обработка платежей. Бывали ли вы на сайтах, где, пройдя все ступени оформления заказа, вы получали уведомление, что теперь вы должны позвонить по телефону для подтверждения заказа? Ясно, что это очень неудобно. В следующих двух разделах представлены два очень важных свойства е-коммерции, позволяющих выполнять онлайн-транзакции: безопасность и обработка платежей.

## 14.6. Безопасность

Компании ежегодно тратят миллионы долларов на программное и аппаратное обеспечение для защиты информации (включая персональную информацию о клиентах) и собственности от криминальных посягательств. В обычном (неэлектронном) бизнесе, например, в магазинах розничной продажи, в почтовом или в банковском отделении для защиты от воровства и вандализма имеются устройства обеспечения безопасности, такие, например, как камеры слежения, пуленепробиваемые стекла, контрольно-пропускные ворота, охранники. Точно так же Web-магазин требует проведения мероприятий по защите от электронных правонарушений. Несмотря на множество имеющихся аспектов обеспечения секретности и безопасности, мы рассмотрим только принципиальные механизмы, используемые при защите транзакций электронного бизнеса, интерфейса Web-магазинов и связанных с ними данных.

Даже до появления современной е-коммерции обеспечение секретности и безопасности информации имело для предприятий очень важное значение. В некоторых случаях это приводило к вводу должности инспектора безопасности по защите данных (Data Security Officer, DSO), следящего за соблюдением стандартов обеспечения безопасности данных. Интегрирование баз данных с глобальной Сетью и использование Интернета в качестве передающего звена для транзакций бизнеса сделали задачу обеспечения безопасности информации еще более актуальной.

Для успешной е-коммерции необходимо гарантировать безопасность и секретность всех коммерческих сделок и данных, связанных с этими транзакциями. В контексте



е-коммерции безопасность включает в себя все действия, связанные с защитой данных и других компонентов е-коммерции от случайного или преднамеренного (возможно, незаконного) доступа к ним или их использования неавторизованными пользователями. *Секретность (privacy)* связана с правами отдельных личностей и организаций и определяет, "кто, что, когда, где и как" может использовать данные.

Глобальная Сеть (World Wide Web) изначально создавалась для совместного использования информации, а не для ее защиты. С этой точки зрения Web хорошо подходит для некоммерческих и адвокатских организаций, которые в большей степени заинтересованы в распространении информации, чем в обеспечении ее секретности. Однако условие конфиденциальности информации в коммерческих организациях требует обеспечения безопасности и секретности в транзакциях е-коммерции. Например, вряд ли кто-нибудь стал бы использовать кредитные карты при онлайнowych платежах, если бы информация кредитных карт не была должным образом защищена от несанкционированного доступа! Как можно защитить от изменений сообщения, передаваемые по Интернету? Как можно идентифицировать отправителя? Как можно защитить Web-витрины от атак кибервандалов? Хотя абсолютно надежного способа защиты от возможных угроз и нет, все же имеется ряд технологий, с помощью которых решаются проблемы обеспечения безопасности.

Безопасность данных е-коммерции должна обеспечиваться от начала транзакции до ее завершения. Например, исследуем следующий сценарий онлайновой покупки, представленный на рис. 14.4.

1. Клиент заказывает товар, вводя информацию о заказе и о своей кредитной карте на коммерческой Web-странице.
2. Эта информация передается от клиентского компьютера по Интернету на коммерческий сервер.
3. Коммерческий сервер использует стороннюю компанию для обработки авторизации платежа.
4. Для авторизации данной транзакции компания, обрабатывающая информацию о платежах, связывается с компанией, предоставившей клиенту кредитную карту.
5. Компания, выдавшая кредитную карту, авторизует транзакцию.
6. Коммерческая компания получает авторизацию, сохраняет заказ и сведения об оплате в базе данных и посылает подтверждение заказа клиенту.
7. Затем компания пользуется услугами сторонней компании по доставке товаров для поставки заказанного товара клиенту.
8. Клиент получает заказ и подтверждает доставку.

В этом сценарии процедуры и технологии обеспечения безопасности должны обеспечивать следующее.

- ❑ *Гарантию идентификации участников транзакции*, подтверждая, что и покупатель и продавец — именно те, за кого они себя выдают. Иначе говоря, необходим безопасный способ идентификации участников транзакции и аутентификации сообщений.
- ❑ *Защиту данных транзакции от неавторизованного изменения* во время передачи их по Интернету. Интернет состоит из миллионов взаимосвязанных сетей. Данные

е-коммерции для передачи их от клиента к серверу должны пройти по нескольким различным сетям, что увеличивает риск хищения информации, ее изменения или утери.

- **Защиту ресурсов (данных и компьютеров).** Сюда включается защита конечного пользователя и коммерческой информации, хранящейся на Web-сервере и в базе данных, от неавторизованного доступа. Сюда также входит защита сервера от атак злоумышленников (хакеров), которые намереваются проникнуть в систему с целью изменения или хищения информации или для ухудшения нормальной работы путем ограничения доступности ресурсов.

### Дополнительная информация

Более подробные сведения о проблемах безопасности в е-коммерции и методах их решений можно найти на сайте [www.course.com](http://www.course.com) (поиск по 0-619-06269-X).

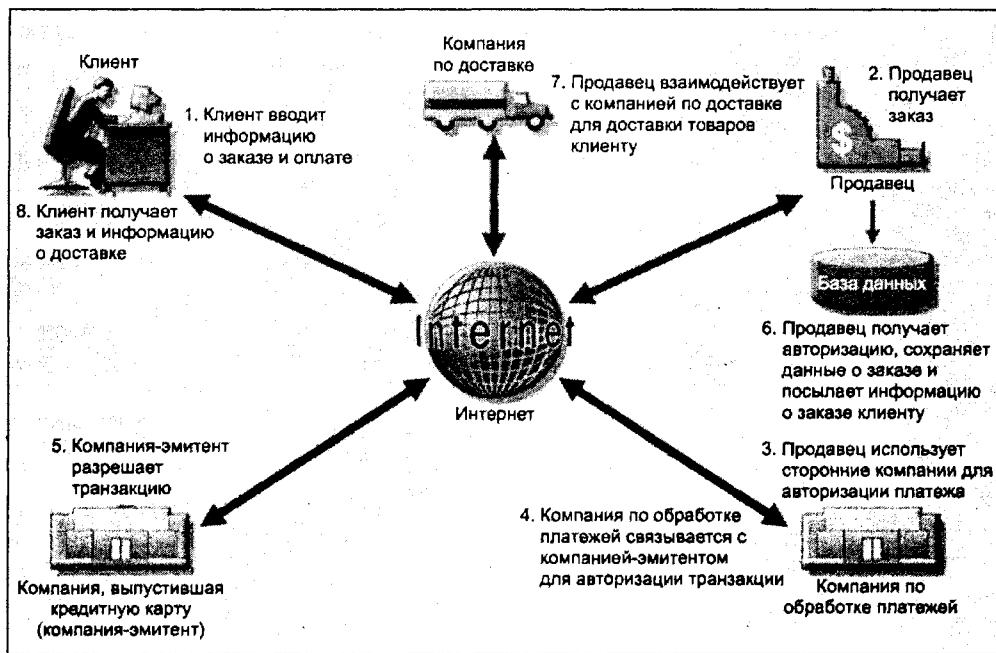


Рис. 14.4. Пример транзакции в е-коммерции

## 14.7. Обработка платежей

Ключевая функция Web-сайта е-коммерции — возможность обработки онлайн-платежей при покупке товаров и/или услуг. В традиционных коммерческих сделках, таких, например, как покупка CD в музыкальном магазине, вы оплачиваете свою покупку наличными, по чеку, почтовым переводом или по кредитной карте. На Web

самая распространенная форма оплаты — кредитная карта, хотя имеются и другие, менее популярные формы. В этом разделе будут кратко представлены три технологии, используемые для обработки электронных платежей: электронные деньги (digital cash), онлайнновая обработка кредитных карт и электронный бумажник (electronic wallet).

### **Дополнительная информация**

Дополнительную информацию о способах и компонентах электронной оплаты можно найти на сайте [www.course.com](http://www.course.com) (поиск по ISBN 0-619-06269-X).

## **14.7.1. Электронные деньги**

К одному из недостатков оплаты с помощью кредитных карт относится риск хищения информации о вашей кредитке, что может привести к появлению "неожиданных" платежей с вашего счета. В отличие от этого, если кто-то "стянул" у вас из бумажника пять долларов, то вы потеряете только эти пять долларов, и вам уже все равно, что будет куплено на эти украденные деньги. В этом смысле использование наличных денег дает определенные преимущества. *Электронные деньги (digital cash)* являются цифровым эквивалентом обычных денег (монет или купюр определенного достоинства). Для идентификации участников транзакции электронные деньги используют *цифровые сертификаты (digital certificates)*, при этом требуются гарантии банка или финансового учреждения, от которого пользователь получил электронные деньги. Когда пользователь приобретает электронные деньги, он получает набор электронных купюр или монет, обеспеченных финансовым учреждением и представленным двоичным идентификатором (ID). Пользователь может передавать электронные деньги коммерческим учреждениям для оплаты товаров и/или услуг. Коммерческое предприятие затем представляет электронные деньги в банк для зачисления их на свой счет. Технология использования электронных денег находится в стадии развития и поддерживается несколькими конкурирующими компаниями, а также несколькими стандартами.

Хотя стоимость транзакций, выполняемых с помощью электронных денег, намного меньше по сравнению с банковскими чеками или кредитными картами, предприятия торговли и клиенты до сих пор редко пользуются электронными деньгами. Это происходит по следующим причинам:

- ☐ предприятия торговли, банки и клиенты очень неохотно меняют формы своей деятельности;
- ☐ существуют различные организационные барьеры;
- ☐ клиенты чаще всего пользуются кредитными картами как наиболее удобным способом оплаты онлайнновых покупок.

Примерами компаний, поддерживающих электронные деньги, являются DigiCash, CyberCash, eCash, PayPal и Clickshare. Учитывая столь невысокую популярность, будущее многих компаний, предоставляющих электронные деньги, выглядит не очень радужно.

## 14.7.2. Обработка кредитных карт

Большинство онлайнowych платежей выполняется с помощью кредитных карт. Хотя очень многие Web-магазины принимают кредитные карты, не везде используются одни и те же методы их обработки. Простейший метод обработки — ручной. При таком сценарии продавец собирает информацию о кредитной карте клиента через Web-сайт с помощью безопасного подключения (в некоторых случаях покупатель должен сделать фактический заказ товара по телефону). Продавец затем связывается с учреждением, выдавшим кредитную карту (по телефону или с помощью устройства проверки кредитных карт) для авторизации клиента. Если возникают какие-то проблемы, например, нехватка денег на счете, неправильный номер карты или карта оказалась краденой, торговое предприятие сообщает клиенту по электронной почте о возникшей проблеме. Хотя такой многоступенчатый процесс все еще имеет место в реальной жизни в некоторых небольших Web-компаниях, его неэффективность вынуждает продавцов модифицировать его или вообще отказаться от него.

Более эффективным вариантом ручного способа является метод, при котором торговое предприятие пользуется программным обеспечением обработки кредитных карт сторонних организаций, таких, например, как IC Verify ([www.icverify.com](http://www.icverify.com), подразделение CyberCash), или использует службы сторонних компаний, таких, например, как InternetSecure Inc. ([www.internetsecure.com](http://www.internetsecure.com)). В этом случае Web-сервер предприятия торговли, сервер компании обработки платежей, клиент и банк предприятия торговли при обработке платежа будут работать совместно. В простейшем случае эта процедура выглядит следующим образом.

1. Клиент посылает информацию о своей кредитной карте по Web на сервер продавца.
2. Web-сервер продавца проверяет заказ и посылает информацию о кредитной карте в компанию, занимающуюся обработкой кредитных карт.
3. Компания обработки кредитных карт проверяет кредитку в банке клиента и получает необходимую авторизацию.
4. Web-сервер продавца получает подтверждение транзакции.
5. Web-сервер продавца отправляет клиенту электронную почту с подтверждением заказа на доставку товара.
6. Когда товар доставлен, продавец и компания по обработке кредитных карт завершают расчеты.

Одна важная особенность всех систем обработки кредитных карт состоит в том, что вся оплата совершается после поставки товара или оказания услуги клиенту.

Во время написания этой книги две основные компании, выдающие кредитные карты, Visa и MasterCard, разработали систему Secure Electronic Transaction (защита электронных платежей). *Secure Electronic Transaction (SET, протокол защиты электронных платежей)* ставит своей целью обеспечить стандартный способ осуществления транзакций кредитных карт через Интернет. Безопасность автоматически обеспечивается для всех коммуникаций между клиентом, продавцом и финансовыми институтами. Одно из основных достоинств системы SET состоит в том, что она предоставляет покупателю способ передачи информации о его кредитной карте

предприятию, выдавшему кредитную карту, так, чтобы продавец не мог видеть эту информацию, — требуя, чтобы предприятие торговли шифровало всю информацию о кредитке. Протокол SET получил широкое распространение среди большинства видных интернет-компаний и поставщиков. Протокол SET также определяет минимальные требования к сетевой инфраструктуре и структуре безопасности компаний, собирающихся выполнять проверку электронных платежей через Web.

### 14.7.3. Электронный бумажник

В действительности желательно хранить деньги, чеки и кредитные карты в бумажнике. Если вы делаете покупки через Интернет, то, скорее всего, пользуетесь кредитной картой. Если вы часто пользуетесь услугами интернет-магазинов, то вам, вероятно, надоедает вводить одну и ту же информацию о вашей кредитке при каждой покупке, к тому же при вводе этих данных всегда есть вероятность ошибки. Для решения этой проблемы можно использовать электронный бумажник. *Электронный бумажник (electronic wallet)* это эквивалент физического бумажника — в нем содержатся кредитные карты конечного пользователя, электронные деньги и другая персональная информация, например, адрес доставки, телефонные номера и адреса электронной почты. Электронный бумажник — это небольшая программа, используемая совместно с Web-обозревателем и Web-сайтом торговой организации для автоматического ввода информации об оплате онлайн-покупок.

## 14.8. Проектирование баз данных для приложений е-коммерции

Опыт подсказывает, что не обязательно кипятить воду новым способом каждый раз, когда хочешь выпить чаю. Точно так же при проектировании базы данных для е-коммерции нет нужды изобретать "новые" технологии проектирования. Наоборот, техника проектирования, которую вы изучили по этой книге (ER-моделирование, нормализация, SDLC, DBLC, управление транзакциями и т. д.), обеспечивает средства и знания, необходимые для разработки удачной базы данных для е-коммерции. Однако для баз данных е-коммерции есть несколько специфических требований, с которыми мы до этого не встречались. Вот почему в этом разделе представлены основы проектирования БД для приложения е-коммерции.

Начнем с определения сферы действия нашей базы данных. Простой Web-сайт е-коммерции должен содержать, по крайней мере, основные средства, обеспечивающие продажу товаров и услуг. Поэтому база данных должна поддерживать возможность представления на Web-сайте перечня доступных товаров и услуг и уметь проводить основные транзакции платежей. Кроме того, Web-сайт е-коммерции должен предоставлять пользовательские сервисы, возврат товара и настройку Web-сайта для удобства пользователя. Для этого в проект базы данных е-коммерции необходимо включить несколько дополнительных таблиц. Однако основное внимание мы уделим сущностям базы данных, непосредственно участвующим в обеспечении продажи товаров.

Чтобы начать процесс проектирования, установим некоторые главные бизнес-правила и определим их влияние на проект:

- ❑ основная цель проекта e-коммерции — продажа товаров клиентам. Поэтому база данных должна, прежде всего, иметь две таблицы PRODUCT (товар) и CUSTOMER (клиент);
- ❑ каждый клиент может разместить один или несколько заказов. Каждый заказ размещается только одним клиентом. Поэтому между таблицами CUSTOMER и ORDER (заказ) есть связь 1:M;
- ❑ каждый заказ содержит одну или более строк (заказанных позиций). Каждая строка заказа содержится в заказе. Поэтому между таблицами ORDER и ORDLINE (строка заказа) есть связь 1:M;
- ❑ каждая строка заказа относится к одному товару. Каждый товар может появиться в нескольких строках заказа (компания может продать больше одного струйного принтера HP!). Поэтому между таблицами PRODUCT и ORDLINE есть связь 1:M;
- ❑ клиенты, просматривающие каталог товаров, должны видеть товары, сгруппированные по категориям или типам (например, хорошо было бы разделить список товаров по компьютерам, принтерам, прикладному программному обеспечению, операционным системам и т. д.). Поэтому каждый товар (PRODUCT) принадлежит одному типу (PRODTYPE) и с каждым типом (PRODTYPE) связан один или несколько товаров;
- ❑ клиенты, которые просматривают Web-каталог, должны иметь возможность выбрать товар и положить его в электронную корзину. Электронная корзина может временно хранить товары до того времени, как клиент закончит работу. Поэтому следующая необходимая сущность — SHOPCART (электронная корзина). Каждая электронная корзина (SHOPCART) принадлежит клиенту (CUSTOMER) и связана с одним или более товаров (PRODUCT);
- ❑ когда клиент заканчивает работу, он вводит информацию о своей кредитной карте и адрес доставки. Эта информация добавляется в таблицу ORDER (обратите внимание, что необходимые атрибуты определяются бизнес-правилом);
- ❑ после получения авторизации кредитной карты заказ помещается в пул заказов корзины покупателя. Информация SHOPCART используется при создании заказа (ORDER), в котором содержится одна или более строк (ORDLINE). *После того как заказ размещен и пользователь покидает Web-сайт, данные электронной корзины удаляются;*
- ❑ поскольку магазин предлагает множество вариантов доставки, создана таблица SHIPTOPTION (варианты доставки) для хранения информации о доставке, т. е. "наземным транспортом", "в течение двух дней", "на следующий день", UPS (единая почтовая служба), FedEx (федеральная служба доставки США) и т. д.;
- ❑ поскольку магазин предлагает множество вариантов оплаты, создана таблица PMTOPTION (варианты оплаты) для хранения сведений о каждом варианте оплаты, т. е. MasterCard, Visa, American Express и т. д.;
- ❑ поскольку в каждом штате свое налоговое законодательство, созданы две таблицы — STATE (штат) и TAXRATE (налоговая ставка) для отслеживания штатов (или стран) и их налоговых ставок.

### Примечание

Для экономии места и чтобы убедиться, что вы в достаточной степени усвоили материал гл. 3 и 4 по переводу бизнес-правил в ER-диаграмму, здесь ER-диаграммы разрабатываться не будут. Основные сущности мы привели в табл. 14.3.

**Таблица 14.3.** Основные таблицы базы данных e-коммерции

| Таблица                       | Описание                                                                                                                                                                                                                                                                                                                                            |
|-------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CUSTOMER</b>               | Содержит подробные сведения по каждому зарегистрированному пользователю. В этой таблице будут храниться основные данные по клиентам, условия доставки, данные по кредитной карте и данные по оплате (для клиентского счета). Некоторые клиенты предпочитают не регистрироваться; они будут вводить сведения о себе всякий раз при размещении заказа |
| <b>PRODUCT</b>                | Содержит сведения о товарах, например, инвентарные номера, цены, количество и т. д.                                                                                                                                                                                                                                                                 |
| <b>PRODTYPE</b>               | Определяет основную классификацию типов товара                                                                                                                                                                                                                                                                                                      |
| <b>ORDER</b>                  | Содержит информацию о заказах, например, дата, количество, заказчик и т. д.                                                                                                                                                                                                                                                                         |
| <b>ORDLINE</b>                | Содержит выбранные товары в каждом заказе                                                                                                                                                                                                                                                                                                           |
| <b>Дополнительные таблицы</b> |                                                                                                                                                                                                                                                                                                                                                     |
| <b>SHOPCART</b>               | В этой таблице указывается количество единиц каждого товара, приобретенного клиентом. Это "рабочая" таблица, содержимое которой удаляется после того, как клиент покидает Web-сайт или закрывает обозреватель                                                                                                                                       |
| <b>PMTTYPE</b>                | Варианты оплаты, предлагаемые магазином                                                                                                                                                                                                                                                                                                             |
| <b>SHIPTYPE</b>               | Варианты доставки, предлагаемые магазином                                                                                                                                                                                                                                                                                                           |
| <b>TAXRATE</b>                | Ставки налогов для каждого штата или страны                                                                                                                                                                                                                                                                                                         |
| <b>STATE</b>                  | Список штатов или стран, в которых выполняется начисление налогов                                                                                                                                                                                                                                                                                   |
| <b>PROMOTION</b>              | Специальные предложения, такие как ваучеры (поручительства) или ценовые скидки                                                                                                                                                                                                                                                                      |
| <b>PRICEWATCH</b>             | Клиенты, которых необходимо уведомлять о том, что цена определенного товара достигла заданного уровня                                                                                                                                                                                                                                               |
| <b>PRODPRICE</b>              | Необязательная таблица, используемая для управления различными уровнями цен                                                                                                                                                                                                                                                                         |

После определения таблиц, необходимых для обеспечения деятельности e-коммерции, следует определить основные атрибуты каждой таблицы. Помните, что приведенный список атрибутов является только примером наиболее важных (и используемых чаще других) атрибутов, но наверняка далеко не полным (ваша рабочая сре-

да определяет, какие из этих атрибутов значимы, а какие атрибуты необходимо добавить).

### Примечание

В рамках обсуждения отметим необходимость некоторого опыта проектирования и программирования при разработке Web-сайтов e-коммерции. Например, поскольку данные обеспечения Web-транзакций хранятся в таблицах, желательно определить несколько полей (названия товаров или их описание) для более наглядного отображения Web-страницы. Также такие поля, как номера кредитных карт и пароли, должны храниться в зашифрованном формате. Даже сами названия полей должны шифроваться, что уменьшает вероятность несанкционированного доступа к такой информации (шифрование поддерживают такие базы данных, как Oracle 9i и DB2).

## Таблица CUSTOMER

В таблице CUSTOMER содержатся сведения о каждом зарегистрированном клиенте. Помните, что некоторые клиенты не хотят регистрироваться, поскольку не желают предоставлять о себе подробные сведения. Поэтому необходимо принять решение о том, надо ли требовать регистрации при совершении каждой покупки.

- ☐ При обязательной регистрации на Web-сайте необходимо иметь учетную форму для сбора сведений о новых клиентах. Также требуется, чтобы пользователь заполнил форму регистрации до того, как он начнет просматривать каталог товаров. Учетные данные по адресу, доставке и информация о кредитной карте клиента могут автоматически помещаться в заказ при утверждении его клиентом (одним из преимуществ регистрации может быть предоставление таким клиентам скидок).
- ☐ При необязательной регистрации нет необходимости заполнять учетные формы для каждой продажи или посещения Web-страницы. Клиент должен будет ввести всю информацию по доставке товара и по своей кредитной карте при каждом оформлении заказа.

Очевидно, самый простой вариант — отсутствие регистрации. Поэтому мы решили сделать регистрацию необязательной. Основные атрибуты таблицы CUSTOMER представлены в табл. 14.4.

**Таблица 14.4. Таблица CUSTOMER**

| Атрибут    | Описание                                        | PK/FK |
|------------|-------------------------------------------------|-------|
| CUS_ID     | Идентификатор клиента — создается автоматически | PK    |
| CUS_DATEIN | Дата добавления информации о клиенте в таблицу  |       |
| CUS_LNAME  | Фамилия                                         |       |
| CUS_FNAME  | Имя                                             |       |
| CUS_ADDR1  | Первая строка адреса                            |       |
| CUS_ADDR2  | Вторая строка адреса                            |       |
| CUS_CITY   | Город                                           |       |



Таблица 14.4 (окончание)

| Атрибут      | Описание                                                                                                                                                            | PK/FK |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| CUS_STATE    | Штат/или регион для международных клиентов                                                                                                                          | FK    |
| CUS_ZIP      | ZIP-код (почтовый индекс)                                                                                                                                           |       |
| CUS_CNTRY    | Страна                                                                                                                                                              |       |
| CUS_PHONE    | Телефон                                                                                                                                                             |       |
| CUS_EMALE    | Адрес электронной почты (e-mail)                                                                                                                                    |       |
| CUS_LOGINID  | Регистрационный идентификатор для зарегистрированных пользователей                                                                                                  |       |
| CUS_PASSWD   | Пароль регистрации — зашифрованное поле                                                                                                                             |       |
| CUS_CCNAME   | Имя в том виде, как оно записано на кредитной карте                                                                                                                 |       |
| CUS_CCNUM    | Номер кредитной карты — зашифрованное поле                                                                                                                          |       |
| CUS_CCEXDATE | Дата окончания действия кредитной карты в формате мм/гг                                                                                                             |       |
| CUS_ACRNUM   | Номер дебиторского счета — для взаимодействия с внутренней системой счетов или номером почтового отделения                                                          |       |
| CUS_BLLADDR1 | Первый адрес доставки счета                                                                                                                                         |       |
| CUS_BLLADDR2 | Второй адрес доставки счета                                                                                                                                         |       |
| CUS_BLLCITY  | Город адреса доставки счетов                                                                                                                                        |       |
| CUS_BLLSTATE | Штат адреса доставки счетов                                                                                                                                         |       |
| CUS_BLLZIP   | Почтовый индекс адреса доставки счетов                                                                                                                              |       |
| CUS_BLLCNTRY | Страна адреса доставки счетов                                                                                                                                       |       |
| SHIP_ID      | Предпочтительный способ доставки                                                                                                                                    |       |
| CUS_SHPADDR1 | Первый адрес доставки                                                                                                                                               |       |
| CUS_SHPADDR2 | Второй адрес доставки                                                                                                                                               |       |
| CUS_SHPCITY  | Город адреса доставки                                                                                                                                               |       |
| CUS_SHPSTATE | Штат адреса доставки                                                                                                                                                |       |
| CUS_SHPZIP   | Почтовый индекс адреса доставки                                                                                                                                     |       |
| CUS_SHPCNTRY | Страна адреса доставки                                                                                                                                              |       |
| CUS_TAXID    | Идентификатор налога для клиентов, не облагаемых налогом                                                                                                            |       |
| CUS_MBRTYPE  | Тип членства — используется для специальных цен на товар в соответствии с уровнем членства, например, обычная цена, цена для зарегистрированных членов, особая цена |       |

## Таблица PRODUCT

Таблица PRODUCT — центральная сущность нашей базы данных. В ней содержится значимая информация о товарах, предлагаемых на Web-сайте. Эта таблица (табл. 14.5) связана с таблицами PRODTYPE, ORDLINE и PROMOTION.

**Таблица 14.5.** Таблица PRODUCT

| Атрибут          | Описание                                                                                                                                                                                                                                            | PK/FK |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| PROD_ID          | Идентификатор товара — создается автоматически                                                                                                                                                                                                      | PK    |
| PROD_NAME        | Сокращенное именование товара — показывается в специальных предложениях, счетах и т. д. Например, Verbatim CD-R                                                                                                                                     |       |
| PROD_DESCR       | Описание товара — подробное описание товара, используется на Web-страницах для информации о товаре                                                                                                                                                  |       |
| PROD_OPTIONS     | Вариант исполнения товара, например, цвет, размер, стиль (есть множество способов задания размеров, цвета для обувной и швейной промышленности, для некоторых требуется создание отдельных записей о товаре или создание других таблиц в связи 1:M) |       |
| PROD_IMAGE_1     | Указатель URL на файл с изображением товара. Может встречаться в нескольких вариантах (вид спереди, сбоку, сзади и снизу)                                                                                                                           |       |
| PROD_SKU         | Инвентарный номер, используемый поставщиком или производителем товара                                                                                                                                                                               |       |
| PROD_PARTNUM     | Шифр изделия от производителя, например, VBTM 34563                                                                                                                                                                                                 |       |
| VEND_UD          | Идентификатор поставщика изделия, т. е. глобальный идентификатор                                                                                                                                                                                    | FK    |
| PTYPE_ID         | Тип товара (категория), т. е. складской номер                                                                                                                                                                                                       | FK    |
| PROD_UNIT_SIZE   | Единица измерения изделия: коробка, блок, штука                                                                                                                                                                                                     |       |
| PROD_UNIT_QTY    | Количество единиц: 12, 6, 1                                                                                                                                                                                                                         |       |
| PROD_QOH         | Количество каждой единицы товара в наличии на складе                                                                                                                                                                                                |       |
| PROD_QORDER      | Количество в заказе — позиции заказанные, но еще не доставленные. Для определения, есть ли данная позиция на складе, вычитается количество в заказе из наличия товара на складе                                                                     |       |
| PROD_REORD_LEVEL | Уровень повторного заказа — когда количество на складе равно этому числу, товар необходимо заказывать у поставщика                                                                                                                                  |       |
| PROD_REORD_QTY   | Сколько товара необходимо заказать у поставщика                                                                                                                                                                                                     |       |
| PROD_REORD_DATE  | Примерный срок получения заказа от поставщика                                                                                                                                                                                                       |       |

Таблица 14.5 (окончание)

| Атрибут       | Описание                                                                                                                                                                                                                  | PK/FK |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| PROD_PRICE    | Обычная цена единицы товара: \$1,05 за один CD-R                                                                                                                                                                          |       |
| PROD_MSRP     | Цена производителя — чтобы показать выгоду                                                                                                                                                                                |       |
| PROD_PRICE_D1 | Скидка 1: 3% — для зарегистрированных членов или по уровню качества                                                                                                                                                       |       |
| PROD_PRICE_D2 | Скидка 2: 6% — специальная цена или по уровню качества                                                                                                                                                                    |       |
| PROD_TAX      | Yes или No (да или нет) — подлежит ли товар налогообложению?                                                                                                                                                              |       |
| PROD_ALTER_1  | Альтернативный продукт, если данного товара нет на складе. Может встречаться несколько раз. Это внешний ключ для той же таблицы товаров. Опять-таки он может быть реализован путем создания отдельной таблицы в связи 1:M | FK    |
| PROD_PROMO    | Yes или No (да или нет). Товар представлен в специальных предложениях? По умолчанию Yes                                                                                                                                   |       |
| PROD_WEIGHT   | Вес товара, требуется при оформлении доставки                                                                                                                                                                             |       |
| PROD_DIMEM    | Размер товара, требуется при оформлении доставки                                                                                                                                                                          |       |
| PROD_NOTES    | Замечания о товаре, доставка, руководство по использованию                                                                                                                                                                |       |
| PROD_ACTIVE   | Yes или No (да или нет). Если значение No ("не активен"), то товар недоступен клиентам. Имеет смысл для отозванных товаров или для товаров, продажу которых необходимо приостановить                                      |       |

В основном на один товар необходима одна строка, за исключением тех товаров, у которых имеются разные размеры, цвета или стили исполнения, такие, например, как рубашки, ботинки и т. д. В этом случае у нас имеются три возможных варианта:

- ☐ клиент вводит в заказе размер, цвет и стиль в качестве дополнительных атрибутов;
- ☐ создается уникальная запись о товаре для каждой комбинации размера, цвета и стиля;
- ☐ создается новая таблица свойств товара (PRODOPT), имеющая связь 1:M с таблицей PRODUCT. В этой таблице будет отдельная запись для каждой комбинации цвета, размера и стиля данного товара.

## Таблица PRODTYPE

В этой таблице описываются различные категории товаров. Категории могут быть ограничены одним уровнем вложенности или иметь несколько уровней. В данном примере мы решили использовать два уровня. Это позволит использовать внутри, например, категории "принтер", такие вложенные категории, как "лазерный" или "струйный" (табл. 14.6).

Таблица 14.6. Таблица PRODTYPE

| Атрибут      | Описание                                            | PK/FK |
|--------------|-----------------------------------------------------|-------|
| PTYPE_ID     | Идентификатор типа товара — создается автоматически | PK    |
| PTYPE_NAME   | Название типа товара — например, "Струйный принтер" |       |
| PTYPE_PARENT | Родительский тип товара — например, "Принтер"       | FK    |

## Таблица ORDER

В этой таблице содержится информация о заказах клиентов. После того как компания, выдавшая кредитную карту, подтверждает транзакцию, заказ добавляется в таблицу ORDER. Если кредитная карта не подтверждается (неправильный номер, кончился срок действия, краденая и т. д.), заказ не добавляется в таблицу. Для каждого нового заказа клиента в таблице ORDER появится одна строка, независимо от количества заказанных товаров. Если зарегистрированный клиент размещает свой заказ, информация о его кредитной карте и адрес доставки автоматически должна заноситься в таблицу ORDER. Таблица ORDER имеет связь 1:M с таблицей ORDLINE (табл. 14.7).

Таблица 14.7. Таблица ORDER

| Атрибут      | Описание                                                                                                                                                                                                             | PK/FK |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| ORD_ID       | Идентификатор заказа — создается автоматически                                                                                                                                                                       | PK    |
| ORD_DATE     | Дата добавления заказа в таблицу                                                                                                                                                                                     |       |
| CUS_ID       | Идентификатор клиента (необязателен) — некоторые клиенты не регистрируются. Если клиент зарегистрирован, CUS_ID автоматически добавляется Web-системой                                                               | FK    |
| PMT_ID       | Идентификатор типа платежа — задается клиентом                                                                                                                                                                       | FK    |
| ORD_CCNAME   | Имя в том виде, в каком оно представлено в кредитной карте, — копируется из данных таблицы CUSTOMER, вводится вручную (незарегистрированным клиентом) или с помощью программного обеспечения электронного бумажника  |       |
| ORD_CCNUM    | Номер кредитной карты (шифрованное поле), копируется из таблицы CUSTOMER или вводится вручную (незарегистрированным пользователем) или с помощью программного обеспечения электронного бумажника                     |       |
| ORD_CCEXDATE | Дата окончания действия кредитной карты в формате мм/гг — копируется из данных таблицы CUSTOMER, вводится вручную (незарегистрированным пользователем) или с помощью программного обеспечения электронного бумажника |       |
| SHIP_ID      | Выбранный тип доставки — вводится автоматически или вручную. Используется только в том случае, если при заказе и доставке используется одна и та же компания                                                         | FK    |

Таблица 14.7 (окончание)

| Атрибут       | Описание                                                                                                                                                      | PK/FK |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| ORD_SHIPADDR1 | Строка 1 адреса доставки — вводится автоматически или вручную                                                                                                 |       |
| ORD_SHIPADDR2 | Строка 2 адреса доставки — вводится автоматически или вручную                                                                                                 |       |
| ORD_SHIPCITY  | Город адреса доставки — вводится автоматически или вручную                                                                                                    |       |
| ORD_SHIPSTATE | Штат адреса доставки — вводится автоматически или вручную                                                                                                     | FK    |
| ORD_SHIPZIP   | Почтовый индекс адреса доставки — вводится автоматически или вручную                                                                                          |       |
| ORD_SHIPCNTY  | Страна адреса доставки — вводится автоматически или вручную                                                                                                   |       |
| ORD_SHIPDATE  | Дата доставки товара — если доставка полностью выполнена. Если доставка выполнена частично см. таблицу ORDLINE по датам поставки для каждой строки товара     |       |
| ORD_SHIPCOST  | Общая стоимость доставки — оценочная стоимость доставки заказа. Это результат применения формулы расчета стоимости доставки в соответствии с методом доставки |       |
| ORD_PRODCOST  | Общая стоимость товара — сумма стоимости товаров, умноженная на количество позиций                                                                            |       |
| ORD_TAXCOST   | Общий налог с продаж — рассчитывается добавлением налога для каждого товара таблицы ORDLINE                                                                   |       |
| PROMO_ID      | Идентификатор специального предложения в заказе (необязателен)                                                                                                | FK    |
| ORD_TOTCOST   | Общая стоимость заказа: $\text{PRODCOST} + \text{SHIPCOST} + \text{TAXCOST} - \text{PRO\_AMT}$                                                                |       |
| ORD_TRXNUM    | Номер подтверждения транзакции от компании, выдавшей кредитную карту                                                                                          |       |
| ORD_STATUS    | Статус заказа — Open (открыт), Shipped (доставлен) или Paid (оплачен)                                                                                         |       |

## Таблица ORDLINE

Эта таблица содержит одну или более строк товаров, связанных с каждым заказом. Каждая строка таблицы ORDLINE связана с одной строкой таблицы PRODUCT и с одной строкой таблицы ORDER. В таблице ORDLINE указаны количество и стоимость каждого заказанного товара. Здесь значения атрибутов PROD\_ID и ORD\_QTY получаются из таблицы SHOPCART (табл. 14.8).

Таблица 14.8. Таблица ORDLINE

| Атрибут      | Описание                                                                                                                                                                                                                      | PK/FK |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| ORL_ID       | Идентификатор строки заказа — создается автоматически                                                                                                                                                                         | PK    |
| ORD_ID       | Идентификатор заказа из таблицы ORDER                                                                                                                                                                                         | PK,FK |
| PROD_ID      | Идентификатор товара                                                                                                                                                                                                          | FK    |
| ORL_QTY      | Заказанное количество                                                                                                                                                                                                         |       |
| ORL_PRICE    | Стоимость товара — после всех скидок и специальных предложений                                                                                                                                                                |       |
| ORL_TAX      | Процент ставки налога для данного товара. Некоторые товары или клиенты могут освобождаться от налога. Если товар/клиент подлежат налогообложению, ставка налога получается в соответствии с атрибутом STATE в адресе доставки |       |
| SHIP_ID      | Компания и тип доставки, используемые для данного товара, — в том случае, когда доставка заказа происходит по частям                                                                                                          | FK    |
| ORL_SHIPDATE | Дата доставки данного товара                                                                                                                                                                                                  |       |

## Таблица SHOPCART

Специальная таблица SHOPCART (корзина) используется на Web-сайте для временного хранения товаров во время работы клиента. Для лучшего понимания того, как работает эта таблица, необходимо уяснить процесс онлайн-заказа.

- ☐ Когда клиент в первый раз посещает Web-сайт, он может зарегистрироваться или же нет. Если клиент регистрируется, Web-сервер сохраняет идентификатор клиента (CUS\_1D) в памяти.
- ☐ Клиент просматривает каталог товаров. Если клиент зарегистрирован, то ему будут представлены цены для постоянных членов, в противном случае — обычные цены (PROD\_PRICE минус соответствующие льготы: PROD\_PRICE\_D1 или PROD\_PRICE\_D2).
- ☐ Корзина автоматически назначается клиенту, когда он первый раз выберет товар, нажав кнопку **Добавить в корзину** или **Заказать сейчас**. Обозреватель клиента и Web-сервер магазина устанавливают уникальный идентификатор корзины во время безопасного сеанса связи. Этот идентификатор действует только до момента, когда клиент закончит работу, отменит заказ или покинет Web-сайт, либо закроет обозреватель.
- ☐ В корзине хранятся PROD\_ID (идентификатор товара) и количество единиц каждого товара, выбранного клиентом.
- ☐ Когда клиент нажмет кнопку **Оформить заказ**, появится экран подтверждения заказа, отображающий сведения обо всех товарах, имеющихся в электронной корзине клиента.

- Когда клиент соглашается сделать заказ, ему предъявляется другой экран, в котором можно ввести информацию о доставке и оплате. Затем клиент подтверждает заказ.
- После подтверждения заказа клиентом Web-сервер требует подтверждения транзакции от компании, выдавшей кредитную карту. Этот процесс может длиться от 10 секунд до одной минуты.
- После получения подтверждения данные таблиц ORDER и ORDLINE сохраняются.
- Данные из таблицы SHOPCART удаляются.

Структура таблицы SHOPCART представлена в табл. 14.9.

**Таблица 14.9.** Таблица SHOPCART

| Атрибут      | Описание                                                                                                                                                                                                                                                                                                                                                                                        | PK/FK |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| CART_ID      | Уникальный идентификатор корзины — создается автоматически                                                                                                                                                                                                                                                                                                                                      | PK    |
| CART_PROD_ID | Идентификатор товара — копия значения PROD_ID. Поскольку таблица SHOPCART может сильно разрастаться по мере добавления и удаления, не стоит связывать ее с таблицей PRODUCT из-за возможного ухудшения производительности. Помните: эти значения автоматически копируются в таблицу ORDLINE, когда выполняется транзакция данного клиента. Однако связь можно и установить, если это необходимо | PK    |
| CART_QTY     | Заказанное количество                                                                                                                                                                                                                                                                                                                                                                           |       |

## Таблица PMTTYE

В этой таблице содержится одна строка для каждого метода платежа, поддерживаемого магазином (табл. 14.10).

**Таблица 14.10.** Таблица PMTTYE

| Атрибут      | Описание                                                                                                                                         | PK/FK |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| PMT_ID       | Идентификатор типа платежа — создается автоматически                                                                                             | PK    |
| PMT_NAME     | Имя платежа — VISA, MasterCard, American Express, Net30                                                                                          |       |
| PMT_MCHNT_ID | Идентификатор магазина — используется системой платежей. Этот идентификатор дается магазину при регистрации его в компании выдачи кредитных карт |       |
| PMT_NOTES    | Дополнительные примечания                                                                                                                        |       |

## Таблица SHIPTYPE

В этой таблице содержится одна строка на каждый метод доставки, поддерживаемый данным магазином (табл. 14.11).

**Таблица 14.11. Таблица SHIPTYPE**

| Атрибут   | Описание                                                                                                                                                                                                                                                 | PK/FK |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| SHIP_ID   | Идентификатор типа доставки — создается автоматически                                                                                                                                                                                                    | PK    |
| SHIP_NAME | Название компании по доставке — UPS Next Day ("UPS на следующий день"), UPS Three Days ("UPS за три дня"), FedEx Overnight и т. д.                                                                                                                       |       |
| SHIP_COST | Стоимость доставки единицы товара — зависит от компании по доставке. В большинстве случаев это связано с ZIP-кодом клиента, размером и весом товара. Поэтому, вероятнее всего, при разработке проекта в этой таблице потребуются дополнительные атрибуты |       |
| PMT_NOTES | Дополнительные примечания по доставке                                                                                                                                                                                                                    |       |

## Таблица TAXRATE

В этой таблице содержатся различные ставки налога с продаж для каждого штата. Обратите внимание, что налог с продаж применяется не ко всем заказам, а только для тех штатов, в которых от магазинов требуется собирать налог с товаров, проданных в данном штате. Налоговые требования определяются федеральными законами и законами штата. Обычно налог с продаж определяется адресом доставки. Однако для этой цели может также быть использован адрес доставки счетов клиента или адрес оплаты кредитной карты.

Эта таблица связана с таблицей STATE. В таблице должны также учитываться учреждения, не облагаемые налогами (табл. 14.12).

**Таблица 14.12. Таблица TAXRATE**

| Атрибут   | Описание                                                               | PK/FK  |
|-----------|------------------------------------------------------------------------|--------|
| STATE_ID  | Идентификатор штата из таблицы STATE — требуется обязательно           | PK, FK |
| TAX_RATE  | Применяемая ставка налога с продаж — требуется обязательно             |        |
| NAX_NOTES | Дополнительные примечания — например, причина взимания налогов и т. д. |        |

## Таблица STATE

В этой таблице хранится одна запись для каждого штата или страны. Таблица связана с таблицей TAXRATE. В таблице могут содержаться модифицированные записи



для стран, использующих почтовые регионы или другие идентификаторы. Эта таблица может также содержать поле COUNTRY (страна) для предприятий, у которых имеются офисы в разных странах (табл. 14.13).

Таблица 14.13. Таблица STATE

| Атрибут    | Описание                                      | PK/FK |
|------------|-----------------------------------------------|-------|
| STATE_ID   | Идентификатор штата — создается автоматически | PK    |
| STATE_NAME | Название штата — требуется обязательно        |       |

## Таблица PROMOTION

Эта таблица используется для представления специальных предложений цен на товары. В ней содержится одна запись для каждого специального предложения, устанавливаемого магазином. У всех специальных цен есть дата начала и дата окончания действия. Некоторые специальные предложения могут применяться к одной линейке товаров или к отдельному товару. Некоторые специальные предложения могут быть выражены ценовой скидкой, а другие предложения (например, поручительства) — специальной страховой суммой. Атрибуты, представленные в табл. 14.14, можно комбинировать разными способами для предоставления различных предложений.

Таблица 14.14. Таблица PROMOTION

| Атрибут       | Описание                                                                                                                     | PK/FK |
|---------------|------------------------------------------------------------------------------------------------------------------------------|-------|
| PROMO_ID      | Идентификатор специального предложения — создается автоматически                                                             | PK    |
| PROMO_NAME    | Название предложения — Summer sale (летняя распродажа), Christmas sale (рождественская распродажа), Voucher (поручительство) |       |
| PROMO_DATE    | Дата создания специального предложения                                                                                       |       |
| PROMO_BEGDATE | Дата начала действия предложения                                                                                             |       |
| PROMO_ENDDATE | Дата окончания действия предложения                                                                                          |       |
| P_TYPE_ID     | Идентификатор типа товара, на который действует специальное предложение                                                      | FK    |
| PROD_ID       | Товар(ы), на который(ые) действует специальное предложение (необязательно)                                                   | FK    |
| PROMO_MINQTY  | Минимальное количество единиц товара, необходимое для действия специального предложения (необязательно)                      |       |
| PROMO_MAXQTY  | Максимальное количество единиц товара, для которого действует специальное предложение                                        |       |
| PROMO_MINPUR  | Минимальная стоимость покупки, необходимая для действия специального предложения (необязательно)                             |       |

Таблица 14.14 (окончание)

| Атрибут       | Описание                                                                          | PK/FK |
|---------------|-----------------------------------------------------------------------------------|-------|
| PROMO_PCTDISC | Скидка в процентах по специальному предложению (необязательно)                    |       |
| PROMO_DOLLAR  | Размер специального предложения в долларах (необязательно)                        |       |
| PROMO_CEILING | Максимальное значение специального предложения (если задан процент) — обязательно |       |

## Таблица PRICEWATCH

Множество Web-сайтов е-коммерции предлагают сервис "наблюдение за ценой" (pricewatch). Эта служба рассылает электронную почту клиентам, когда цена на товар станет ниже или равна цене, предварительно установленной клиентом. Таблица PRICEWATCH реализует такую возможность (табл. 14.15). Вариант такой таблицы можно использовать для обратного предложения цен клиентами.

Таблица 14.15. Таблица PRICEWATCH

| Атрибут      | Описание                                                                                                                                                      | PK/FK |
|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| PW_ID        | Идентификатор наблюдения за ценой — создается автоматически                                                                                                   | PK    |
| PW_DATE      | Штамп времени, вставляемый в таблицу                                                                                                                          |       |
| CUS_ID       | Идентификатор клиента. Этот сервис предоставляется всем потенциальным клиентам, а также зарегистрированным пользователям                                      | FK    |
| PW_CUS_NAME  | Имя клиента — обязательный атрибут. Вводится вручную или автоматически копируется из таблицы CUSTOMER                                                         |       |
| PW_CUS_EMAIL | Адрес электронной почты клиента — обязательный атрибут. Адрес e-mail, на который клиенту отправляется уведомление                                             |       |
| PW_ENDDATE   | Дата окончания действия сервиса наблюдения за ценой. Клиент вводит эту дату по желанию                                                                        |       |
| PROD_ID      | Товары, для цен на которые требуется установить этот сервис                                                                                                   | FK    |
| PW_LOWPRICE  | Цена, о которой клиента необходимо уведомлять. Если цена на товар равна или меньше этого значения, то система посылает клиенту сообщение по электронной почте |       |

## Таблица PRODPRICE

Таблица PRODPRICE предназначена для управления многоуровневыми ценами. Некоторые сайты е-коммерции предлагают множество различных цен в зависимости от количества заказанных товаров. Например, если вы покупаете от одной до пяти (включительно) пар обуви, то цена пары — \$39,95. Однако если вы покупаете шесть пар или более, то цена пары может быть снижена до \$35,95. Эта таблица имеет связь 1:M с таблицей PRODUCT (табл. 14.16). При использовании многоуровневых цен атрибут PROD\_PRICE в таблице PRODUCT не используется.

**Таблица 14.16.** Таблица PRODPRICE

| Атрибут         | Описание                                                                                                            | PK/FK |
|-----------------|---------------------------------------------------------------------------------------------------------------------|-------|
| PROD_ID         | Идентификатор товара из таблицы PRODUCT                                                                             | PK,FK |
| PRODPRC_QTYFROM | Начальное значение диапазона количества единиц купленного товара — обязательный атрибут. Например, 1, или 6, или 11 | PK    |
| PRODPRC_QTYTO   | Конечное значение диапазона количества единиц купленного товара — обязательный атрибут. Например, 5, 10 или 9999    | PK    |
| PRODPRC_PRICE   | Стоимость диапазона товара — обязательный атрибут                                                                   |       |

## 14.9. Расширяемый язык разметки (XML)

Ранее в этой главе мы говорили, что большинство транзакций е-коммерции происходит между двумя предприятиями. Поскольку B2B-стиль е-коммерции интегрирует бизнес-процессы компаний, он требует передачи коммерческой информации между различными элементами бизнеса. Но способ ведения бизнеса, идентификация и использование данных существенно отличаются от предприятия к предприятию (например, неясно: *product code* — то же самое, что *item-ID*?). До самого последнего времени считалось, что заказы будут передаваться по Сети в форме HTML-документа, содержащего теги и данные. Web-страница HTML, отображенная в Web-обозревателе, должна включать в себя теги форматирования, а также сведения о заказе. Если приложению необходимо получать данные о заказе от Web-страницы, то оказывается, что не так-то просто извлечь из HTML-документа эти сведения (например, номер заказа, дату, номер клиента, заказанную позицию, количество, стоимость или сведения об оплате). (На самом деле, HTML-подход требует каждый раз просматривать весь документ.) HTML-документ может только описывать способ отображения заказа в Web-обозревателе, но он не позволяет манипулировать элементами данных заказа, т. е. датами, информацией по доставке, сведениями об оплате, информацией о товаре и т. д. Для решения этой проблемы был разработан новый стандарт языка разметки, называемый eXtensible Markup Language, или XML.

Расширяемый язык разметки (*eXtensible Markup Language, XML*) представляет собой метаязык, предназначенный для представления и манипулирования элементами данных. Язык XML разработан для облегчения обмена структурированными доку-

ментами, например, заказами или счетами, по Интернету. Консорциум W3C (World Wide Web Consortium)<sup>3</sup> опубликовал первый стандарт XML в 1998 году. Этот стандарт подготовил почву для практического использования XML в качестве независимой от поставщика платформы. Поэтому неудивительно, что XML быстро стал стандартом обмена информацией для приложений e-коммерции.

Метаязык XML позволяет определять новые теги (например, <ProdPrice>) для описания элементов данных, используемых в XML-документах. Учитывая эту возможность, XML можно назвать *расширяемым* (extensible) языком. XML берет свое начало в *стандартном обобщенном языке разметки SGML* (Standard Generalized Markup Language) — международный стандарт для публикаций и распространения сложных технических документов, используемый в авиации и военной промышленности). Язык SGML слишком сложен и громоздок для Web. XML-документ, как и HTML-документ (который также происходит из SGML), является обычным текстовым файлом, однако он обладает несколькими очень важными дополнительными свойствами:

- ☐ XML позволяет определять новые теги для описания элементов данных, например <ProductID>;
- ☐ XML чувствителен к регистру: <ProductID> это не то же самое, что <ProductId>;
- ☐ XML-теги должны быть правильно построены, т. е. каждому открывающему тегу должен соответствовать закрывающий тег. Например, идентификатор товара требует следующего формата:

```
<ProductId>2345-AA</ProductId>
```

- ☐ XML-теги должны быть правильно вложены. Например, правильно вложенный тег XML должен выглядеть так:

```
<Product><ProductId>2345-AA</ProductId></Product>
```

- ☐ для добавления комментариев в XML-документ используются символы <-- и -->;
- ☐ префиксы "XML" и "xml" зарезервированы только для тегов XML.

Язык XML не является новой версией или заменой HTML, он связан с описанием и представлением данных, а не с их отображением (отображение данных остается прерогативой HTML). Язык XML предоставляет определенную семантику, облегчающую совместное использование, обмен и манипуляцию структурированными документами вне границ компании. Говоря кратко, XML и HTML выполняют параллельные, а не перекрывающиеся функции. Для иллюстрации использования XML представим себе пример приложения В2В-стиля, в котором компания А использует XML для обмена информацией о товаре с компанией В через Интернет. На рис. 14.5 представлено содержимое документа ProductList.xml.

На примере XML-документа, представленного на рис. 14.5, можно выяснить некоторые важные возможности XML:

- ☐ первая строка представляет объявление XML-документа и она обязательна;
- ☐ каждый XML-документ имеет *корневой элемент*. В нашем примере вторая строка объявляет корневой элемент ProductList;

---

<sup>3</sup> Страница консорциума W3C находится по адресу [www.w3.org](http://www.w3.org). Для получения дополнительной информации об усилиях, предпринимаемых для развития языка XML, вы можете посетить этот Web-сайт.

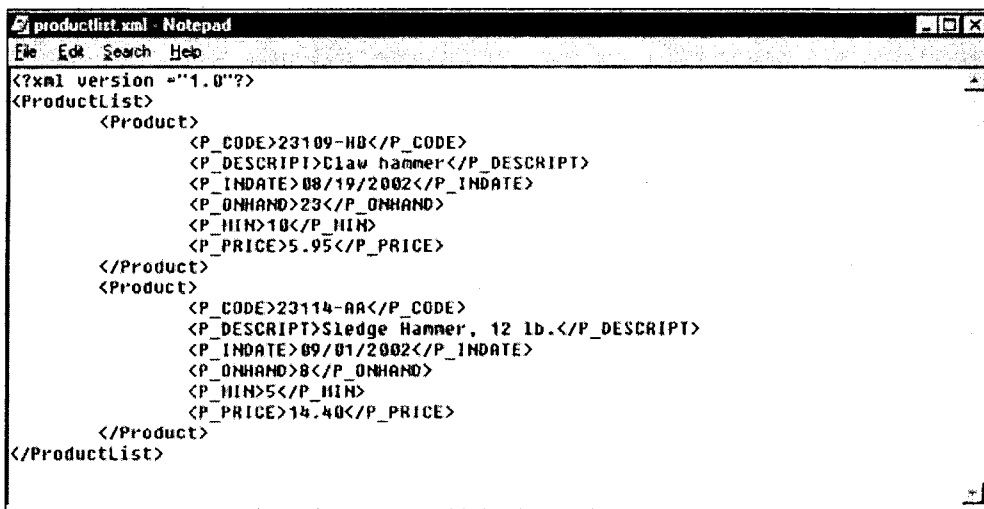


Рис. 14.5. Содержимое документа ProductList.xml

- корневой элемент содержит *дочерние элементы*, или *субэлементы*. В нашем примере в третьей строке объявляется элемент Product как дочерний элемент ProductList;
- каждый элемент может содержать субэлементы. Например, каждый элемент Product состоит из нескольких дочерних элементов P\_CODE, P\_DESCRIPTION, P\_INDATE, P\_ONHAND, P\_MIN и P\_PRICE;
- XML-документ отражает иерархическую древовидную структуру документа, где элементы соединены друг с другом связью родитель-потомок; каждый родительский элемент может иметь множество дочерних элементов. Например, корневой элемент — ProductList, Product — дочерний элемент ProductList. Product имеет шесть дочерних элементов P\_CODE, P\_DESCRIPTION, P\_INDATE, P\_ONHAND, P\_MIN и P\_PRICE.

Как только компания В получает документ ProductList.xml, она его обрабатывает в предположении, что понимает теги, используемые в компании А. Значение XML-тегов в примере, представленном на рис. 14.5, очевидно, но при этом нет простого способа проверить данные или полноту данных. Например, мы видим, что значение P\_INDATE равно "25/14/2001", но является ли это значение правильным? И что случится, если компания В ожидает еще и элемент Vendor? Как компании могут совместно использовать описание своих элементов данных? В следующем разделе показано, как для решения этих проблем используются определение типа документа и XML-схемы.

### 14.9.1. Определение типа документа (DTD) и XML-схемы

Решения B2B-стиля требуют высокой степени бизнес-интеграции компаний. Компании, использующие транзакции B2B-стиля, должны иметь способ понимать и прове-

рять любые другие теги. Один метод выполнения этой задачи состоит в использовании *определения типа документа (Document Type Definition, DTD)*. DTD представляет собой файл с расширением dtd, который описывает XML-элементы, — на самом деле, DTD-файл обеспечивает построение логической модели базы данных и определяет синтаксические правила или теги проверки для каждого типа XML-документов (DTD-компоненты очень похожи на общедоступный словарь для бизнеса). Компании, которые собираются заниматься е-коммерцией, должны разработать DTD и предоставить его в совместное использование. На рис. 14.6 представлено содержимое файла ProductList.dtd, соответствующего файлу ProductList.xml (см. рис. 14.5).

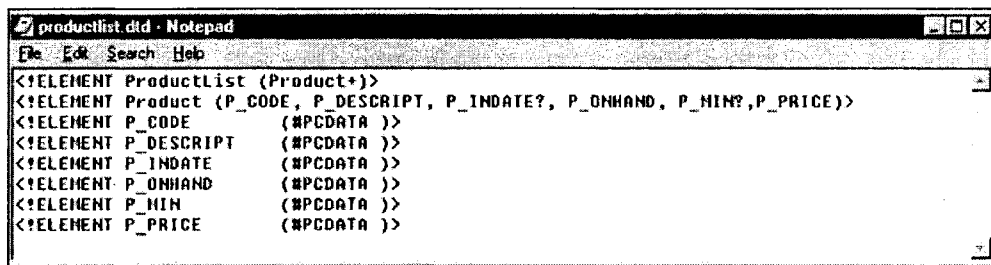


Рис. 14.6. Содержимое файла ProductList.dtd

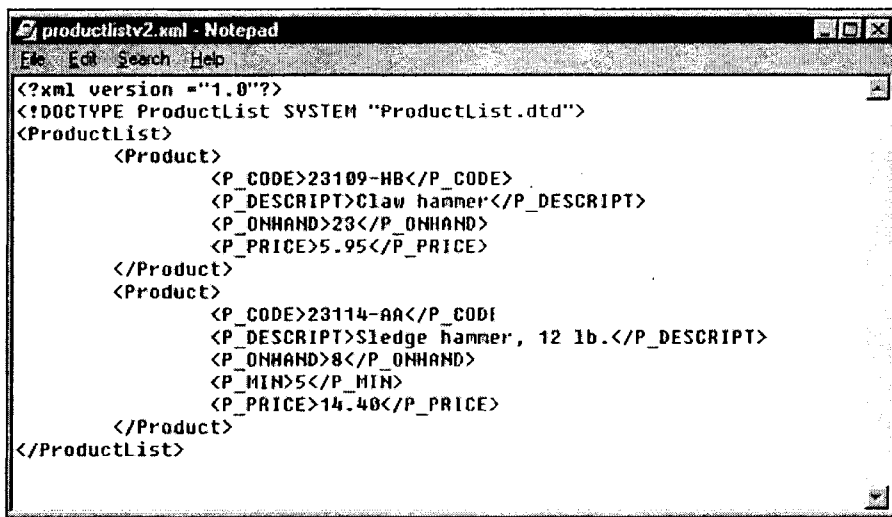
На рис. 14.6 обратите внимание, что файл ProductList.dtd предоставляет определения элементов в документе ProductList.xml. Особенно обратите внимание на следующее:

- ☐ первая строка декларирует корневой элемент ProductList;
- ☐ корневой элемент ProductList имеет один дочерний элемент Product;
- ☐ символ "+" указывает, что Product встречается один или более раз внутри элемента ProductList;
  - символ звездочка "\*" означает, что дочерний элемент встречается 0 или более раз;
  - знак вопроса "?" означает, что дочерний элемент необязателен;
- ☐ вторая строка описывает элемент Product;
- ☐ знак вопроса "?" после P\_INDATE и P\_MIN указывает, что это необязательные элементы;
- ☐ строки с третьей по восьмую показывают, что элемент Product имеет шесть дочерних элементов;
- ☐ ключевое слово #PCDATA представляет реальные текстовые данные.

Чтобы использовать DTD-файл для определения элементов внутри XML-документа, на DTD необходимо сослаться изнутри XML-документа. На рис. 14.7 представлен документ ProductList2.xml, в который включена ссылка на документ ProductList.dtd (во второй строке).

При исследовании рис. 14.7 обратите внимание, что элементы P\_INDATE и P\_MIN не появляются во всех определениях, поскольку они объявлены как необязательные. На документ DTD могут ссылаться много XML-документов одного и того же типа.

Например, если компания А регулярно обменивается данными о товарах с компанией В, необходимо создать DTD-документ только один раз. Все последующие XML-документы будут ссылаться на этот DTD-документ, и компания В сможет проверять полученные данные.



**Рис. 14.7.** Содержимое документа ProductListv2.xml

Для дальнейшей демонстрации использования XML и DTD для обмена данными в e-коммерции представим себе две компании, которые обмениваются данными о заказах. На рис. 14.8 представлены DTD и XML-документы для такого сценария.

Хотя использование DTD является значительным шагом вперед в совместном использовании данных с помощью Web, DTD обеспечивает только описательную информацию, позволяющую понять, как элементы — корневые, родительские, дочерние, обязательные или необязательные — связываются друг с другом. Документ DTD предоставляет ограниченную дополнительную семантику, например, поддержку типов данных или правила проверки данных. Такая информация очень важна для администраторов баз данных, отвечающих за большие базы данных e-коммерции. Для решения этой проблемы в мае 2001 года консорциум W3C опубликовал стандарт XML schema, обеспечивающий наилучший способ описания XML-данных.

*XML schema* это усовершенствованный язык определения данных, используемый для описания структуры (элементы, типы данных, типы связей, диапазоны и значения по умолчанию) документа XML-данных. Одно из главных достоинств языка XML schema еще и в том, что он наиболее точно отражает терминологию баз данных. Например, XML Schema (или XML-схема) позволяет определить общие типы баз данных, такие как даты, целые или вещественные, минимальные и максимальные значения, список допустимых значений и необходимые элементы. Используя XML-схему, компании могут выявлять значения данных, которые могут находиться

вне заданного диапазона, некорректные данные, допустимые значения и т. д. Например, университетское приложение должно позволять определить, что значение GPA (средний балл) должно находиться в диапазоне от 0 до 4 и выявить, что значение дня рождения "14.13.2001" является неверным. Многие поставщики быстро приняли этот новый стандарт и разработали специальные средства, транслирующие DTD-документы в документы XML Schema Definition (XSD). Ожидается, что XML-схемы заменят DTD в качестве средства описания XML-данных.

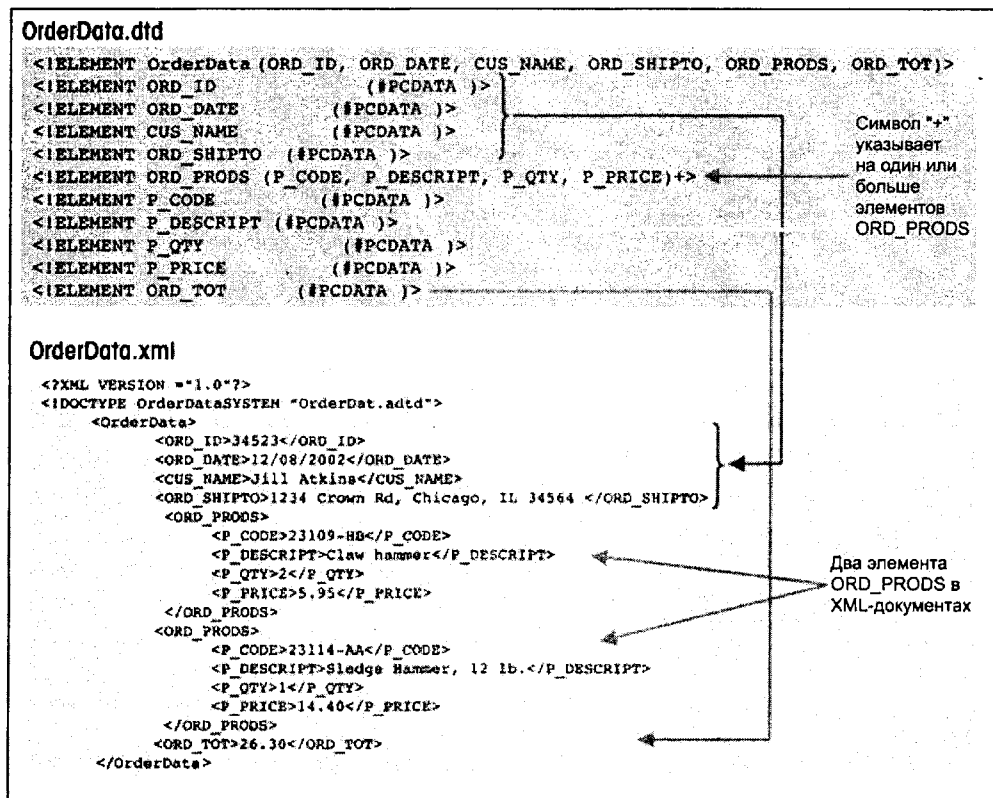


Рис. 14.8. Документы DTD и XML для данных заказа

В отличие от DTD-документа, в котором используется уникальный синтаксис, файл *XML Schema Definition (XSD)* использует синтаксис, который напоминает XML-документ. На рис. 14.9 представлен документ XML Schema Definition для нашего XML-документа по данным заказа.

Код, показанный на рис. 14.9, представляет собой упрощенную версию документа XML-схемы. Как видите, синтаксис XML-схемы очень похож на синтаксис XML-документа. Кроме того, XML-схема представляет дополнительную семантическую информацию для XML-документа по данным заказа, например, строки, даты и чис-



ленные типы данных, обязательные элементы, максимальные и минимальные значения для элементов данных.

### OrderData.xsd

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >

 <xsd:element name="OrderData" type="Order"/>

 <xsd:complexType name="Order">
 <xsd:sequence base="empty">
 <xsd:element name="ORD_ID" type="xsd:string"/>
 <xsd:element name="ORD_DATE" type="xsd:date"/>
 <xsd:element name="CUS_NAME" type="xsd:string"/>
 <xsd:element name="ORD_SHIPTO" type="xsd:string"/>
 <xsd:element name="ORD_PRODS" type="productlist"/>
 <xsd:element name="ORD_TOT" type="xsd:decimal"/>
 </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="productlist">
 <xsd:sequence base="empty">
 <xsd:element name="product" type="aproduct" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="aproduct">
 <xsd:sequence base="empty">
 <xsd:element name="P_CODE" type="xsd:string" use="required"/>
 <xsd:element name="P_DESCRIPTOR" type="xsd:string" use="required"/>
 <xsd:element name="P_QTY" type="xsd:positiveInteger" use="required"/>
 <xsd:element name="P_PRICE" type="xsd:decimal" use="required"/>
 </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

Рис. 14.9. Документ XML Schema Definition для данных заказа

## 14.9.2. XML-представление

Одно из основных достоинств XML — отделение структуры данных от представления и обработки. Разделяя данные и их представление, можно представить одни и те же данные различными способами — что очень напоминает представления в SQL. Но какой механизм используется для представления данных?

Спецификация Extensible Stylesheet Language предоставляет механизм для отображения XML-данных. *Extensible Style Language (XSL*, расширяемый язык стилей) представляет собой спецификацию, задающую правила, с помощью которых XML-данные форматируются и отображаются. XSL-спецификация состоит из двух частей.

- ❑ *Extensible Style Language Transformation (XSLT*, расширяемый язык стилей для преобразований) описывает основной механизм, используемый для обработки и извлечения данных из одного XML-документа и трансформации их в другой документ. С помощью XSLT мы можем извлекать данные из XML-документа и конвертировать их в текстовый файл, Web-страницу HTML или Web-страницу, форматированную для мобильного устройства. То, что пользователь увидит в

этих случаях, это фактическое представление, или HTML-представление фактических XML-данных. XSLT может также использоваться для извлечения некоторых элементов из XML-документа, например, кодов товара и стоимости товара для создания каталога товаров. Мы можем даже использовать XSLT для преобразования XML-документа в другой XML-документ.

- ❑ *Таблицы стилей XSL* определяют правила представления, применяемые к XML-элементам, когда они отображаются в окне обозревателя, на экране мобильного телефона, экране портативного компьютера и т. д.

На рис. 14.10 представлена структура, используемая различными компонентами для преобразования XML-документов в наглядные Web-страницы, XML-документ или какой-нибудь другой документ.

Современные обозреватели пока ограниченно поддерживают XML, XSLT и XSL — последняя версия Microsoft Internet Explorer (MSIE) поддерживает только некоторые стандарты XML, а Netscape Navigator (версии 4.7 или более ранней) не поддерживает XML. В следующих разделах будет продемонстрирована поддержка XML обозревателем MSIE (версии 5.0 или более ранней).

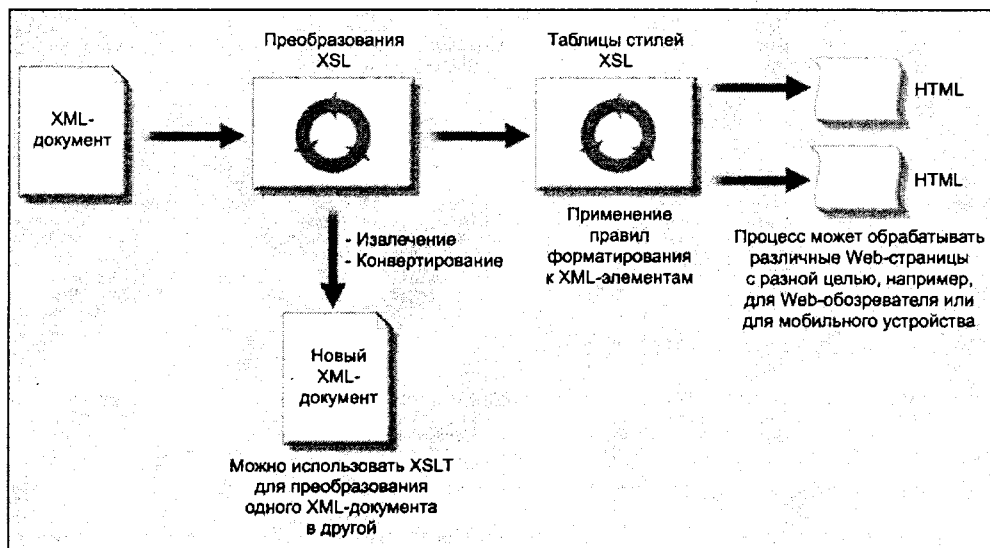


Рис. 14.10. Структура для XML-преобразования

Для отображения XML-документа с помощью MSIE 5.0 просто введите URL-адрес XML-документа в адресной строке обозревателя. Рис. 14.11 основан на документе ProductList.xml, созданном ранее. На рис. 14.11 обратите внимание, что MSIE представляет XML-данные в виде сворачиваемой древовидной структуры с цветовыми выделениями (фактически это таблица стилей MSIE по умолчанию, используемая для представления XML-документов).

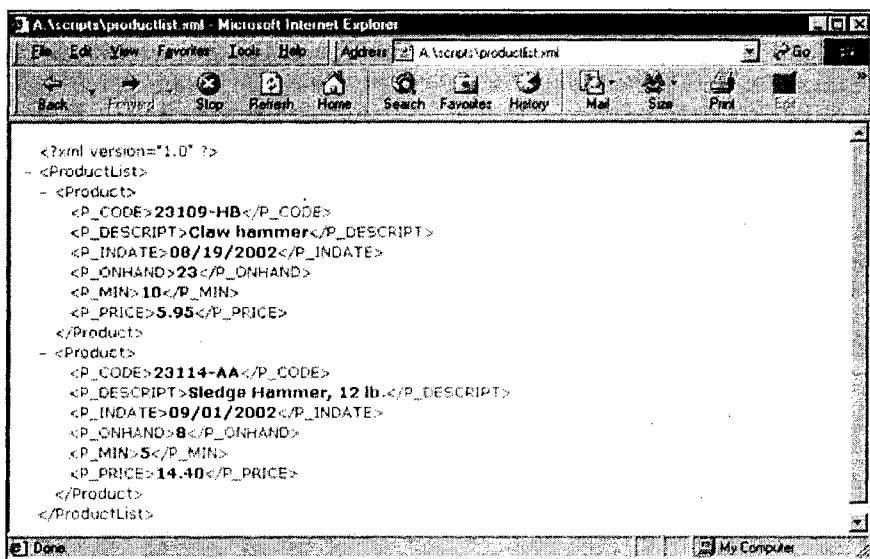


Рис. 14.11. Отображение XML-документа в MSIE 5.0

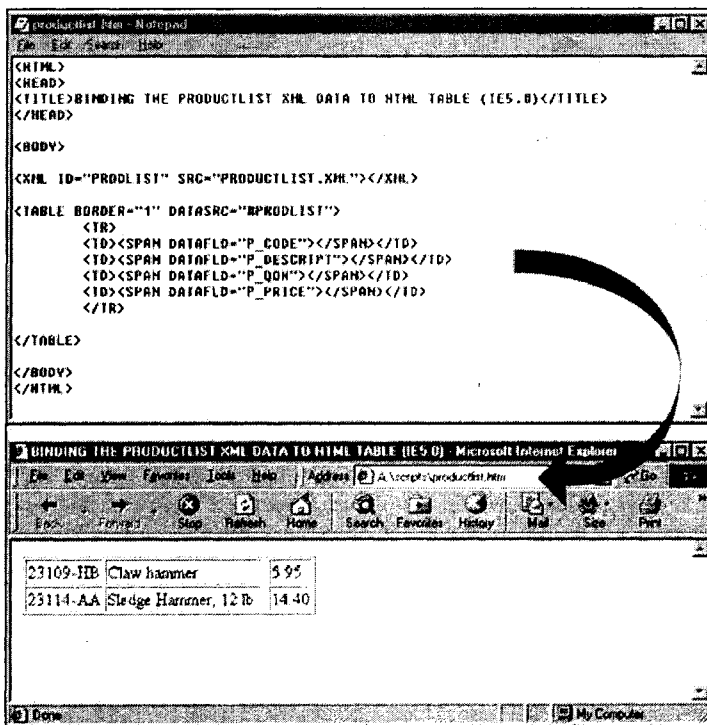


Рис. 14.12. Связывание XML-данных в MSIE 5.0

MSIE 5.0 также обеспечивает *связывание данных (data binding)* для HTML-документов. На рис. 14.12 представлен HTML-код, используемый для связывания XML-документа и HTML-таблицы. Этот пример работает только в MSIE 5.0 или более ранней версии.

### 14.9.3. XML-приложения

Теперь, когда вы имеете определенное представление об XML, возникает следующий вопрос: какого рода приложения лучше всего представляются с помощью XML? В этом разделе мы перечислим некоторые применения XML. Нужно иметь в виду, что использование XML в будущем ограничивается только воображением и творческой фантазией разработчиков, проектировщиков и программистов.

- ❑ *Обмены стиля B2B.* Как уже отмечалось ранее, XML допускает обмен данными стиля B2B, обеспечивая стандарт для использования всеми организациями, которым необходим обмен данными с партнерами, конкурентами, правительственными учреждениями или клиентами. В частности, XML позиционируется как замена EDI<sup>4</sup> в качестве стандарта автоматизации документооборота, поскольку этот язык дешевле и к тому же достаточно гибок.
- ❑ *Интеграция с действующими системами.* XML представляет собой некий "клей" для соединения действующих систем с современными Web-системами e-коммерции. Иначе говоря, Web- и XML-технологии должны использоваться для того, чтобы вдохнуть жизнь в старые, но надежные приложения. Другой пример — использование XML для импорта данных транзакций из нескольких операционных баз данных в базу данных хранилища.
- ❑ *Разработка Web-страниц.* XML предоставляет средства, делающие этот язык удобным для разработки Web-сценариев. Например, Web-порталы с большим объемом персонифицированных данных могут использовать XML для получения данных из множества внешних источников (таких как новости, погода и курс акций) и применения различных правил представления для форматирования страниц на экране компьютера, а также на мобильных устройствах.
- ❑ *Поддержка баз данных.* Базы данных — сердце приложений e-коммерции. СУБД, поддерживающие XML-обмен, могут интегрироваться с внешними системами (Web, мобильные данные, действующие системы и т. д.), таким образом допуская создание новых типов систем. Фирмы IBM, Oracle и Microsoft уже предлагают продукты баз данных, поддерживающие XML. Такие базы данных могут импортировать данные в XML-формате или создавать XML-документы из SQL-запросов и т. д., в то же время позволяя хранить XML-данные в естественном формате. У этих возможностей большое будущее — можно даже сохранить иерархическую древовидную структуру в реляционной структуре. Конечно, такие действия требуют расширения языка запросов до поддержки запросов XML-данных.
- ❑ *Метасловари баз данных.* XML можно использовать для создания метасловарей, или лексиконов баз данных. Такие метасловари могут использоваться приложе-

---

<sup>4</sup> Electronic Data Interchange — электронный обмен данными (общее название группы стандартов безбумажной технологии). — *Прим. пер.*

ниями, которым необходимо получать доступ к другим внешним источникам данных. (До сих пор каждый раз, когда приложение обменивается данными с другим приложением, необходимо специально создавать для этого новый интерфейс.) Поставщики СУБД могут публиковать метасловари для упрощения обмена данными и создания представления данных из нескольких приложений — иерархических, реляционных, объектно-ориентированных, объектно-реляционных и расширенных реляционных. В метасловарях будет использоваться общий язык, независимо от типа СУБД. Предполагается создание специфических отраслевых метасловарей для обеспечения разработки сложных взаимодействий B2B, которые могут найти применение в авиации, автомобильной и фармацевтической промышленности. Также ожидают метасловари для приложений. Например, предполагается создать XML метасловари для организации хранилищ данных, приложений по управлению системами и сложных статистических приложений. Даже ООН и некоммерческая организация по разработке стандартов Oasis работают над новой спецификацией, называемой *ebXML*, которая создаст стандартный XML-словарь для e-бизнеса.

- *Базы данных XML*<sup>5</sup>. Учитывая огромный объем обмена XML-данными, бизнес уже ищет пути для лучшего управления и использования таких данных. В настоящее время на рынке есть множество различных продуктов, имеющих отношение к этой проблеме, от простого промежуточного программного обеспечения XML для объектных баз данных с XML-интерфейсом до устройств (engines) и серверов баз данных, полностью основанных на XML. Современное поколение реляционных баз данных настроено на хранение нормализованных строк, т. е. на манипулирование одной строкой данных за один раз. Мы знаем, что деловая информация не всегда соответствует этому требованию. Базы данных XML обеспечивают возможность хранения данных, имеющих сложные связи. Например, базу данных XML удобно использовать для хранения содержимого книги (структура книги определяет структуру базы данных: книга как правило содержит главы, разделы, абзацы, рисунки, диаграммы, колонтитулы, сноски, примечания и т. д.). Примерами баз данных XML можно считать dbXML компании dbXML Group ([www.dbxml.org](http://www.dbxml.org)), Ipedo XML Database от компании Ipedo ([www.ipedo.com](http://www.ipedo.com)) и Tamino компании Software AG ([www.softwareag.com](http://www.softwareag.com)).
- *XML-сервисы*. Многие компании уже разрабатывают новое поколение сервисов на базе XML- и Web-технологий. Эти сервисы обещают преодолеть проблемы взаимодействия операционных систем, а также организационные барьеры компаний. В письме, датированном 14 июля 2001 года, Билла Гейтса (Bill Gates), руководителя компании Microsoft, спросили: "Как будет выглядеть следующее поколение Интернета?" Ответом стала .NET — новая платформа разработки XML Web-сервисов для интеграции бизнеса способами, до этого неосуществимыми. XML предоставляет инфраструктуру, облегчающую совместную работу разнородных систем на Рабочем столе, предприятии и по всему миру. Сервисы должны использовать XML и другие интернет-технологии для публикации своих интерфейсов. Другие сервисы, которым необходимо взаимодействовать с имеющимися сервисами, будут находить такие службы и изучать их словари (сервисы запросов и ответов) для установления "разговора".

<sup>5</sup> Для более полного анализа продуктов баз данных XML см. "XML Database Products", Ronald Bourret на сайте [www.rpbouret.com](http://www.rpbouret.com).

## Резюме

### **Дополнительная информация**

Расширенная информация об истории е-коммерции, ее компонентах и ее применении представлена на сайте [www.course.com](http://www.course.com) (поиск по 0-619-06269-X).

Е-коммерцией называют использование электронных компьютерных технологий (в частности, Интернета) для представления на рынке новых товаров, услуг или идей и для поддержки расширенных бизнес-операций. Интернет оказывает очень большое влияние на то, как компании осуществляют свой бизнес. Интернет также оказывает существенное влияние на некоммерческие организации и правительственные учреждения.

Е-коммерция позволяет компаниям рекламировать и продавать товары и/или услуги миллионам пользователей на всемирном рынке. Компании, заинтересованные в е-коммерции, используют Интернет для интеграции своих систем с системами других компаний, чтобы повысить эффективность работы и снизить затраты.

Использование технологий е-коммерции позволяет компаниям быстро отвечать на вызовы конкурентов, расширяя внутренние операции и облегчая бизнес-транзакции. Некоторые из достоинств е-коммерции для клиентов включают: возможность сравнивать товар в разных магазинах, снижение цен, улучшенное восприятие, круглосуточная работа интернет-магазинов без выходных и удобный интерфейс. Преимущества для бизнеса состоят в следующем: выход на глобальный рынок с миллионами клиентов, простота выхода на рынок, оперативная информация о состоянии рынка. Однако е-коммерция имеет и определенные недостатки как для клиентов, так и для бизнеса — например, скрытые затраты, недостаточная надежность сетей, недостаточная секретность, низкий уровень обслуживания и юридические проблемы, такие как нарушение прав собственности и мошенничество.

Стиль е-коммерции может классифицироваться как бизнес-бизнес (B2B), бизнес-потребитель (B2C) и внутренний бизнес. Стиль B2B е-коммерции связан со всеми транзакциями, которые проводятся между коммерческими предприятиями, например, между предприятиями и поставщиками, предприятиями и оптовиками, оптовиками и розничными торговцами, а также розничными торговцами и службами доставки. Стиль B2C е-коммерции связан со всеми транзакциями, которые имеют место между предприятиями розничной торговли и клиентами. Е-коммерция внутреннего бизнеса связана с бизнес-процессами, проходящими внутри организации.

Архитектура е-коммерции представляет собой структуру, описывающую технические компоненты Web-сайтов е-коммерции и их взаимодействие. Архитектура разделяется на три отдельных взаимодействующих уровня служб и приложений. Нижний уровень (базовые службы Интернета) предоставляет "систему каналов", необходимую для передачи данных между компьютерами. Средний уровень (службы обеспечения бизнеса) обеспечивает дополнительные службы для поддержки бизнес-транзакций. Эти службы добавляют средства обеспечения безопасности, надежности, целостности и эффективности, необходимые для работы в среде Интернета. Верхний уровень архитектуры (службы обеспечения е-коммерции) использует нижние уровни для отображения бизнес-логики так, чтобы представлять и автоматизировать бизнес-процессы.

Безопасность и секретность бизнес-транзакций и данных очень важны для успешной е-коммерции. От начала транзакции до ее окончания должна обеспечиваться безопасность данных е-коммерции, а также хранение сведений о транзакции и ее результатов.

Обработка кредитных карт и электронные деньги — наиболее общий способ обработки платежей в Интернете. Электронные бумажники позволяют пользователю хранить персональные сведения, информацию о кредитных картах, электронных деньгах в безопасном месте и впоследствии использовать эти данные для проведения онлайн-платежей.

Базы данных е-коммерции содержат нормализованные структуры, похожие на обычные операционные базы данных. В этой главе была разработана основа для структуры базы данных е-коммерции.

Расширяемый язык разметки (XML) облегчает обмен B2B-данными через Интернет. XML обеспечивает семантику и облегчает обмен, совместное использование и манипулирование структурированными документами за пределами организаций. С помощью XML компании любого рода могут обмениваться данными, содержащимися в XML-документах. XML позволяет выполнять описание и представление данных, устанавливая такой уровень манипуляции данными, который до этого был просто невозможен. XML-документы могут проверяться с помощью документов Document Type Definition (DTD) и XML Schema Definition (XSD). Использование DTD, XML schema и XML-документов позволяет перейти на более высокий уровень интеграции различных систем, чем это было возможно ранее. XML влияет на обмен данными B2B-стиля, интеграцию существующих систем, разработку Web-страниц, взаимодействие систем баз данных, а также на хранение и управление документами.

## Основные термины

Web-кэширование — Web caching

Web-разработка — Web development

Web-сайт — Web site

Web-сервер — Web server

Web-страница — Web page

XML-схема — XML Schema

Безопасность — security

Бизнес-бизнес — business to business (B2B)

Бизнес-клиент — business to consumer (B2C)

Обозреватель, Web-обозреватель, браузер, Web-браузер — Web browser

Внутренний бизнес — intra-business

Выравнивание нагрузки — load balancing

Гиперссылка — hyperlink

Глобальная Сеть — World Wide Web, WWW или Web

- Динамическая Web-страница — dynamic Web page
- Защита электронных платежей — Secure Electronic Transaction, SET
- Интеграция баз данных — database integration
- Интернет — Internet
- Интранет — intranet
- Консорциум World Wide Web — World Wide Web Consortium (W3C)
- Маршрутизатор — router
- Наблюдение за сайтом и анализ данных — site monitoring and data analysis
- Обмен сообщениями — messaging
- Обработка транзакций — transaction processing
- Определение XML-схем — XML Schema Definition (XSD)
- Определение типа документа — Document Type Definition (DTD)
- Персонализация — personalization
- Поддержка беспроводных устройств — wireless device support
- Поисковая служба — search services
- Протокол TCP/IP
- Протокол передачи гипертекста — Hypertext Transfer Protocol (HTTP)
- Протокол передачи файлов — File Transfer Protocol (FTP)
- Расширяемый язык разметки — eXtensible Markup Language (XML)
- Расширяемый язык стилей для преобразований — Extensible Style Language Transformation (XSLT)
- Расширяемый язык стилей — Extensible Style Language (XSL)
- Секретность — privacy
- Сеть, глобальная сеть — Web
- Службы новостей и дискуссионных групп — news and discussing group services
- Стандартный обобщенный язык разметки — Standard Generalized Markup Language (SGML)
- Статичная Web-страница — static Web page
- Таблицы стилей XSL — XSL style sheets
- Унифицированный указатель информационных ресурсов, URL-адрес — Uniform Resource Locator (URL)
- Управление содержимым — content management
- Цифровой сертификат — digital certificate
- Эксплуатационные испытания — load testing
- Экстранет — extranet
- Электронная коммерция, е-коммерция — electronic commerce, e-commerce



Электронная почта — electronic mail, e-mail

Электронные деньги — digital cash

Электронный бумажник — electronic wallet

Язык разметки гипертекста — Hypertext Markup Language (HTML)

## Вопросы

1. Что такое е-коммерция и какова история ее развития?
2. Определите и кратко поясните пять преимуществ и пять недостатков е-коммерции.
3. Определите и сопоставьте стили е-коммерции B2B и B2C.
4. Опишите и дайте пример каждой из двух основных форм стиля B2B.
5. Опишите архитектуру е-коммерции, затем дайте краткое описание каждого из ее компонентов.
6. Какие типы служб представлены на нижнем уровне архитектуры е-коммерции?
7. Назовите и поясните действия основных функциональных блоков Интернета и его основных служб.
8. Что такое обеспечение бизнеса? Какие службы обеспечивают бизнес? Приведите шесть примеров таких служб.
9. Дайте определение безопасности. Поясните, почему безопасность так важна для транзакций е-коммерции.
10. Приведите пример сценария транзакции е-коммерции. Какова основная цель обеспечения безопасности в этой транзакции е-коммерции?
11. Вас приняли на работу в качестве офицера по защите информации в компанию, занимающуюся е-коммерцией. Кратко поясните, какие технические проблемы вы должны упомянуть в своем рабочем плане обеспечения безопасности.
12. Что такое язык XML и каковы его основные достоинства?
13. Что такое Document Type Definition (DTD) и каково его предназначение?
14. Что собой представляют документы XML Schema Definition (XSD) и каково их предназначение?

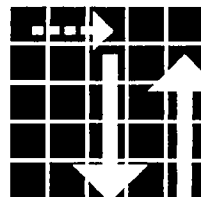
## Задачи

1. Используйте Интернет в вашей университетской компьютерной лаборатории или дома для исследования сценариев, описанных в задачах 1—10, затем решите следующие задачи:
  - какие Web-сайты вы посетили?
  - классифицируйте каждый сайт (B2B, B2C и т. д.);
  - какую информацию вы накопили? Чем полезна эта информация? Почему?
  - какое решение вы приняли на основе собранной вами информации?

2. Исследуйте (и запишите) процесс покупки нового автомобиля? Поясните, почему вы планируете купить именно этот автомобиль на основе проведенных вами исследований.
3. Исследуйте (и запишите) процесс покупки нового дома.
4. Вы ищете на рынке новую работу. Просмотрите Сеть в поисках идеальной работы для вас. Запишите процесс поиска работы и мотивы вашего выбора.
5. Вам необходимо рассчитать налоги. Скачайте форму 1040 расчета налогообложения IRS и исследуйте правила расчета налогообложения. Запишите ваши действия.
6. Исследуйте процесс приобретения 20-летнего страхового полиса и запишите ваши наблюдения.
7. Исследуйте (и запишите) процесс покупки нового компьютера.
8. Отпуск! Исследуйте (и запишите) процесс поиска места вашего отдыха будущим летом.
9. Вы хотите инвестировать некоторую сумму. Исследуйте (и запишите) информацию о фондах, предоставляемых на основе обоюдных обязательств и предназначенных для инвестиций. Создайте отчет о вашем инвестиционном решении на основе проведенных вами исследований.
10. Создайте пример XML-документа и DTD для обмена:
  - данными о клиенте;
  - данными о товаре и ценах;
  - данными о заказе;
  - данными о специальных предложениях;
  - данными о наблюдении за ценой;
  - данными о студентах.

(Совет: используйте материал *разд. 4.9.*)

## Глава 15



# Разработка баз данных для Web

В этой главе мы изучим:

- как используются базы данных Интернета;
- архитектуру промежуточного программного обеспечения (ППО) баз данных для Web (Web-БД);
- как используется промежуточное программное обеспечение баз данных для Web при интеграции баз данных с Интернетом;
- специфику разработки баз данных для Web.

## Обзор

В *гл. 14* мы рассмотрели основные функциональные блоки Интернета. Мы также обсудили механизмы, необходимые для обеспечения безопасности инфраструктуры е-коммерции при представлении товаров и услуг на мировом рынке пользователей Интернета. Кроме того, мы изучали некоторые важные проблемы, оказывающие влияние на проектирование и разработку баз данных в среде е-коммерции.

База данных — центральный склад информации для коммерческих транзакций. Такие транзакции могут создаваться по традиционным каналам бизнеса или с помощью Интернета. В этой главе мы рассмотрим способы подключения базы данных к глобальной Сети (Web).

Учитывая возрастающую взаимосвязь Web и баз данных, профессиональные разработчики БД должны знать, как создавать, использовать и управлять Web-интерфейсами таких баз данных. В этой главе представлены основы разработки баз данных для Web. В соответствии с практическим подходом, принятым в этой книге, вам будет предложено испытать на деле ColdFusion — инструментальный сервер Web-приложений, предназначенный для создания интерфейсов баз данных для Web.

## 15.1. Интернет-технологии и базы данных

Интернет предоставляет всеобщий и универсальный интерфейс доступа к информации — Web-обозреватель (Web-браузер), который прост в использовании и может

выполняться на различных платформах. Способность баз данных для Web (Web-БД) к взаимодействию обеспечивает доступ к самым передовым службам, позволяющим:

- ☐ немедленно реагировать на возрастающую конкуренцию, оперативно поставляя товары и услуги на рынок;
- ☐ удовлетворять требования пользователя путем создания служб, основанных на интернет-технологиях;
- ☐ быстро и эффективно распространять информацию посредством универсального доступа к локальным и глобальным ресурсам.

Принимая во внимание преимущества Web-технологий, многие информационные службы стоят перед необходимостью создания архитектуры универсального доступа к данным на основе стандартов Интернета, чтобы ускорить выполнение операций и облегчить принятие решений. В табл. 15.1 представлены основные характеристики интернет-технологий и преимущества их использования.

**Таблица 15.1. Преимущества использования интернет-технологий**

| Характеристика                                          | Преимущества                                                                                     |
|---------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| Независимость от аппаратного и программного обеспечения | Экономия средств на приобретение аппаратно/программного обеспечения                              |
|                                                         | Работа на большинстве имеющихся конфигураций                                                     |
|                                                         | Независимость от платформы и переносимость                                                       |
| Простой и удобный интерфейс пользователя                | Отсутствие необходимости разработок для нескольких платформ                                      |
|                                                         | Сокращение времени и экономия средств на обучении                                                |
|                                                         | Экономия средств на поддержке конечных пользователей                                             |
| Независимость от местоположения                         | Отсутствие необходимости разработок для нескольких платформ                                      |
|                                                         | Глобальный доступ с помощью инфраструктуры Интернета                                             |
|                                                         | Отсутствие необходимости в организации специализированных подключений                            |
| Быстрая разработка и контролируемые цены                | Доступность множества средств разработки                                                         |
|                                                         | Разработка инструментария по открытому стандарту технологии plug-and-play ("подключи и работай") |
|                                                         | Большая интерактивность разработки                                                               |
|                                                         | Сокращение времени на разработку                                                                 |
|                                                         | Относительно низкая стоимость разработки                                                         |
|                                                         | Бесплатные средства доступа клиентов (Web-обозреватели)                                          |
|                                                         | Невысокая входная стоимость — бесплатные Web-серверы и неограниченный доступ                     |
|                                                         | Сокращение затрат на обслуживание частных вычислительных сетей                                   |
|                                                         | Распределенная обработка и масштабируемость на основе множества серверов                         |

Учитывая состояние инфраструктуры современного бизнеса и глобальной информационной среды, легко понять, почему многие профессиональные разработчики баз данных рассматривают "подключение" СУБД к Интернету как важнейший элемент информационных разработок. Как будет показано в следующих разделах этой главы, на разработку приложений баз данных и, в особенности, на создание пользовательских интерфейсов и управление ими, а также на обеспечение соединяемости баз данных основное влияние оказывает глобальная Сеть (Web). Однако Web-интерфейс базы данных не устраняет проблем разработки и реализации, рассмотренных в предыдущих главах. В конечном счете, делаете ли вы покупку в Интернете или отстояв очередь в магазине, на системном уровне детали транзакций одни и те же, и они требуют одних и тех же структур и связей от баз данных. Главный урок, который необходимо усвоить, — *урон от плохо спроектированной, неудачно реализованной и плохо управляемой базы данных многократно усиливается в среде, где транзакции исчисляются миллиардами, а не сотнями в день.*

## 15.2. Типичные проблемы баз данных Интернета

Интернет быстро меняет способы формирования информации, доступа к ней и ее распределения. Основа этих изменений — возможности Web по получению доступа (локального или удаленного) к базам данных, простой интерфейс и кросс-платформенные (гетерогенные) функциональные возможности. Web помогает создать новый стандарт распространения информации.

Хотя е-коммерция и ее укомплектованность средствами управления транзакциями и обеспечения безопасности растут по экспоненте, обычно использование баз данных в Интернете сводится к *совместному использованию информации, которую желательно обнаруживать*. Поскольку такая информация общедоступна, к ней могут получить доступ не только клиенты, но и конкуренты. Поэтому основу совместно используемой информации, по всей вероятности, должны составлять не критично важные данные. Например, в базе данных EDGAR (Electronic Data Gathering and Retrieval, электронная система сбора и поиска информации) Комитета по Безопасному Обмену Информацией (Security Exchange Commission) хранится финансовая информация о тысячах бизнесменов, доступная любому пользователю Интернета (<http://www.sec.gov/edaux/search.htm>). Хотя коммерческие предприятия могут использовать Интернет для проведения маркетинга и обеспечения поддержки своих клиентов, организуя доступ к поисковым техническим базам данных, каталогам товаров, контактной информации и т. д., интернет-технологии в настоящее время используются и для разработки систем, которые предоставляют корпоративную информацию пользователям компании (интранет), а также систем, делающих значимую информацию доступной для пользователей других компаний, с которыми данная компания имеет деловые отношения (экстранет). Например, компания А может использовать приложения экстранет, чтобы позволить компании В осуществлять автоматизированную поставку товаров для компании А, независимо от того достигнут или нет компанией А минимальный уровень наличия товара. Интернет-технология также используется в разработке как передовых бизнес-приложений, например, клиентских сервисов, так и приложений

второго плана, например, промышленных приложений. В табл. 15.2 представлена сводка основных приложений предприятия, основанных на технологиях Интернета.

**Таблица 15.2.** Пример приложений баз данных на основе Web-технологий

| Тип приложения                         | Внутренняя доступность | Внешняя доступность |
|----------------------------------------|------------------------|---------------------|
| <b>Основная информация предприятия</b> |                        |                     |
| Список телефонов                       | Да                     | Да                  |
| Книга предложений                      | Да                     | Да                  |
| <b>Производственная информация</b>     |                        |                     |
| Проект товара                          | Да                     | Нет                 |
| Стоимость материальных ценностей       | Да                     | Нет                 |
| Слежение за проектами                  | Да                     | Нет                 |
| <b>Информация о продажах</b>           |                        |                     |
| Каталог продукции                      | Да                     | Да                  |
| Заказы                                 | Да                     | Да                  |
| Поддержка клиентов                     | Да                     | Да                  |

Следующий раздел посвящен промежуточному программному обеспечению (ППО), обеспечивающему работу базы данных в среде Web, и интерфейсам, по которым пользователи взаимодействуют с базой данных.

### 15.3. Промежуточное программное обеспечение баз данных Web: серверные расширения

В гл. 14 мы рассматривали основы взаимодействия между клиентским Web-обозревателем и Web-сервером. Web-сервер, как правило, является основным звеном, через которое можно получить доступ ко всем службам Интернета. Например, когда конечный пользователь использует Web-обозреватель для динамических запросов к базе данных, обозреватель клиента фактически запрашивает Web-страницу. Когда Web-сервер получает запрос на страницу, он ищет эту страницу на жестком диске и, найдя ее, отправляет клиенту (например, курсы акций, каталог продукции или расписание самолетов).

Было показано, что динамические Web-страницы составляют основу Web-сайтов е-коммерции. При таком сценарии Web-сервер создает содержимое Web-страницы перед тем, как отослать страницу клиентскому Web-обозревателю. Здесь только одна проблема — Web-сервер должен включить результаты запроса к базе данных в страницу *перед* тем, как отослать ее клиенту. К сожалению, ни Web-обозреватель, ни

Web-сервер не знает, как подключиться к базе данных и считать оттуда информацию (помните, что Web-сервер и Web-обозреватель "понимают" только страницы в формате HTML). Поэтому для поддержки запросов к базе данных Web-сервер необходимо расширить до понимания и обработки запросов такого типа. Это выполняется с помощью серверных расширений.

*Серверное расширение (server-side extension)* это программа, напрямую взаимодействующая с Web-сервером для обработки запросов специфического вида. Программа — серверное расширение извлекает данные из базы данных и передает их Web-серверу, который, в свою очередь, отправляет их клиентскому обозревателю для отображения. Кроме того, что серверное расширение позволяет извлекать данные и представлять результаты запроса, оно предоставляет Web-серверу службы таким образом, что они *абсолютно прозрачны (невидимы) клиентскому обозревателю*. Иначе говоря, серверное расширение увеличивает функциональные возможности Web-сервера и, следовательно, Интернета в целом.

Программу серверного расширения баз данных называют еще *промежуточным программным обеспечением баз данных для Web* — ППО Web-БД (более подробно о промежуточном программном обеспечении рассказано в гл. 12). На рис. 15.1 представлено взаимодействие между обозревателем, Web-сервером и ППО Web-БД (в данном случае ColdFusion).

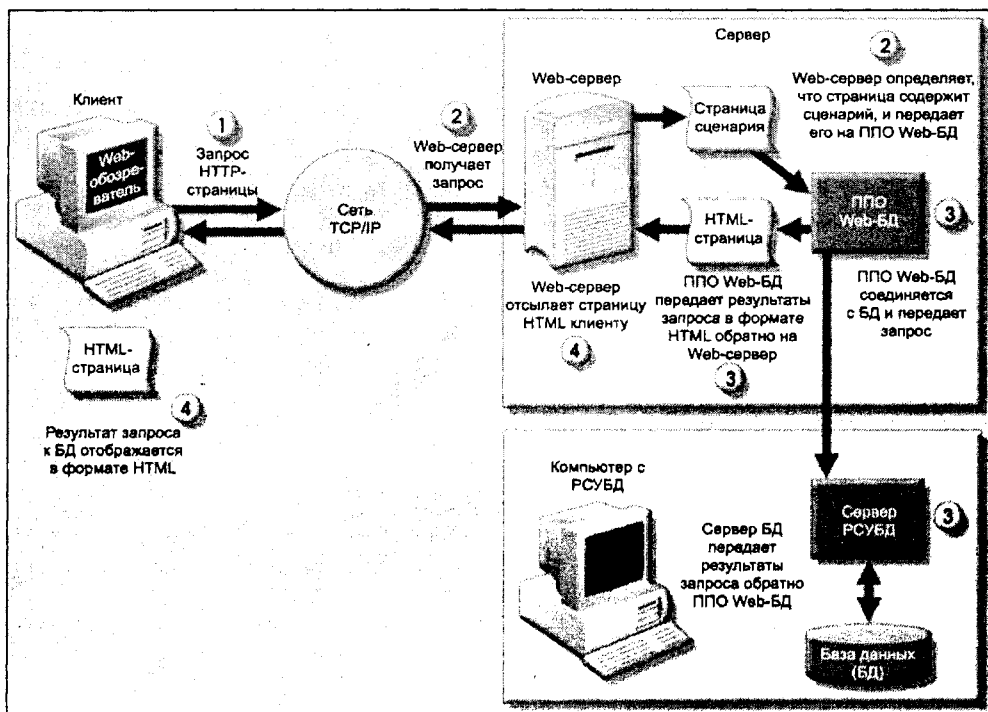


Рис. 15.1. Промежуточное программное обеспечение баз данных Web, ППО Web-БД (ColdFusion)

Проследим работу ППО Web-БД по рис. 15.1.

1. Клиентский обозреватель посылает запрос на страницу Web-серверу.
2. Web-сервер получает и проверяет запрос. В данном случае сервер передаст запрос на ППО Web-БД для обработки. Как правило, на запрашиваемой странице содержатся некоторые программы, написанные на одном из языков сценариев, позволяющих взаимодействовать с базой данных.
3. ППО Web-БД считывает, проверяет и выполняет сценарий. В данном случае происходит подключение к базе данных, запрашивается БД, динамически создается страница в формате HTML, в которую включаются данные, извлеченные из БД, и затем отсылается Web-серверу.
4. Web-сервер возвращает только что созданную HTML-страницу (в которую теперь включены результаты запросов) клиентскому обозревателю для отображения.

Очевидно, взаимодействие между Web-сервером и ППО Web-БД является ключевым моментом для разработки и успешной реализации базы данных Интернета. Поэтому ППО должно быть эффективно интегрировано с другими службами Интернета и компонентами, необходимыми для его работы. Например, при установке ППО Web-БД необходимо проверить тип используемого Web-сервера и установить ППО в соответствии с требованиями Web-сервера. Кроме того, взаимодействие Web-сервера и службы ППО Web-БД будет зависеть от интерфейсов, поддерживаемых Web-сервером.

### 15.3.1. Интерфейсы Web-сервера

Расширение функциональных возможностей Web-сервера предполагает, что Web-сервер и ППО Web-БД надлежащим образом связываются друг с другом (профессиональные проектировщики БД обычно используют термин "взаимодействие" — "interoperate" для указания на то, что каждая сторона соединения отвечает за связь с другой. В нашем случае слово связь — communicate — также предполагает взаимодействие). Если Web-сервер успешно связывается с внешней программой, то обе программы должны использовать стандартный способ обмена сообщениями и ответов на запросы. Способ связи Web-сервера с внешними программами определяется интерфейсом Web-сервера. В настоящее время имеются два хорошо проработанных интерфейса Web-сервера:

- Common Gateway Interface (CGI);
- Application Programming Interface (API).

*Common Gateway Interface* (CGI, *общий иллюзовый интерфейс*) использует файлы сценариев, передаваемых на Web-сервер. Сценарий представляет собой небольшую программу, содержащую команды языка программирования, обычно PERL, C++ или Visual Basic. Содержимое файла сценария может использоваться для связи с базой данных и извлечения из нее данных с помощью параметров, передаваемых Web-сервером. Затем сценарий конвертирует извлеченные данные в формат HTML и передает их на Web-сервер, который посылает HTML-страницу клиенту.

Основной недостаток использования CGI-сценариев в том, что сценарий является внешней программой, выполняемой индивидуально для каждого запроса пользователя, что ухудшает производительность системы в целом. Например, если имеется



200 одновременных запросов, сценарий загружается 200 раз, что влечет за собой интенсивное использование ресурсов Web-сервера. Язык и методы, используемые при создании сценария, тоже влияют на производительность системы. Например, производительность уменьшается при использовании интерпретируемых языков или при неудачно написанном сценарии.

*Application Programming Interface* (API, *интерфейс прикладного программирования*) это новейший стандарт интерфейса Web-сервера, более эффективный, чем сценарии CGI. API более эффективен потому, что он реализован в виде совместно используемого кода или *динамически подключаемых библиотек* (*dynamic-link library, DLL*). Это означает, что API представляет собой часть программы Web-сервера, которая динамически иницируется при необходимости.

Интерфейсы API работают быстрее, чем CGI-сценарии, поскольку код находится в памяти, и поэтому нет необходимости выполнять внешнюю программу для каждого запроса. Вместо этого все запросы обслуживает один и тот же интерфейс API. Другое преимущество состоит в том, что API может использовать общее подключение к базе данных вместо создания нового подключения каждый раз, как это происходит при использовании CGI-сценариев.

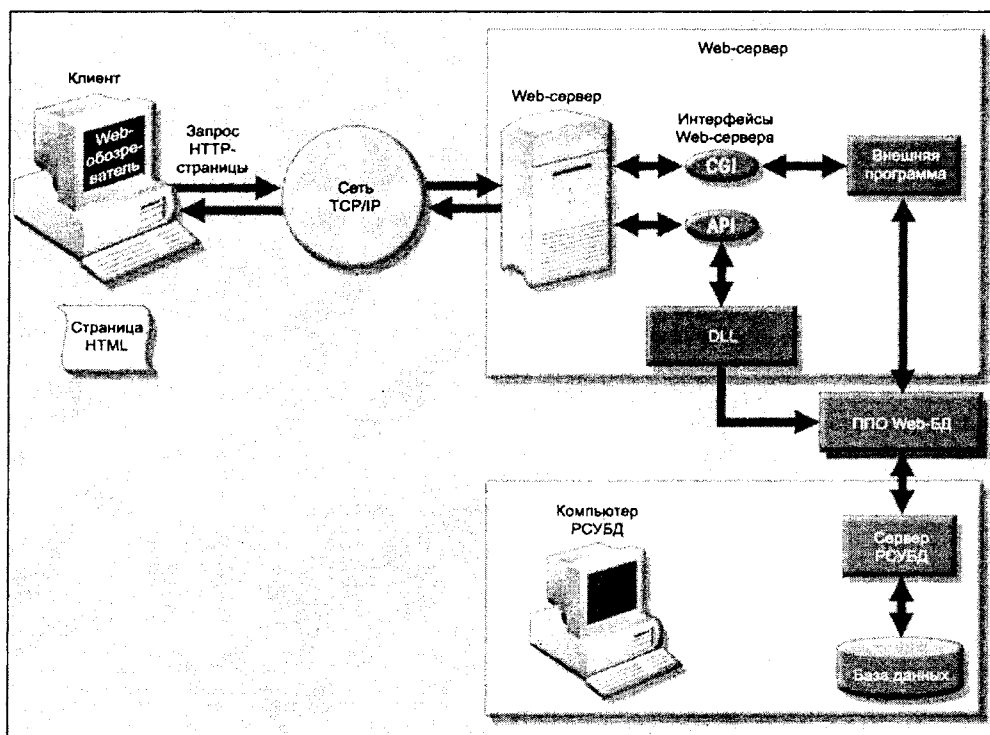


Рис. 15.2. API- и CGI-интерфейсы Web-серверов

Хотя интерфейсы API более эффективны при обработке запросов, у них все же есть недостатки. Поскольку все API совместно используют то же пространство памяти, что и Web-сервер, ошибки в API могут вывести из строя сервер. Другой недостаток состоит в том, что API привязаны к конкретному Web-серверу и к операционной системе.

Во время написания этой книги имелось три хорошо проработанных интерфейса API для Web-сервера:

- ☐ Netscape API (NSAPI) для серверов Netscape;
- ☐ Internet Server API (ISAPI) для Web-серверов Microsoft Windows;
- ☐ WebSite API (WSAPI) для серверов O'Reilly WebSite.

Различные типы Web-интерфейсов представлены на рис. 15.2.

Независимо от типа используемого интерфейса Web-сервера, программа ППО Web-БД должна связаться с базой данных. Это соединение может быть выполнено двумя способами:

- ☐ использование собственного ППО SQL, входящего в поставку. Например, можно использовать SQL\*Net, если установлена СУБД Oracle;
- ☐ использование службы Open DataBase Connectivity (ODBC) или интерфейса Object Linking and Embedding Database (OLE DB), если используется Windows-платформа.

Большинство поставщиков в качестве стандартного способа подключения к базе данных используют ODBC. Поэтому далее мы рассмотрим применение ODBC в качестве ППО доступа к базе данных.

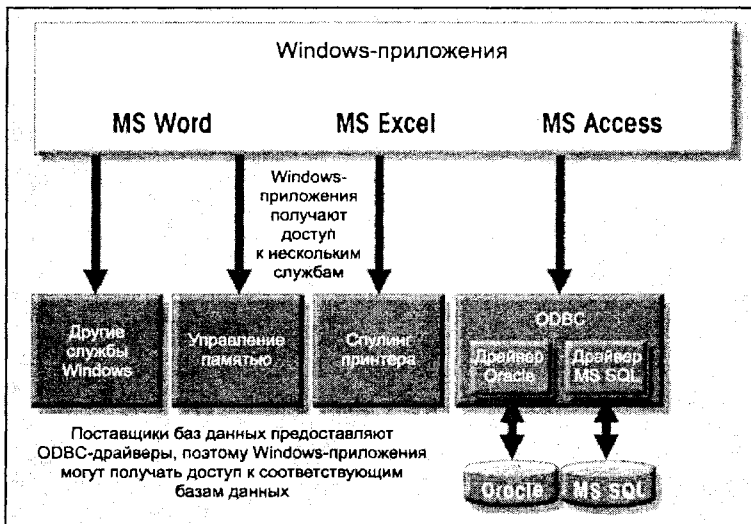
### 15.3.2. Открытый интерфейс доступа к базам данных (ODBC)

Чтобы узнать, как взаимодействуют программа расширения и СУБД, посмотрим, что для этого требуется в большинстве операционных систем Microsoft Windows. В среде Windows наиболее эффективные приложения для доступа к базам данных используют специальные программы, такие как Access, DB2, Oracle или SQL Server. Такой доступ к базам данных достигается с помощью специального ППО доступа к БД, называемого *открытым интерфейсом доступа к базам данных (Open DataBase Connectivity, ODBC)*. ODBC это реализация супернабора стандартов на доступ к базам данных Call Level Interface (CLI) группы SQL Access Group корпорации Microsoft. ODBC позволяет любому приложению Windows получать доступ к любому источнику данных — от реляционной системы баз данных до любого плоского файла — с помощью стандартного языка SQL. Рис. 15.3 иллюстрирует использование ODBC и других сервисов в операционных системах линейки Windows.

Из рис. 15.3 следует, что:

- ☐ все Windows-приложения используют сервисы совместного доступа, такие как управление памятью, спулинг (подкачка данных), управление интерфейсом пользователя и т. д.;

- ❑ одним из таких сервисов совместного доступа является ODBC. Например, приложение MS Access может использовать ODBC для прямого доступа к данным из внешнего источника. Такие данные могут быть расположены на том же компьютере или на другом компьютере локальной сети;
- ❑ Windows-сервисы совместного доступа реализованы как разделяемый код, который может динамически подключаться к операционной среде Windows (библиотеки DLL, хранимые в виде файлов с расширением dll). Выполняемый как DLL, этот код сокращает время загрузки и выполнения;
- ❑ поставщики баз данных обычно предлагают собственное ППО для установки необходимого подключения к БД. Например, в Oracle для получения доступа к БД необходимо установить ППО Oracle SQL\*Net.



**Рис. 15.3.** Применение ODBC для доступа к базам данных в Windows-приложениях

Помните, что Web-обозреватель не может напрямую обратиться к БД. Основное предназначение Web-обозревателя — *запрос Web-страниц у Web-сервера*. Это свойство имеет важное значение в том случае, если обозреватель остается независимым от операционной системы. Например, интерфейс ODBC доступен только в операционных системах Windows и в некоторых реализациях UNIX. Правильное использование ODBC в Web-среде представлено на рис. 15.4.

На рис. 15.4 обратите особое внимание на последовательность действий:

1. Web-обозреватель затребовал страницу у Web-сервера.
2. Web-сервер передает запрос на ППО Web-БД с помощью CGI или API.
3. ППО Web-БД использует ODBC для подключения к БД.
4. ППО получает результаты запроса и создает HTML-страницу.

5. ППО Web-БД посылает эту страницу обратно Web-серверу с помощью стандартов передачи CGI или API.
6. Web-сервер посылает HTML-страницу обозревателю.
7. Страница HTML отображается на Web-обозревателе клиентского компьютера.

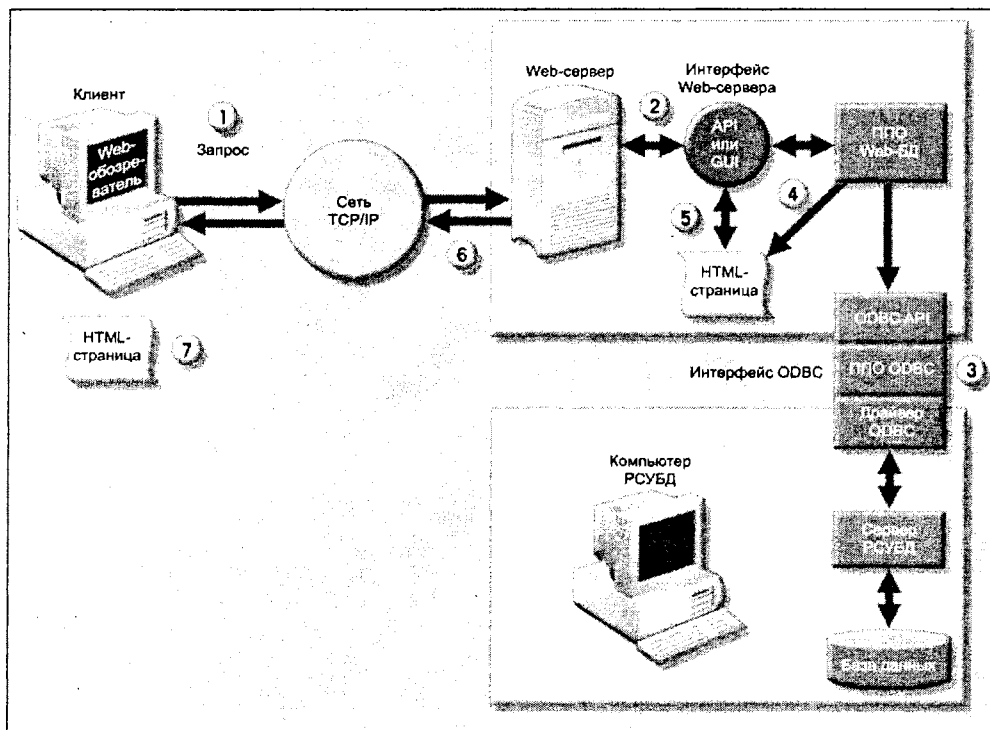


Рис. 15.4. Использование ODBC в качестве ППО Web-БД для доступа к базам данных

## 15.4. Web-обозреватель

Web-обозреватель размещен на компьютере клиента и представляет собой интерфейс пользователя к World Wide Web (вспомните, что основная функция обозревателя — запрос страниц и представление их конечному пользователю). Каждый раз, когда пользователь щелкает мышью на гиперссылке, обозреватель создает страницу запроса HTTP GET, которую посылает соответствующему Web-серверу с помощью протокола TCP/IP.

Работа Web-обозревателя заключается в том, чтобы интерпретировать HTML-код, полученный от Web-сервера, и представить различные компоненты страницы в стандартном виде. К сожалению, возможностей обозревателя по представлению и

интерпретации страниц недостаточно для разработки Web-приложений. Для расширения этих возможностей приходится использовать подключаемые модули (plug-ins) и другие клиентские расширения.

### 15.4.1. Клиентские расширения

*Клиентские расширения (client-side extensions)* дополняют функциональные возможности Web-обозревателя. Хотя эти расширения доступны в различных формах, чаще всего они представлены следующим образом.

- подключаемые модули (plug-ins);
- языки Java и JavaScript;
- элементы управления ActiveX и язык сценариев VBScript.

*Подключаемый модуль (plug-in)* представляет собой внешнее приложение, которое при необходимости автоматически иницируется обозревателем. Поскольку приложение внешнее, оно зависит от операционной системы. Подключаемый модуль связан с объектом данных (в основном, с помощью расширения имени файла) и позволяет Web-серверу надлежащим образом обрабатывать данные, которые изначально не поддерживались. Например, если один из компонентов страницы представляет собой документ формата PDF (portable document format, формат переносимого документа), Web-сервер получит эти данные, распознает их как объект в формате переносимого документа и запустит программу Adobe Acrobat Reader для представления и манипулирования таким документом на клиентском компьютере.

*Java* представляет собой объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems, который выполняется на верхнем уровне программного обеспечения Web-обозревателя. Java-приложения компилируются и хранятся на Web-сервере (язык Java во многом напоминает язык C++). Вызовы Java-программ встроены в HTML-страницу. Когда обозреватель находит такой вызов, он загружает Java-код с Web-сервера и выполняет его на клиентском компьютере. Основное достоинство Java — возможность выполнять созданные однажды приложения в различных средах. (При разработке Web-приложений способность к взаимодействию очень важна. К сожалению, клиентские обозреватели Microsoft и Netscape несовместимы на 100%, что ограничивает переносимость приложений.)

*JavaScript* — язык сценариев (серий команд, или макросов) фирмы Netscape, позволяющий Web-разработчикам создавать интерактивные сайты. Поскольку язык JavaScript проще, чем Java, его легче изучить. Код JavaScript встраивается в Web-страницу, загружается вместе с Web-страницей и активируется при возникновении определенного события — например, при щелчке кнопкой мыши на объекте или при загрузке страницы с сервера в память.

*ActiveX* — разработанная корпорацией Microsoft альтернатива Java. Элементы управления ActiveX представляют собой спецификацию написания программ, выполняемых внутри клиентского обозревателя Microsoft Internet Explorer. Поскольку спецификация ActiveX ориентирована главным образом на Windows-приложения, переносимость ее невысока. ActiveX расширяет возможности Web-обозревателя, добавляя на Web-страницу новые элементы управления. (Примерами таких элементов управления являются всплывающие окна, ползунковые регуляторы, календарь, каль-

кулятор и т. д.). Эти элементы управления позволяют манипулировать данными внутри обозревателя и загружаются с Web-сервера по мере необходимости. Элементы управления ActiveX могут создаваться на многих языках программирования, чаще других используются C++ и Visual Basic.

*VBScript* — еще один продукт корпорации Microsoft, предназначенный для расширения функциональных возможностей обозревателя. Язык разработки сценариев VBScript происходит от языка Microsoft Visual Basic. Как и в случае с JavaScript, код VisualBasic встраивается в HTML-страницу и активируется при возникновении события, например, при щелчке кнопкой мыши на ссылке.

С точки зрения разработчика необходимо использовать программы, позволяющие выполнять проверку данных на клиентской стороне. Например, если при вводе данных в Web-форму на клиентской стороне не выполняется никакой проверки, то весь набор данных (возможно, неверных) отправляется на Web-сервер. В этом случае сервер должен выполнить все работы по проверке данных, что занимает достаточно много времени центрального процессора. Поэтому проверка вводимых данных на стороне клиента — одно из основных требований при разработке Web-приложений. Большинство программ проверки данных написано на Java, JavaScript, ActiveX или VBScript.

## 15.5. Инструменты Web-БД: ColdFusion

Для понимания работы интерфейсов ППО Web-БД необходимо знать, как они создаются, и посмотреть их в действии. В этом разделе мы испытаем ColdFusion — новое поколение продуктов, называемых серверами Web-приложений. *Сервер Web-приложений* представляет собой промежуточное программное обеспечение, расширяющее функциональные возможности Web-серверов путем подключения их к разнообразным сервисам, таким как базы данных, системы каталогов, поисковые машины и т. д. Сервер Web-приложений также предоставляет надежную исполнительную среду для Web-приложений.

ППО ColdFusion может использоваться в следующих целях:

- ☐ подключение и запросы к базе данных с Web-страницы;
- ☐ представление данных БД на Web-странице в различных форматах;
- ☐ создание динамических поисковых Web-страниц;
- ☐ создание Web-страниц для вставки, обновления и удаления информации в БД;
- ☐ определение необходимых и необязательных связей;
- ☐ определение необходимых и необязательных полей формы;
- ☐ обеспечение целостности ссылок полей формы;
- ☐ использование простых и вложенных запросов и выбранных полей форм для представления бизнес-правил.

### Примечание

Хотя ColdFusion обладает большой функциональностью, цель данного раздела — показать, как создать и использовать простой и полезный интерфейс Web-БД. Подробнее о возможностях ColdFusion вы можете узнать, посетив сайт [www.macromedia.com](http://www.macromedia.com).

Мы выбрали продукт ColdFusion, разработанный в Allaire Corporation (недавно объединившейся с корпорацией Macromedia), по нескольким причинам:

- ❑ ColdFusion — мощный и надежный программный продукт, позволяющий создавать даже сложные решения по доступу к Web-БД;
- ❑ несмотря на свою мощь, это продукт достаточно дружелюбный для пользователей и разработчиков. Многие корпорации все больше поддерживают ColdFusion. Используя ColdFusion, можно получить достаточный практический опыт работы со средой Web-БД и расширить свои познания в этой области;
- ❑ корпорация Macromedia предлагает тестовую 30-дневную версию последнего выпуска ColdFusion, которую можно загрузить с [www.macromedia.com](http://www.macromedia.com). Поскольку в ColdFusion включен полный набор оперативной документации с работающими демонстрационными приложениями, иллюстрирующими функциональные возможности продукта, можно не заботиться о покупке дополнительной литературы. Кроме того, для образовательных учреждений система ColdFusion поставляется на весьма выгодных условиях, что делает ее доступной даже при недостаточном финансировании программ и позволяет использовать в целях обучения.

Во время написания этой книги были доступны и многие другие серверы Web-приложений, например, Oracle Application Server корпорации Oracle, Enterprise Application Server корпорации Sybase, WebLogic компании BEA Systems, Netscape Application Server и NetDynamics корпорации Sun Microsystems, Authoring Server компании NetObjects, Visual InterDev корпорации Microsoft, Sapphire/Web корпорации Bluestone, WebObjects компании Apple, а также HANTsite корпорации HANT Software.

Сервер Web-приложений предоставляет следующие функциональные возможности:

- ❑ интегрированную среду разработки (IDE) с менеджером сеансов и поддержкой переменных приложения;
- ❑ обеспечение безопасности и аутентификации пользователей на основе идентификаторов и паролей;
- ❑ специализированные языки для представления и хранения бизнес-логики на сервере приложений;
- ❑ автоматическую генерацию HTML-страниц со встроенными вызовами Java, JavaScript, VBScript, CGI-программ на C++ и т. д.;
- ❑ обеспечение высокой производительности и отказоустойчивости, например, средства выравнивания нагрузки, кэширования запросов, организации пула связей и кластеризации сервера с возможностью восстановления после отказа;
- ❑ доступ к базе данных с возможностями управления транзакциями;
- ❑ доступ к другим сервисам с помощью стандартных интерфейсов, например, таких как протокол передачи файлов (FTP), связь с базой данных (ODBC и OLE DB), почтовые службы и службы каталогов, Java Database Connectivity (JDBC), Object Repository и т. п.

Все эти продукты обеспечивают возможность связи Web-серверов со многими источниками данных и другими сервисами. Эти продукты отличаются друг от друга набором служб, надежностью, масштабируемостью, простотой использования, со-

вместимостью с базами данных и другими средствами Web и возможностями среды разработки.

### 15.5.1. Как работает ColdFusion

ColdFusion представляет собой вполне законченный сервер Web-приложений, обеспечивающий средства связи с базами данных, системами электронной почты, системами каталогов, поисковыми машинами и т. д. Для того чтобы обеспечить эти возможности, в ColdFusion имеется серверный язык разметки — ColdFusion Markup Language (CFML), предназначенный для создания прикладных страниц ColdFusion, называемых *сценариями*. Сценарий представляет собой серию инструкций, выполняемых в режиме интерпретации. Файл сценария не компилируется, как программа на языках COBOL, C++ или Java (в этом отношении сценарий напоминает командный файл DOS). Сценарии ColdFusion содержат код, необходимый для подключения, подачи запроса и обновления базы данных от внешнего интерфейсного приложения Web.

Сценарии ColdFusion содержат комбинацию тегов HTML, тегов ColdFusion и, если необходимо, операторов Java, JavaScript или VBScript. Термин ColdFusion начинается с символов <CF и в него могут включаться открывающие и закрывающие компоненты, например, <CFQUERY> (начало запроса) и </CFQUERY> (конец запроса). Сценарии ColdFusion хранятся в файлах с расширением cfm. Когда клиентский обозреватель запрашивает cfm-страницу у Web-сервера, сервер приложений ColdFusion выполняет инструкции cfm-сценария и посылает результат (в формате HTML) на Web-сервер. Web-сервер затем посылает этот документ на компьютер клиента для отображения. На рис. 15.5 представлены компоненты сервера Web-приложений ColdFusion и их работа.

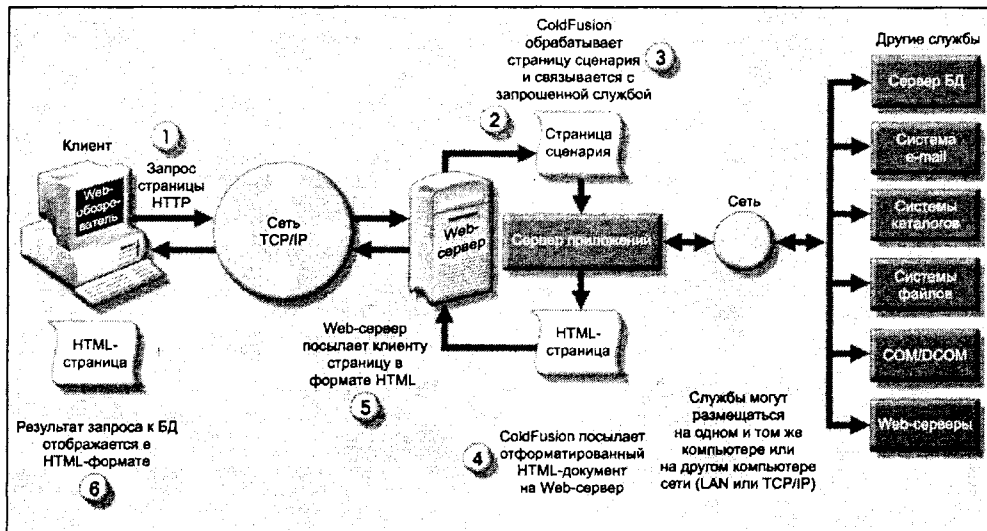


Рис. 15.5. Работа сервера Web-приложений ColdFusion



## 15.5.2. Демонстрационная база данных компании RobCor

Покажем создание интерфейса Web-БД с помощью ColdFusion на примере небольшой базы данных Microsoft Access с именем RobCor. Следующие разделы посвящены созданию сценариев ColdFusion для выборки, ввода, обновления и удаления данных в БД RobCor.

### Примечание

Чтобы уделить основное внимание использованию тегов CFML для доступа к БД, в примерах предполагается, что вы знакомы с тегами HTML и с процессом редактирования HTML-документов. Примеры, представленные в этой главе, можно создать в стандартном текстовом редакторе Windows, например, NotePad (Блокнот).

База данных RobCor, реляционная схема которой представлена на рис. 15.6, спроектирована для отслеживания заказов, которые делаются клиентами в компании, состоящей из нескольких отделов.

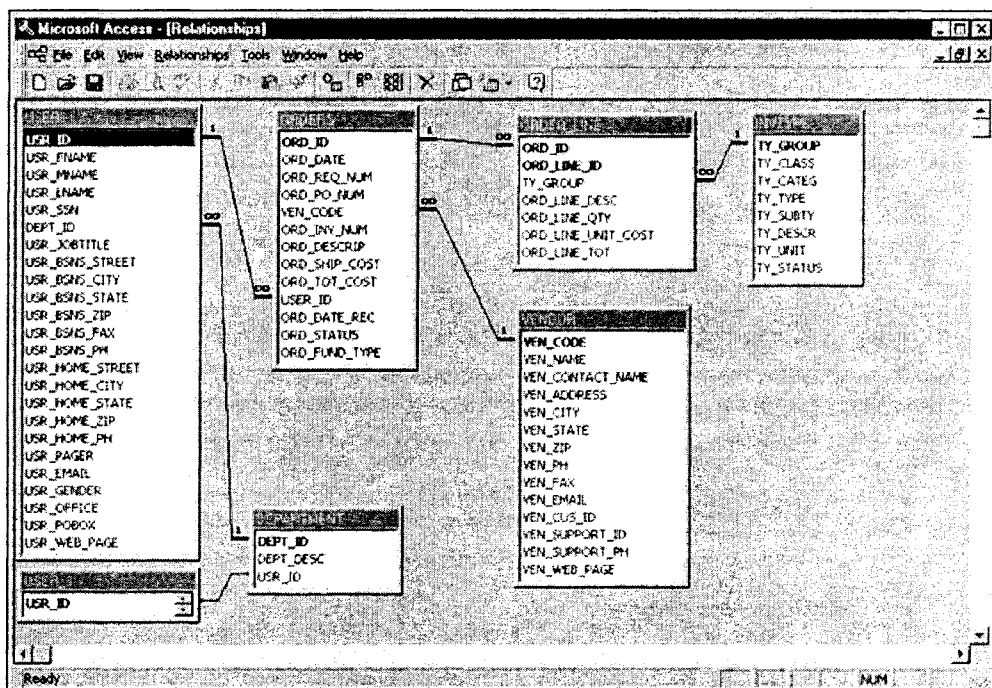


Рис. 15.6. Реляционная схема базы данных RobCor

Из рис. 15.6 следует, что в базе данных имеются следующие таблицы: USER (сотрудники), DEPARTMENT (отделы), VENDOR (поставщики), INVTYPE (тип

товара), ORDERS (заказы) и ORDER\_LINE (строки заказов). Связи между этими таблицами построены на основе следующих бизнес-правил:

- ☐ в штате отдела несколько сотрудников;
- ☐ отдел управляется только одним сотрудником;
- ☐ каждый сотрудник принадлежит одному отделу, и в каждом отделе много сотрудников;
- ☐ каждый отдел *может* иметь руководителя (т. е. руководитель необязателен);
- ☐ каждый заказ адресуется только одному поставщику, и каждый поставщик может получить несколько заказов;
- ☐ каждый заказ содержит одну или более строк;
- ☐ каждая строка заказа связана только с одним типом товара.

Таблица USER\_1 — это виртуальный компонент, созданный MS Access при задании множественной связи (multiple relationship) между таблицами USER и DEPARTMENT. MS Access создал виртуальную таблицу USER\_1 для представления связи "сотрудник (USER) управляет отделом (DEPARTMENT)". Это связь 1:1, что позволяет полю USER\_ID таблицы DEPARTMENT принимать пустые значения (null). (Мы создали эти связи, чтобы в дальнейшем продемонстрировать, как можно управлять необязательными связями с помощью Web-интерфейса.)

### 15.5.3. Создание простого запроса с помощью тегов <CFQUERY> и <CFOUTPUT>

Начнем с разработки простого сценария, создающего запрос для получения списка всех поставщиков в таблице VENDOR. Этот сценарий должен выполнять следующие два шага:

1. Запросить базу данных с помощью стандартного SQL для извлечения набора данных, в котором содержатся все записи, найденные в таблице VENDOR.
2. Конвертировать все записи, созданные на шаге 1, в формат HTML, чтобы можно было включить их в страницу, отправляемую клиентскому обозревателю.

Сценарий, представленный в листинге 15.1, содержит необходимый для этого код.

#### Примечание

Файлы сценариев, которые можно найти на [www.course.com](http://www.course.com) в Instructor's Resource Kit (материалы для преподавателей), соответствуют листингам сценариев, представленным для каждого примера. Например, файл листинга 15.1 называется ch15-1.cfm.

#### Листинг 15.1. Создание запроса с помощью тегов ColdFusion <CFQUERY> и <CFOUTPUT>

```
1: <HTML>
2: <HEAD>
3: <TITLE> Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
```

```

4: <CFQUERY Name="venlist" DataSource="RobCor">
5: SELECT * FROM VENDOR ORDER BY VEN_CODE
6: </CFQUERY>
7: </HEAD>
8: <BODY BGCOLOR="LIGHTBLUE">
9: <H1>
10: <CENTER>Simple Query using CFQUERY and CFOUTPUT</CENTER>
11: <CENTER>(VERTICAL OUTPUT)</CENTER>
12: </H1>
13:

14: <HR>
15: <CFOUTPUT>
16: Your query returned #venlist.RecordCount# records
17: </CFOUTPUT>
18: <CFOUTPUT QUERY="venlist">
19: <PRE>
20: VENDOR CODE: #VEN_CODE#
21: VENDOR NAME: #VEN_NAME#
22: CONTACT PERSON: #VEN_CONTACT_NAME#
23: ADDRESS: #VEN_ADDRESS#
24: CITY: #VEN_CITY#
25: STATE: #VEN_STATE#
26: ZIP: #VEN_ZIP#
27: PHONE: #VEN_PH#
28: FAX: #VEN_FAX#
29: E-MAIL: #VEN_EMAIL#
30: CUSTOMER ID: #VEN_CUS_ID#
31: SUPPORT ID: #VEN_SUPPORT_ID#
32: SUPPORT PHONE: #VEN_SUPPORT_PH#
33: VENDOR WEB PAGE: #VEN_WEB_PAGE#
34: <HR></PRE>
35: </CFOUTPUT>
36: <FORM ACTION="chl5-0.cfm" Method="post">
37: <input type="submit" value="Main Menu ">
38: </FORM>
39: </BODY>
40: </HTML>

```

### Примечание

Если вы установили демонстрационную версию ColdFusion, то сможете выполнить этот сценарий (и все последующие), указав в обозревателе адрес Web-сервера. Если ваш компьютер является Web-сервером, то в качестве Web-адреса вы можете использовать <http://127.0.0.1/RobCor/ch15-1.cfm>. Если вашим Web-сервером является другой компьютер, назначенный преподавателем, используйте адрес, заданный преподавателем.

В листинге 15.1 обратите внимание, что тег ColdFusion `<CFQUERY>` предназначен для запроса к базе данных, а тег `<CFOUTPUT>` — для отображения данных, возвращенных после выполнения запроса (теги ColdFusion в листингах выделены полужирным шрифтом). Рассмотрим эти два тега CFML подробнее.

□ Тег `<CFQUERY>` (строки 4–6). Этот тег устанавливает подключение к базе данных и выполняет вложенный SQL-оператор. Вы должны включать все операторы запроса *перед* или *внутри* заголовка HTML-документа (`<HEAD>`). При этом страница отображает вывод на стороне клиента *после выполнения всех запросов*. В противном случае обозреватель будет восприниматься как "медленный", поскольку страница начинает отображать вывод, а затем ожидать дополнительных данных, поступающих с сервера. Тег `CFQUERY` требует наличия следующих параметров:

- `NAME` = "имязапроса". Это обязательный параметр, имя которого используется для уникальной идентификации набора записей, возвращаемых по данному запросу, — в данном случае `venlist`. В одном сценарии можно создать несколько запросов, каждый со своим уникальным именем;
- `DATASOURCE` = "имяисточникаданных". Это также обязательный параметр, в котором задается имя базы данных, как определено в ODBC. Помните, что имя базы данных чувствительно к регистру. Поэтому если база данных называется RobCor, то это не то же самое, что ROBCOR или robcor. Используйте интерфейс администратора ColdFusion для определения всех источников данных. Источник данных может использовать ODBC, свои собственные драйверы (например, Oracle SQL\*Net) или Microsoft OLE (внедрение и связывание объектов);
- SQL-оператор (строка 5) — еще один обязательный параметр. В данном случае этот параметр определяет запрос

```
SELECT * FROM VENDOR ORDER BY VEN_CODE
```

но можно использовать любой SQL-оператор, совместимый с ODBC.

□ Тег `<CFOUTPUT>` (строки 15–17 и 18–35). Этот тег предназначен для отображения результатов работы тега `<CFQUERY>` или для вызова других переменных или функций ColdFusion. Параметрами этого тега являются:

- `QUERY` = "имязапроса". Это необязательный параметр (см. строку 18). Если вы используете имя для запроса, который возвращает 10 строк, то этот тег будет выполнять все команды между открывающим и закрывающим тегами `<CFOUTPUT>` 10 раз (по одному разу на каждую строку). Иначе говоря, этот тег работает как оператор цикла, который выполняется столько раз, сколько имеется строк в именованном запросе.
- Можно вставлять любой допустимый тег HTML или текст внутри открывающего и закрывающего тегов `<CFOUTPUT>`. В ColdFusion знак "решетка" (`#`)

используется для ссылки на поля запроса в результирующем наборе или для вызова других функций или переменных ColdFusion. Когда исполняющая система ColdFusion встречает имя внутри знаков #, она использует эту именованную переменную для проверки того, является ли она названием поля или запроса, внутренней переменной или функцией. На основе этой оценки ColdFusion заменяет именованную переменную значением, соответствующим запросу, внутренней переменной или функции. В данном случае строка 16 — это обращение к внутренней переменной ColdFusion. При выполнении запроса значением этой переменной является число строк в ответе на запрос. После имени запроса должно стоять ключевое слово RecordCount и эти два компонента должны разделяться точкой. Например, мы используем в данном запросе #venlist.RecordCount# для именования переменной в строке 16.

- Строки 19—34. Эти строки циклически повторяются по одной для каждой записи, возвращенной именованным запросом. В нашем примере цикл определяется выражением QUERY = "venlist". Обратите внимание (строки 20—33), что названия полей (заключенные в символы #) будут заменены фактическими значениями полей, возвращенных запросом.

Вывод, который будет получен в результате выполнения сценария 15.1, представлен на рис. 15.7.

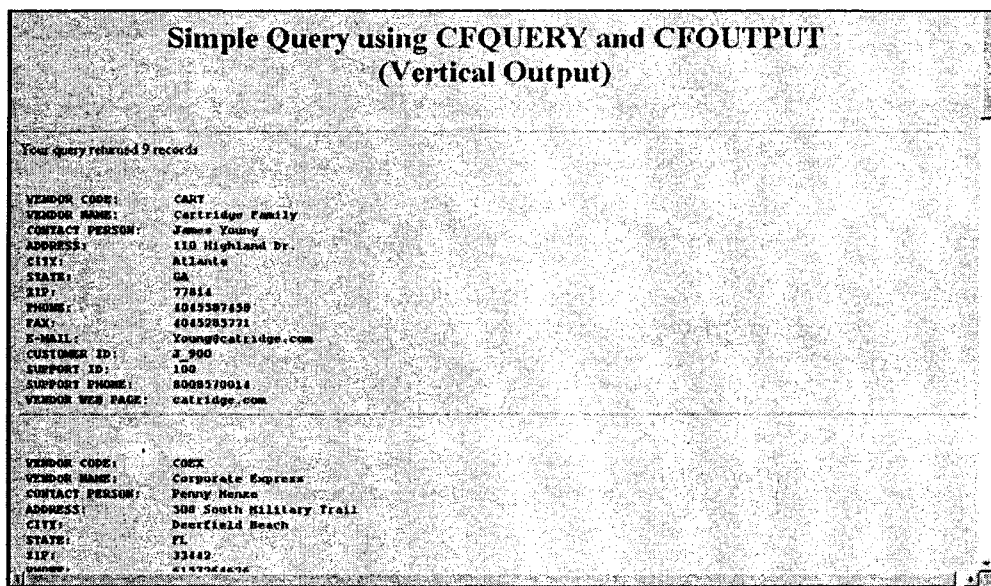


Рис. 15.7. Результаты выполнения сценария ch15-1.cfm

Вариант только что описанного подхода представлен в листинге 15.2, где вывод предварительно форматируется (с помощью HTML-тега <PRE>) чтобы на каждую запись приходилась одна строка с помощью тега <CFOUTPUT>.

### Листинг 15.2. Тег <CFQUERY> с табулированным тегом <CFOUTPUT>

```

1: <HTML>
2:
3: <CFQUERY NAME="VENDATA" DATASOURCE="RobCor">
4: SELECT * FROM VENDOR ORDER BY VEN_CODE
5: </CFQUERY>
6:
7: <HEAD>
8: <TITLE>Rob & Coronel – Chapter 15: Cold Fusion Examples</TITLE>
9: </HEAD>
10:
11: <BODY BGCOLOR="LIGHTBLUE">
12: <H1>
13: <CENTER>Simple Query using CFQUERY and CFOUTPUT</CENTER>
14: <CENTER>(HORIZONTAL OUTPUT)</CENTER>
15: </H1>
16:

17: <HR>
18: <PRE>
19:
20: VENDOR CODE VENDOR NAME CONTACT PERSON ADDRESS CITY STATE
21: ZIP PHONE
22:
23: <PRE>
24: <CFOUTPUT QUERY='VENDATA">
25: <PRE>
26: #VEN_CODE# #VEN_NAME# #VEN_CONTACT_NAME# #VEN_ADDRESS# #VEN_CITY
27: #VEN_STATE# #VEN_ZIP# #VEN_PH#

28: </PRE>
29: </CFOUTPUT>
30:
31: <FORM ACTION="chl5-0.cfm" Method="post">
32: <input type="submit" value="Main Menu ">
33:
34: </BODY>
35: </HTML>

```

Результат выполнения сценария, представленного в листинге 15.2, показан на рис. 15.8.

| VENDOR CODE | VENDOR NAME           | CONTACT PERSON | ADDRESS                  | CITY            | STATE | ZIP   | PHONE      |
|-------------|-----------------------|----------------|--------------------------|-----------------|-------|-------|------------|
| CART        | Cartridge Family      | James Young    | 110 Highland Dr.         | Atlanta         | GA    | 77814 | 4045587488 |
| COEX        | Corporate Express     | Fenny Henze    | 509 South Military Trail | Deerfield Beach | FL    | 33442 | 6157264826 |
| DELL        | Dell Computer         | Kim Berringer  | 250 Main St.             | Round Rock      | TX    | 78682 | 8007815222 |
| GLOBAL      | Global Corporation    | Michael Fox    | 4744 Rock Rd.            | Chicago         | IL    | 45782 | 8004571555 |
| LSREX       | Laser Express Company | Micole Whether | 5140 Pine St.            | Albany          | NY    | 41580 | 5184547110 |

Main Menu

Рис. 15.8. Результат выполнения сценария  
(файл ch15-2.cfm)

#### 15.5.4. Создание простого запроса с помощью тегов <CFQUERY> и <CFTABLE>

Как видно на рис. 15.8, вывод, полученный с помощью тега <CFOUTPUT>, не выровнен. Чтобы получить более элегантный список поставщиков, представленный в табличной форме, можно использовать тег <CFTABLE>. Этот тег автоматически выводит результаты в табличной форме, где каждая строка набора данных располагается в строке результирующей таблицы. Исходный код представлен в листинге 15.3.

##### Листинг 15.3. Теги <CFQUERY> и <CFTABLE>

```

1: <HTML>
2: <HEAD>
3: <TITLE> Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: </HEAD>
5: <BODY BGCOLOR="LIGHTBLUE">
6: <CFQUERY NAME="VENDATA" DATASOURCE="RobCor">
7: SELECT * FROM VENDOR ORDER BY VEN_CODE

```

```

8: </CFQUERY>
9:
10: <H1>
11: <CENTER>SIMPLE QUERY USING CFQUERY and CFTABLE</CENTERX>
12: </H1>
13:

14: <table bgcolor="Silver" bordercolor="Fuchsis" frame="Border">
15: <tr><HR></tr>
16: <tr>
17: <cftable query="VENDATA" startrow="1" colspacing="1" colheaders>
18: <CFCOL HEADER="CODE" WIDTH="8" TEXT="#VEN_CODE#">
19: <CFCOL HEADER="VENDOR_NAME" WIDTH="25" TEXT="#VEN_NAME#">
20: <CFCOL HEADER="CONTACT_PERSON" WIDTH="15"
21: TEXT="#VEN_CONTACT_NAME#">
22: <CFCOL HEADER="ADDRESS" WIDTH="20" TEXT="#VEN_ADDRESS#">
23: <CFCOL HEADER="CITY" WIDTH="10" TEXT="#VEN_CITY#">
24: <CFCOL HEADER="STATE" WIDTH="2" TEXT="#VEN_STATE#">
25: <CFCOL HEACER="ZIP" WIDTH="5" TEXT="#VEN_ZIP#">
26: <CFCOL HEADER="PHONE" WIDTH="10" TEXT="VEN_PH#">
27: <CFCOL HEADER="FAX" WIDTH="10" TEXT="#VEN_FAX#">
28: <CFCOL HEADER="E-MAIL" WIDTH="10" TEXT="#VEN_EMAIL#">
29: <CFCOL HEADER="CUSTOMER_ID" WIDTH="8" TEXT="#VEN_CUS_ID#">
30: <CFCOL HEADER="SUPPORT_PHONE" WIDTH="6"
31: TEXT="#VEN_SUPPORT_ID#">
32: <CFCOL HEADER="WEB_PAGE" WIDTH="15" TEXT="#VEN_WEB_PAGE#">
33: </CFTABLE>
34: </tr>
35: <TR>
36: <FORM ACTION="chl5-0.cfm" Method="post">
37: <input type="submit" value="Main Menu ">
38: </FORM>
39: </tr>
40: </table>
41: </BODY>
42: </HTML>

```

Результат выполнения файла сценария представлен на рис. 15.9.



| Simple Query Using CFQUERY and CFTABLE |                       |                |                      |            |    |       |            |            |            |
|----------------------------------------|-----------------------|----------------|----------------------|------------|----|-------|------------|------------|------------|
| CODE                                   | VENDOR NAME           | CONTACT PERSON | ADDRESS              | CITY       | ST | ZIP   | PHONE      | FAX        | E-MAIL     |
| CART                                   | Cartridge Family      | James Young    | 110 Highland Dr.     | Atlanta    | GA | 77814 | 4045587458 | 4045285771 | Youngj@cat |
| CDEX                                   | Corporate Express     | Penny Henze    | 508 South Military T | Deerfield  | FL | 33442 | 6157264626 | 6152554772 | pennyhenz  |
| DELL                                   | Dell Computer         | Kia Berringer  | 250 Main St.         | Round Rock | TX | 78682 | 8007815222 | 8008425686 | Customer@  |
| GLOBAL                                 | Global Corporation    | Michael Fox    | 4744 Rock Rd.        | Chicago    | IL | 45787 | 8004371333 | 8007872421 | Glo_3338@  |
| LSBEV                                  | Laser Express Company | Nicole Whether | 5540 Pine St.        | Albany     | NY | 41550 | 5184547110 | 5185754570 | Janen@kno  |

Рис. 15.9. Результат выполнения сценария 15.3

Тег ColdFusion <CFTABLE> в сценарии 15.3 состоит из следующих компонентов.

□ Тег <CFTABLE> (строки 17—31). Этот тег, который используется для отображения результатов запроса CFQUERY (строки 4—6) в табличной форме, требует следующих параметров:

- QUERY = "имязапроса". Необходимый параметр, в котором используется имя запроса, генерирующего набор данных для отображения в табличной форме;
- STARTROW = "1". Необязательный параметр, предназначенный для информирования ColdFusion о том, с какой строки запроса будет отображаться результат. Этот параметр особенно важен, в частности, когда запрос возвращает много строк, а вы не хотите отображать их все на одной длинной странице. Вместо этого, можно, например, разместить по 10 строк на странице, тогда на первой странице номер стартовой строки должен быть равен 1, на второй странице 11, на третьей 21 и т. д.;
- COLSPACING = "1". Необязательный параметр, предназначенный для определения количества пробелов, помещаемых между столбцами;
- COLHEADERS. Необязательный параметр, делающий первую строку в таблице строкой заголовка, который содержит имена столбцов, заданные в теге <CFCOL>.

□ Тег <CFCOL> (строки 18—30) предназначен для определения каждого столбца таблицы с помощью следующих параметров:

- HEADER = "текстзаголовка". Это текст заголовка, который появится в строке заголовка таблицы для каждого отображаемого столбца. Текст заголовка может включать HTML-теги;
- WIDTH = "x". Этот параметр определяет ширину столбца;
- TEXT = "#имяполязапроса#". Это фактическое значение, которое необходимо поместить в выбранный столбец. Например, в строке 18 исполняющая система ColdFusion заменит #VEN\_CODE# реальным значением, извлеченным с помощью запроса для этого поля.

## 15.5.5. Создание страниц динамического поиска

Здесь будет показано, как использовать теги `<CFQUERY>`, `<CFOUTPUT>` и `<CFTABLE>` для отправки SQL-оператора к базе данных для поиска набора данных. С помощью файлов сценария, созданных ранее в этой главе, запрос всегда будет возвращать одни и те же записи из таблицы `VENDOR` (поскольку SQL-оператор "жестко" закодирован внутри страницы, его нельзя изменить — если только SQL-оператор не изменять вручную всякий раз, когда нужно выполнить другой поиск). Такое статичное представление, очевидно, ограничивает возможности запросов.

Чтобы сконструировать полезный адаптируемый запрос, можно создать запрос динамического поиска, в котором условия поиска можно менять с помощью пользовательских опций — без необходимости редактирования всей страницы сценария. Чтобы продемонстрировать эту процедуру, мы используем два поля — код поставщика и штат (местонахождение) поставщика для поиска заданных пользователем записей о поставщике (естественно, можно создать поисковую форму для нескольких полей, которые, по вашему мнению, необходимы).

Для выполнения динамического запроса с помощью Web нужно выполнить два шага:

1. Создать сценарий, генерирующий форму, необходимую для ввода критериев поиска. Иначе говоря, эта форма должна предоставлять пользователю возможность ввести значение параметра, которое будет использовано в операторе запроса.
2. Создать сценарий, выполняющий запрос и отображающий результаты на основе значений параметров, переданы ему сценарием, созданным на шаге 1.

Сценарий, необходимый для выполнения шага 1, представлен в листинге 15.4а.

### Листинг 15.4а Запрос динамического поиска: форма ввода критериев поиска

```

1: <HTML>
2: <HEAD>
3: <TITLE> Rob & Coronel — Chapter 15: Cold Fusion Examples</TITLE>
4: </HEAD>
5:
6: <CFQUERY NAME="STATELIST" DATASOURCE="RobCor">
7: Select VEN_STATE from VENDOR Order by VEN_STATE
8: </CFQUERY>
9:
10: <BODY BGCOLOR="LIGHTBLUE">
11: <H1>
12: <CENTER>Dynamic Search Query: Criteria Entry Form</CENTER>
13: </H1>
14: <FORM ACTION="chl5-4b.cfm" METHOD=POST>
15: <table align="CENTER" bgcolor="Silver">
16: <TR>

```

```

17: <TD ALIGN="right">VEN_CODE</TD>
18: <TD>
19: <INPUT TYPE ="text" NAME="VEN_CODE" SIZE="10"
 MAXLENGTH="10"></TD>
20: </TR>
21: <TR>
22: <TD ALIGN="right">VEN_STATE</TD>
23: <TD>SELECT NAME="VEN_STATE" SIZE=1>
24: <OPTION SELECTED VALUE="ANY">ANY
25: <CFOUTPUT Query="STATELIST">
26: <OPTION VALUE="#STATELIST.VEN_STATE#">#VEN_STATE#
27: </CFOUTPUT>
28: </SELECT>
29: </TD>
30: </TR>
31: <TR>
32: <TD ALIGN="right" valign="middle">
33: <INPUT TYPE="submit" VALUE="Search">
34: </FORM>
35: </TD>
36: <TD ALIGN="right" valign="middle">
37: <FORM ACTION="ch15-0.cfm" Method="post">
38: <input type="submit" value="Main Menu ">
39: </FORM>
40: </TD>
41: </TR>
42: </TABLE>
43:
44: </BODY>
45: </HTML>

```

Результат выполнения сценария 15.4a представлен на рис. 15.10.

На рис. 15.10 показано, что сценарий 15.4a создает форму, в которой пользователь вводит параметры поиска. Посмотрим, как работает сценарий 15.4a.

- ❑ В строках 6—8 извлекаются имеющиеся значения из поля VEN\_STATE в таблице VENDOR. Этот запрос фактически сообщает нам, какие штаты представлены в таблице поставщиков. Если в данном штате не имеется поставщиков, то нечего будет передавать в результирующий набор. Этому запросу присвоено имя "STATELIST" и впоследствии он будет использоваться в нашей форме.

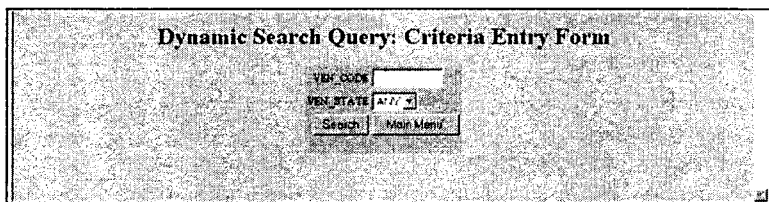


Рис. 15.10. Результат выполнения сценария 15.4а

- ❑ В строках 14—34 определяется форма. Обратите внимание, что когда пользователь нажимает кнопку **Submit**, то для передачи значений переменных формы VEN\_CODE и VEN\_STATE вызывается форма сценария 15.4б (который будет представлен чуть позже).
- ❑ В строке 19 представлено первое текстовое поле, где пользователь может вводить значение для переменной формы VEN\_CODE. Это значение используется в SQL-операторе для поиска всех записей с данным значением VEN\_CODE.
- ❑ В строках 23—29 создается раскрывающийся список SELECT, чтобы пользователь мог выбрать в нем штат, который будет использоваться при запросе таблицы поставщиков. Сделанный выбор позже будет передан в сценарий 15.4б.
- ❑ В строках 25—27 используется тег <CFOUTPUT> для создания опций выбора штатов, встречающихся в таблице VENDOR. Опция по умолчанию ("SELECTED"), представленная в строке 24, позволяет пользователю выполнять поиск без указания какого-то конкретного штата. Если не включить строку 24, то невозможно будет выполнить поиск только по коду поставщика (VEN\_CODE) и отобразить список поставщиков по всем штатам. Для ограничения поиска поле формы VEN\_STATE должно допускать возможность включения пустых значений. Иначе говоря, значение по умолчанию в строке 24 обеспечивает достаточную гибкость сценария.

Когда пользователь нажимает кнопку **Submit**, выполняется сценарий 15.4б и в него передаются значения двух переменных (VEN\_CODE и VEN\_STATE). Этот сценарий представлен в листинге 15.4б.

#### Листинг 15.4б. Результаты поиска поставщиков

```

1: <HTML>
2: <CTQUEPY NAME="SearchVendor" DATASOURCE="RobCor">
3: SELECT VEN_CODE, VEN_NAME, VEN_CONTACT_NAME, VEN_ADDRESS,
4: VEN_CITY, VEN_STATE, VEN_PH
5: FROM VENDOR
6: WHERE 0=0
7: <CFIF #FORM.VEN_CODE# IS NOT "">
8: AND VENDOR.VEN_CODE LIKE '%#FORM.VEN_CODE#%'
9: </CFIF>

```

```

9: <CFIF #FORM.VEN_STATE# IS NOT "ANY">
10: AND VENDOR.VEN_STATE LIKE '%"FORM.VEN_STATE%'
11: </CFIF>
12: ORDER BY VEN_CODE
13: </CFQUERY>
14: <HEAD>
15: <TITLE>Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
16: </HEAD>
17: <BODY BGCOLOR="LIGHTBLUE">
18: <H1>
19: <CENTER>Vendor Search Results</CENTER>
20: </H1>
21:

22: <CFTABLE QUERY="SearchVendor" STARTROW="1" COLSPACING="1"
 COLHEADERS>
23: <CFCOL HEADER="VENDOR CODE" WIDTH="15" TEXT="#VEN_CODE#">
24: <CFCOL HEADER="VENDOR NAME" WIDTH="20" TEXT="#VEN_NAME#">
25: <CFCOL HEADER="CONTACT PERSON" WIDTH="20"
 TEXT="#VEH_CONTACT_NAME#">
26: <CFCOL HEADER="ADDRESS" WIDTH="20" TEXT="#VEN_ADDRESS#">
27: <CFCOL HEADER="CITY" WIDTH="20" TEXT="VEN_CITY#">
28: <CFCOL HEADER="STATE" WIDTH="2" TEXT="#VEN_STATE#">
29: <CFCOL HEADER="PHONE" WIDTH="20" TEXT="#VEH_PH#">
30: </CFTABLE>
31: <CFIF #SearchVendor.RecordCount# IS 0>
32: <H2>No records were found matching your criteria </H2>
33: <CFELSE>
34: <CFOUTPUT>Your search returned #SearchVendor.RecordCount#
 record(s).</CFOUTPUT>
35: </CFIF>
36: <FORM ACTION="ch15-0.cfm" Method="post">
37: <input type="submit" value="Main Menu" >
38: </FORM>
39:
40: </BODY>
41: </HTML>

```

Результат выполнения сценария 15.46 представлен на рис. 15.11.

| Vendor Search Results |                       |                |                       |                 |               |
|-----------------------|-----------------------|----------------|-----------------------|-----------------|---------------|
| VENDOR CODE           | VENDOR NAME           | CONTACT PERSON | ADDRESS               | CITY            | ST PHONE      |
| CART                  | Cartridge Family      | James Young    | 110 Highland Dr.      | Atlanta         | GA 4045587458 |
| COEX                  | Corporate Express     | Fanny Menze    | 588 South Military Tr | Deerfield Beach | FL 6157264626 |
| DELL                  | Dell Computer         | Kia Herzinger  | 250 Main St.          | Round Rock      | TX 8007815222 |
| GLOBAL                | Global Corporation    | Michael Fox    | 4744 Rock Rd.         | Chicago         | IL 8004571555 |
| LIREX                 | Laser Express Company | Nicole Wheeler | 5540 Pine St.         | Albany          | NY 5184547110 |

You search returned 1 record(s).

Main Menu

Рис. 15.11. Результат выполнения сценария 15.4б

В сценарии 15.4б обратите внимание на то, каким образом в нем делается ссылка на переменные формы, созданные при выполнении сценария 15.4а, и как используется тег <CFIF> для динамического формирования SQL-оператора. Рассмотрим подробнее сценарий 15.4б.

- ☐ В строках 2—13 создается SQL-оператор и запрос к базе данных.
- ☐ В строке 5 создается пустой оператор WHERE, используемый для прикрепления условий запроса. Выражение "0=0" — это условие по умолчанию, когда перечисляются все записи. Эта строка необходима для формирования основы дополнительных условий с помощью логического SQL-оператора. Естественно, если не определено никаких дополнительных условий, то значение по умолчанию гарантирует, что будут выведены все записи.
- ☐ В строках 6—8 используется тег <CFIF> для расчета поля формы VEN\_CODE, переданный из вызываемой страницы (в данном случае вызывающую страницу создает сценарий 15.4а). Если поле, заданное в строке 6, не пустое (""), то условие, определенное в строке 7, добавляется к запросу.
- ☐ В строках 9—11 используется тег <CFIF> для расчета поля формы VEN\_STATE, переданного из вызывающей страницы (15.4а). Если это поле не равно "ANY", то в строке 10 к запросу добавляется "оператор" условия (помните, что "ANY" это выбор по умолчанию для поля VEN\_STATE сценария 15.4а). Этот выбор по умолчанию гарантирует, что необязательно выполнять поиск по определенному штату. Поэтому пользователь может выбрать штат для ограничения вывода по данному запросу или не выбирать штат и при этом получить список поставщиков по всем штатам.
- ☐ В строках 22—30 используются теги <CFTABLE> и <CFCOL> для отображения результатов запроса в табличной форме.
- ☐ Наконец, используется тег <CFIF> для расчета количества записей в результате набора и вывода соответствующего сообщения. Например, если возвращается ноль (0) записей, то строка 31 гарантирует вывод сообщения "No records where found matching your search criteria" ("Не найдено записей, соответствующих заданному критерию"), текст сообщения содержится в строке 32. Если количес-

во записей отлично от нуля, то строка 34 обеспечит вывод сообщения, в котором указывается число записей, возвращенных в результате запроса.

На рис. 15.12 представлен вывод динамического поиска на основе условия `VEN_STATE = "GA"` (поставщики штата Джорджия).

**Dynamic Search Query: Criteria Entry Form**

VEN\_CODE:

VEN\_STATE:

PL  
IL  
NY  
TX

**Vendor Search Results**

| VENDOR CODE | VENDOR NAME     | CONTACT PERSON | ADDRESS          | CITY    | ST PHONE     |
|-------------|-----------------|----------------|------------------|---------|--------------|
| CART        | CAPTISON Family | James Young    | 110 Highland Ex. | Atlanta | GA 404587456 |

Your search returned 1 record(s).

**Рис. 15.12.** Список поставщиков для условия `VEN_STATE = "GA"` (штат Джорджия)

Возможность динамического поиска, конечно же, делает работу в Web более привлекательной для пользователей. Однако для проведения *транзакций* необходимо также иметь возможность модифицировать содержимое таблиц баз данных. (Например, если вы берете что-нибудь со склада, то необходимо иметь возможность обновить содержимое таблицы `INVENTORY`. Если вы делаете покупку, то требуется возможность создавать запись о счете, и т. д.) Поэтому теперь обратим внимание на процедуры, которые можно использовать при вводе, обновлении и удалении данных с помощью Web-интерфейса. Перед тем как разрабатывать транзакции на базе Web-приложений, необходимо знать, каким образом недостатки HTML и обозревателей влияют на возможности манипулирования данными. Эти недостатки — следствие основной структуры Web, которую можно отнести к системе, не хранящей информацию о своем состоянии (*stateless system*).

### 15.5.6. Web как система без сохранения состояния

Web называют системой, не сохраняющей информацию о своем состоянии. Попросту говоря, термин "система без сохранения информации о состоянии" (*stateless system*) говорит о том, что Web-сервер не знает ничего о статусе своих клиентов, подключенных к нему. То есть не существует открытой линии связи между сервером

и каждым подключенным к нему клиентом — что, конечно, было бы просто невозможно для *глобальной* сети! Вместо этого взаимодействие клиентских и серверных компьютеров выглядит как серия коротких "бесед", соответствующих схеме "запрос-ответ". Например, обозреватель знает только *текущую* страницу, поэтому следующая страница абсолютно ничего не знает о том, что было загружено на первой странице. Клиентский и серверный компьютеры взаимодействуют только при запросе страницы (когда пользователь щелкает на ссылке) и когда сервер посылает запрошенную страницу клиенту. Поэтому хотя вы при просмотре страницы и *думаете*, что линия связи открыта, на самом деле, вы только просматриваете HTML-документ, хранящийся в локальном кэше (временный каталог) клиентского обозревателя. Сервер представления не имеет о том, что клиент делает с документом, какие данные он вводит в форму, какие опции выбирает и т. д. Если мы хотим в Web действовать по выбору клиента, необходимо перейти на новую страницу (вернуться на Web-сервер), тем самым потеряв то, что мы делали до этого.

Незнание предыдущих действий или того, что клиент выбрал перед тем, как он перешел на данную страницу, затрудняет встраивание бизнес-логики в Web. Предположим, что необходимо написать программу, которая выполняет следующие действия: отображает экран ввода, получает данные, проверяет данные и сохраняет их. Эта последовательность может быть выполнена с помощью простой Cobol-программы, поскольку Cobol использует рабочий раздел памяти, где хранятся все переменные, используемые в программе. А теперь представьте ту же Cobol-программу, но каждый из разделов которой (оператор PERFORM) является отдельной программой! Это в точности отражает работу Web. Иначе говоря, свойство Web не сохранять информацию о своем состоянии означает, что массиванная обработка данных, необходимая для выполнения программы, не может выполняться непосредственно на одной Web-странице; возможности обработки данных клиентским обозревателем ограничены возможностями самого обозревателя и недостатком места для хранения переменных, которые используются всеми страницами Web-сайта.

Необходимо иметь в виду, что основная функция Web-обозревателя — отображение страницы на клиентском компьютере. Обозреватель (в процессе его использования в HTML) не обладает вычислительными возможностями, кроме форматирования текста и получения вводимых полей форм. Даже когда обозреватель получает данные полей формы, невозможно немедленно выполнить проверку данных. Поэтому выполнение таких важных задач на стороне клиента в Web отводится программам на других языках Web-программирования, например, Java, JavaScript и VBScript. Обозреватель больше напоминает неинтеллектуальный терминал, который только отображает данные и выполняет рудиментарные задачи, например, форматирование входных данных. Большая часть обработки данных выполняется на сервере — в данном случае, на сервере Web-приложений.

Чтобы обойти упомянутые проблемы Web-среды, большинство серверов Web-приложений обладают возможностями управления сессией, позволяющими Web-серверу обслуживать статусную информацию о каждой сессии клиента в памяти сервера. Каждый поставщик Web-сервера предлагает собственное решение по обработке такой информации. В примере с сервером приложений ColdFusion переменные сессии декларируются при помощи синтаксиса `<CFSET session.variablename=value>`. Например, чтобы декларировать пользовательский тип переменной сессии, необходимо записать: `<CFSET session.usertype = "ADMIN">`. Затем эта переменная будет



доступна всем страницам в данной сессии клиента. Сессия клиента начинается, когда клиентский обозреватель подключается к Web-серверу, а заканчивается, когда клиент получает доступ к странице вне данного Web-сайта, закрывает свой обозреватель или бездействует какой-то определенный отрезок времени.

Знание и понимание ограничений глобальной Сети (которая по своей природе не сохраняет информацию о своем состоянии) приводит к разработке и использованию альтернативных стратегий обработки данных. Далее мы обсудим, как можно эффективно использовать Web (несмотря на все ее ограничения) для управления ответственными транзакциями, имеющимися в большинстве бизнес-процессов.

### 15.5.7. Ввод данных

В этом разделе будет создана форма для ввода данных в таблицу DEPARTMENT (отдел). В следующем примере таблица DEPARTMENT содержит три поля: идентификатор отдела, описание отдела (обязательное поле) и необязательное поле — идентификатор руководителя отдела, которое ссылается на таблицу USER (сотрудники). Конечно, *если пользователь вводит идентификатор пользователя, то этот идентификатор должен соответствовать идентификатору сотрудника в таблице USER*. Используя эту схему, посмотрим, как можно использовать сервер ColdFusion для установки базовой проверки данных на стороне сервера для нужных полей и как реализовать ввод данных и управление вводом для необязательной связи.

Нам потребуется, по крайней мере, две Web-страницы для выполнения только что поставленных задач. Первая страница, полученная с помощью сценария 15.5a, создает форму для получения данных. Вторая страница, полученная с помощью сценария 15.5b, будет вводить данные в таблицу. Проверка данных будет проводиться на стороне сервера. Прежде всего, посмотрим на листинг 15.5a. Результат его выполнения представлен на рис. 15.13.

#### Листинг 15.5a. Экран ввода данных запроса

```
1: <HTML>
2: <HEAD>
3: <TITLE>Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: <CFQUERY NAME="USRLIST" DATASOURCE="RobCor">
5: SELECT USR_ID, USR_LNAME, USR_FNAME, USR_MNAME
6: FROM USER
7: WHERE USR_ID NOT IN (SELECT USR_ID FROM DEPARTMENT
8: WHERE USR_ID > 0)
9: ORDER BY USR_LNAME, USR_FNAME, USR_MNAME
10: </CFQUERY>
11: </HEAD>
12: <BODY BGCOLOR="LIGHTBLUE">
13: <H1>
14: <CENTER> Insert Query: Data Entry Screen</CENTER>
```

```

14: </H1>
15: <FORM ACTION="chl5-5b.cfm" method="post">
16: <CENTER><H1>DEPARTMENT DATA</H1></CENTER>
17: <!-- следующий код определяет необходимые поля -->
18: <input type="hidden" name="DEPT_ID_required" VALUE="You must enter
a 20 DEPT_ID">
19: <input type="hidden" name="DEPT_DESC_required" VALUE="You must
enter a description">
20: <table align="CENTER" bgcolor="Silver">
21: <tr>
22: <td>
23: <pre>
24: Department ID: <input type="text" name="DEPT_ID" size=10
maxlength=10>

25: Description: <input type="text" name="DEPT_DESC" size=35
maxlength=35>

26: Manager: <SELECT NAME="USR_ID" SIZE=1> <!-- выбор пользователя из
списка -->
27: <OPTION VALUE="">-----
28: <CFOUTPUT Query="USRLIST">
29: <OPTION VALUE="#UsrList.USR_ID#">[#USR_ID#] #USR_LNAME#,
#USR_FNAME#, #USR_MNAME#
30: </CFOUTPUT>
31: </SELECT>
32: </pre>
33: </td>
34: <td>
35: <input type="submit" value="Add Record">
36: </FORM>
37: <FORM ACTION="chl5-0.cfm" Method="post">
38: <input type="submit" value="Main Menu ">
39: </FORM>
40: </td>
41: </tr>
42: </table>
43: </body>
44: </HTML>

```

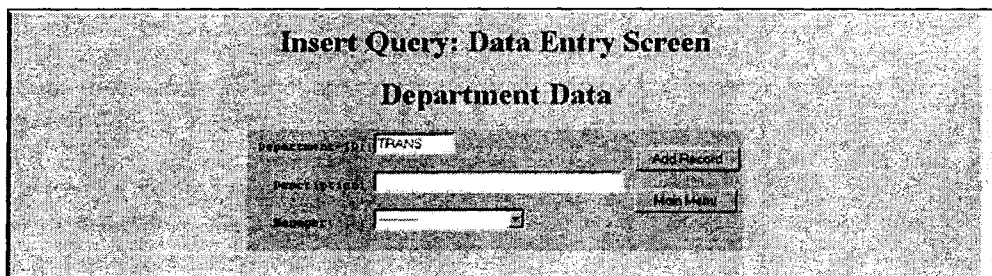


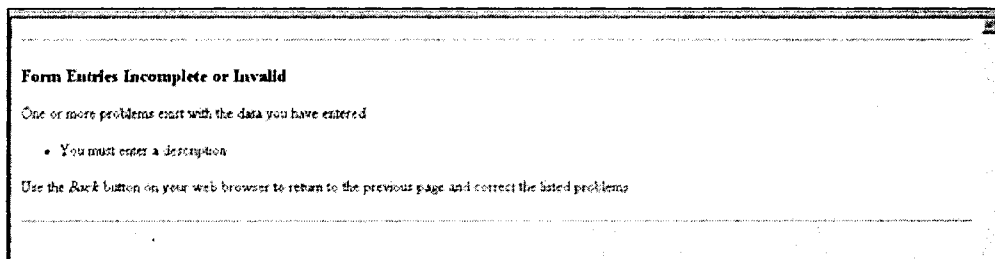
Рис. 15.13. Результат выполнения сценария 15.5a

Листинг 15.5a отображает пользовательскую форму ввода новых данных для отдела. Чтобы представить работу формы, мы добавим новый транспортный отдел (TRANS) и назначим Кима Карлтона (Kim Carlton) руководителем этого отдела. Сценарий 15.5a разработан для выполнения проверки данных в поле USR\_ID с помощью окна запроса и выбора. Для того чтобы увидеть, как сценарий выполняет эти задачи, рассмотрим некоторые наиболее важные строки.

- В строках 4—9 выполняется вложенный запрос поиска всех идентификаторов сотрудников, которые еще не являются руководителями отделов. Выполнение этой части процедуры проверки данных гарантирует отсутствие дублирующих идентификаторов в таблице DEPARTMENT. Поэтому будет выполняться бизнес-правило, в соответствии с которым сотрудник не может руководить более чем одним отделом. Также, поскольку у отдела может и не быть (пока) руководителя, поле USR\_ID может и вообще не иметь значения. Для выполнения необходимой проверки мы начинаем вложенный запрос с условия `USR_ID > 0`. Это условие справедливо только для тех записей, где имеется руководитель (USR\_ID).
- Строки 15—36 определяют форму, предназначенную для ввода данных. Обратите внимание, что сценарий 15.5a будет вызываться, когда пользователь нажмет кнопку **Add Record**.
- Строки 18 и 19 содержат специальные теги, используемые в ColdFusion для выполнения проверки данных на стороне сервера. В данном случае мы проверим ввод для двух обязательных полей — идентификационного кода отдела DEPT\_ID и описания отдела DEPT\_DESC. Эта задача выполняется с помощью тега `<INPUT>` со следующими параметрами:
  - `TYPE = "hidden"`. Этот параметр гарантирует, что поле не будет отображаться на экране;
  - `NAME = "имяполя_required"`. Этот параметр определяет поле, которое необходимо проверить, после имени которого следует слово (опция) `_required`. Другие возможные опции параметра:
    - ◊ `_integer` — для проверки целых значений;
    - ◊ `_date` — для проверки допустимой даты только для формата мм/дд/гг;
    - ◊ `_range` — для проверки нахождения значения в заданном диапазоне. Диапазон задается выражением `VALUE = "значение"`, например: `VALUE = "MIN=10 MAX=20"`;

- **VALUE** = "сообщение об ошибке". Этот параметр содержит сообщение об ошибке, которое должно отображаться, когда нарушается ограничение (в нашем случае, когда поле пустое). Если тип параметра **NAME** задан как **\_range**, это поле содержит максимальное и минимальное значения, используемые для проверки.
- В строках 26—31 создается раскрывающийся список для вывода всех *доступных* сотрудников, которые могут быть выбраны на должность руководителя отдела. Особое внимание обратите на следующие строки:
- в строке 27 определяется необязательное "пустое" значение (**null**), представляемое на экране тонкой линией для указания на то, что у данного отдела нет руководителя. Эта линия используется для реализации необязательности поля **USR\_ID**. Если не включить эту линию, то невозможно будет присвоить пустое значение полю **USR\_ID**. Обратите внимание, что в данном случае это значение = "thin" (тонкая), что означает, что **USR\_ID** будет присвоено пустое значение ("thin");
  - строки 28—30 предназначены для создания списка всех сотрудников, которые могут управлять новым отделом. Имейте в виду, что запрос **USRLIST** возвращает только **USR\_ID** (идентификатор) пользователей, которых уже нет в таблице **DEPARTMENT**. Иначе говоря, запрос **USRLIST** перечисляет только тех сотрудников, *которые могут стать руководителями отделов* (помните, что бизнес-правило утверждает, что сотрудник может руководить только одним отделом).

Когда пользователь нажимает кнопку **Add Record**, форма отсылается Web-серверу на обработку. Там ColdFusion вычисляет необходимые поля, посылает сообщение об ошибке, если одно из обязательных полей пусто (рис. 15.14).



**Рис. 15.14.** Форма ввода запроса; сообщение об ошибке при проверке на стороне сервера

Если проверка на стороне сервера подтверждает корректность данных, выполняется второй сценарий, 15.56, который получает поля формы от сценария 15.5а и использует тег **<CFINSERT>** для добавления записей в базу данных. Как только запись добавлена, сценарий 15.56 представляет конечному пользователю экран для подтверждения.

Листинг 15.56. Экран подтверждения запроса на ввод данных

```
1: <HTML>
2: <HEAD>
3: <TITLE>Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: <!-- Ввод записи в таблицу -->
5: <CFINSERT datasource="RobCor" Tablename= "DEPARTMENT">
6: </HEAD>
7: <BODY BGCOLOR="LIGHTBLUE">
8: <H1>
9: <CENTER>Insert Query: Result Confirmation</CENTER>
10: </H1>
11: You have added the following record:
12:
13: <CFOUTPUT>
14: <PRE>
15: DEPT_ID : #DEPT_ID#
16: DEPT_DESC : #DEPT_DESC#
17: USR_ID : #USR_ID#
18: </PRE>
19:
20: </CFOUTPUT>
21: <FORM ACTION="chl5-0.cfm" Method="post">
22: <input type="submit" value="Main Menu ">
23: </FORM>
24: </BODY>
25: </HTML>
```

Результат выполнения этого сценария представлен на рис. 15.15.

Чтобы увидеть, как тег `<CFINSERT>` используется в сценарии 15.56 для добавления записи в базу данных, взгляните на строку 5. Обратите внимание, что в этом теге есть два параметра:

- ❑ `DATASOURCE = "имяисточникаданных"` — имя подключения к базе данных ODBC;
- ❑ `TABLENAME = "имятаблицы"` — имя таблицы, которую необходимо обновить.

Как тег `<CFINSERT>` узнает, какие поля необходимо вставить в таблицу? И где необходимо получить значения для вставки в столбцы таблицы? Ответ на оба эти вопроса можно найти, исследуя тег `<CFINSERT>` сценария 15.56. В этом теге используются имена полей, определенные в форме, созданной в сценарии 15.5a.

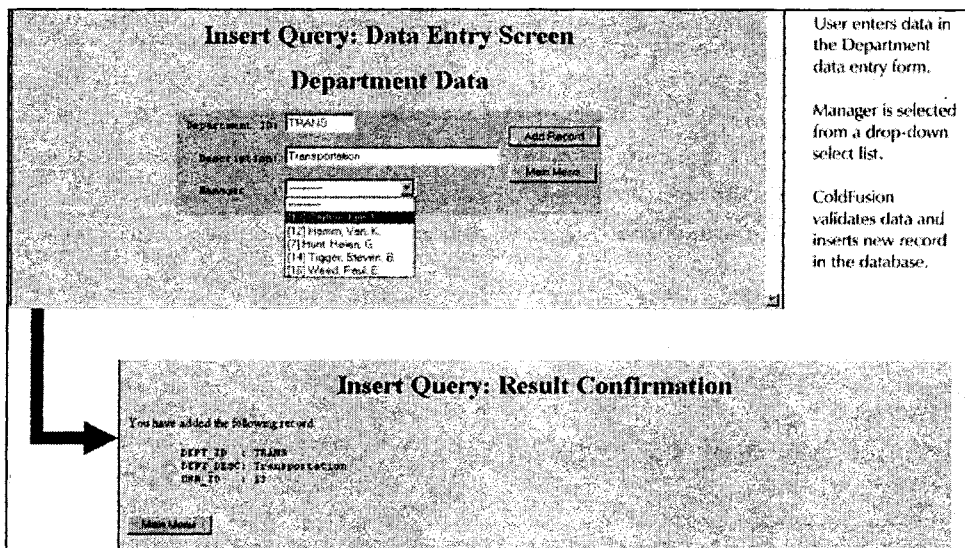


Рис. 15.15. Результат выполнения сценария 15.56

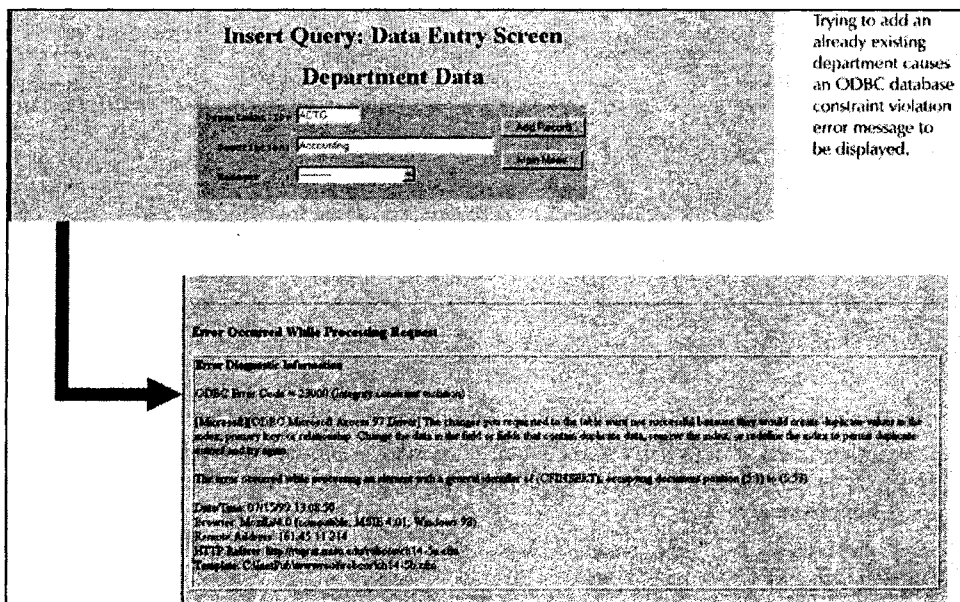


Рис. 15.16. Запрос ввода; сообщение об ошибке нарушения целостности ODBC

Вы должны помнить, что сценарий 15.5a создает форму, в которой содержатся поля DEPT\_ID, DEPT\_DESC и USER\_ID. Тер <CFINSERT> сравнивает имена этих полей с именами столбцов таблицы, чтобы сделать соответствующую вставку. Чтобы

избегать ошибок, имена полей формы, созданной на вызывающей странице, должны совпадать с именами столбцов таблицы.

База данных MS Access гарантирует целостность на уровне сущностей для таблицы DEPARTMENT базы данных RobCor. Поэтому ввод существующего идентификатора отдела в форме ввода автоматически вызовет ошибку нарушения целостности базы данных ODBC, как показано на рис. 15.16 (помните, что целостность на уровне сущностей нарушается, когда дублируется значение первичного ключа).

## 15.8.8. Обновление данных

Для обновления данных требуется несколько страниц. Например, если нужно выполнить простое обновление записей в таблице DEPARTMENT, то процесс обновления потребует трех различных страниц.

- ❑ Первая страница, созданная сценарием 15.6а, позволяет выбрать запись, которую необходимо обновить. Когда пользователь нажимает кнопку **Edit**, вызывается вторая страница, созданная сценарием 15.6б, и значение поля поиска первой страницы передается этой странице. (Чтобы максимально упростить процесс, мы будем использовать первичный ключ DEPT\_ID, чтобы гарантировать уникальность совпадений. Если используется вторичный ключ поиска, то может встретиться более одной записи, и придется использовать дополнительную страницу для выбора одной записи из нескольких для создания уникального совпадения.)
- ❑ На второй странице (сценарий 15.6б) считывается выбранная запись, затем отображается форма ввода данных, чтобы пользователь мог изменить данные. Когда пользователь нажмет кнопку **Update**, эта страница вызывает третий сценарий и передает значения полей второй страницы третьей странице.
- ❑ Третья и последняя страница, созданная сценарием 15.6в, обновит базу данных и выведет подтверждающее сообщение для конечного пользователя.

Хотя этот процесс выглядит достаточно просто, возникает несколько проблем, которые придется в дальнейшем решить. Рассмотрим листинг сценария 15.6а.

### Листинг 15.6а. Запрос обновления. Экран выбора записи

```
1: <HTML>
2: <HEAD>
3: <TITLE>Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: <CFQUERY Name="Deptlist" DataSource="RobCor">
5: SELECT * FROM DEPARTMENT ORDER BY DEPT_ID
6: </CFQUERY>
7: </HEAD>
8: <BODY BGCOLOR="LIGHTBLUE">
9: <H1>
10: <CENTER>UPDATE Query: Record Selection Screen</CENTER>
11: </H1>
```

```

12: <table align="CENTER" bgcolor="Silver"
13: <tr valign="TOP">
14: <td>
15: <FORM ACTION="chl5-6b.cfm" Method="post">
16: <SELECT NAME="DEPT_ID" SIZE=1>
17: <CFOUTPUT QUERY="Deptlist">
18: <OPTION VALUE="#DEPT_ID#">[#DEPT_ID#] - $DEPT_DESC#
19: </CFOUTPUT>
20: </SELECT>
21: <input type=hidden name="DEPT_ID_required" value="DEPT_ID is
 required">
22: </TD>
23: <td>
24: <INPUT TYPE="submit" VALUE=" Edit ">
25: </FORM>
26: </TD>
27: <td>
28: <FORM ACTION="chl5-0.cfm" Method="post">
29: <input type="submit" value="Main Menu ">
30: </FORM>
31: </TD>
32: </TR>
33: </table>
34: </BODY>
35: </HTML>

```

Результат выполнения сценария 15.6a представлен на рис. 15.17.

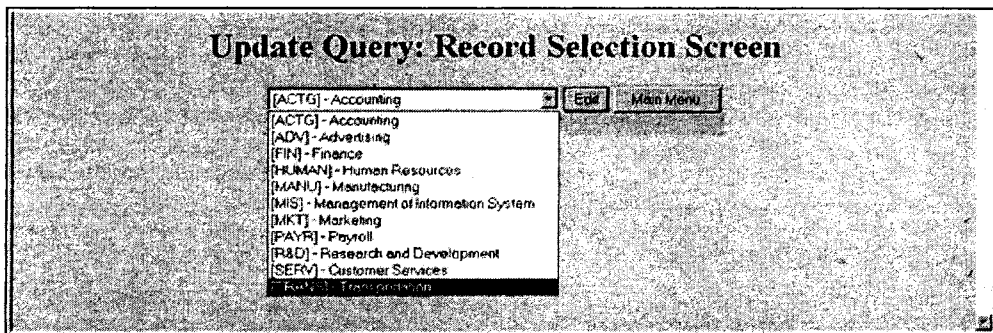


Рис. 15.17. Результат выполнения сценария 15.6a



Сценарий 15.6а создает форму, позволяющую пользователю выбрать запись, которую необходимо обновить. Процесс выбора записи требует выполнения следующих действий.

- ❑ В строках 4—6 выполняется запрос ("Deptlist") для извлечения записей из таблицы DEPARTMENT. Этот набор записей будет позже использован для создания раскрывающегося списка выбора.
- ❑ В строках 17—19 используется тег <CFOUTPUT> для создания списка допустимых опций. В данном примере система ColdFusion использует тег <OPTION> для каждого отдела в запросе "Deptlist".
- ❑ В строках 15—25 создается форма выбора записи. Эта форма использует тег формы SELECT HTML для выбора отдела, который необходимо обновить.
- ❑ В строке 21 определяется поле формы DEPT\_ID для обязательного поля. Это определение гарантирует, что конечный пользователь не получит сообщение об ошибке "variable not defined" ("переменная не определена") при вызове следующей страницы.

Когда пользователь нажимает кнопку **Edit**, вызывается сценарий 15.6б. Этот сценарий получит поле формы DEPT\_ID в качестве параметра, используя его для поиска совпадения в таблице DEPARTMENT. Затем в сценарии подготавливается и отображается форма редактирования данных.

#### Листинг 15.6б Запрос обновления. Экран редактирования записи

```

1: <HTML>
2: <HEAD>
3: <TITLE>Rob 6 Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: </HEAD>
5: <CFQUERY NAME="DeptData" DATASOURCE="RobCor">
6: SELECT * FROM DEPARTMENT WHERE (DEPARTMENT.DEPT_ID
7: = '#form.DEPT_ID#')
8: </CFQUERY>
9: <CFQUERY NAME="USRLIST" DATASOURCE="RobCor">
10: SELECT USR_ID, USR_LNAME, USR_FNAME, USR_MNAME
11: FROM USER
12: WHERE USR_ID NOT IN (SELECT USR_ID FROM DEPARTMENT
13: WHERE USR_ID > 0 AND DEPT_ID <> 'Nform.DEPT_ID#')
14: ORDER BY USR_LNAME, USR_FNAME, USR_MNAME
15: </CFQUERY>
16: </HEAD>
17: <BODY BGCOLOR="LIGHTBLUE">
18: <H1>

```

```

18: <CENTER>UPDATE Query: Edit Record Screen</CENTER>
19: </H1>
20: <FORM ACTION="chl5-6c.cfm" method="post">
21: <table align="CENTER" bgcolor="Silver" bordercolor="Blue"
 bordercolorlight="Aqua"
22: <TR>
23: <TD>
24: <CFOUTPUT Query="DeptData">
25: <PRE>
26: <input type="hidden" name="DEPT ID" value="#form.DEPT ID#">
27: Department ID: #DEPT_ID#

28: Description' : <input type="text" name="DEPT_DESC"
 value="#DEPT_DESC#"size=35 maxlength=35>

29: Manager : <SELECT NAME="USR_ID" SIZE=1><!-- выбор пользователя
 из списка --->
30: <OPTION <CFIF #DeptData.usr_id# EQ "">SELECTED</CFIF> VALUE="">-----
31: </CFOUTPUT>
32: <CFOUTPUT Query="USRLIST">
33: <OPTION <CFIF #DeptData.usr_id# EQ #UsrList .USR_ID#>
 SELECTED</CFIF> VALUE="#UsrList.USR_ID#">[#USR_ID#] #USR_LNAME#,
 #USR_FNAME#, #USR_MNAME#
34: </CFOUTPUT>
35: </SELECT>
36: </pre>
37: </td>
38: <td valign="TOP">
39: <input type="submit" value="Update">
40: </FORM>
41: <FORM ACTION="chl5-0.cfm" Method="post">
42: <input type="submit" value="Main Menu">
43: </FORM>
44: </td>
45: </TR>
46: </table>
47: </BODY>
48: </HTML>

```

Результат выполнения сценария 15.66 представлен на рис. 15.18.

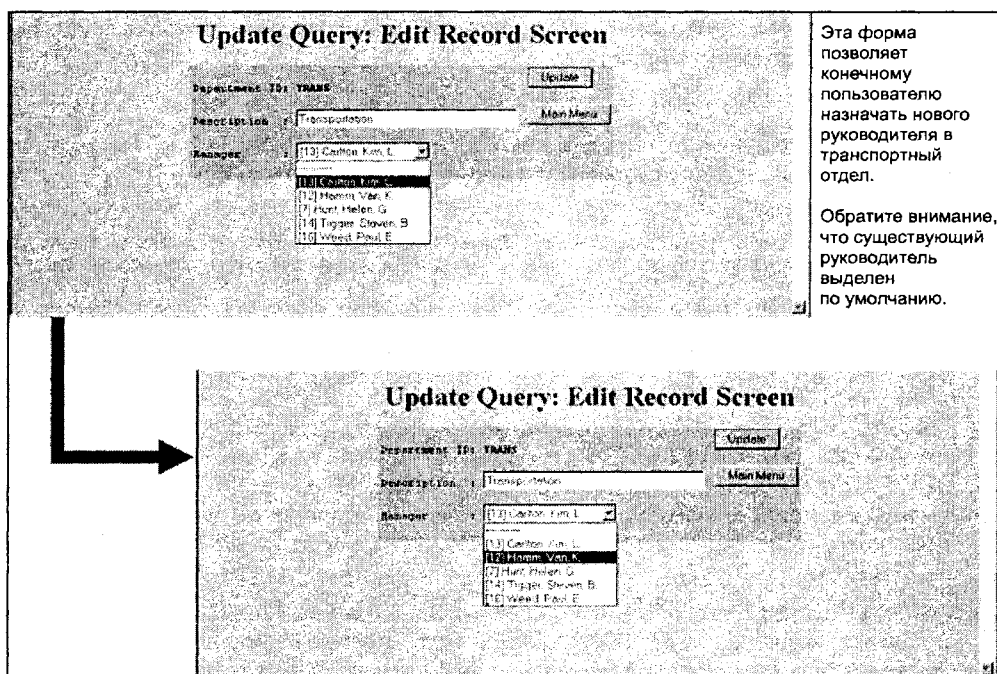


Рис. 15.18. Результат выполнения сценария 15.66

Из листинга сценария 15.66 и рис. 15.19 видно, что сценарий выполняет следующие действия.

- ❑ В строках 5—7 читаются данные таблицы DEPARTMENT с помощью параметра #form.DEPT\_ID#, полученного из сценария 15.6a.
- ❑ В строках 8—14 определяется и выполняется ключевой запрос. Когда вводится информация о новом отделе, для вывода списка всех сотрудников, не являющихся руководителями отделов, используется запрос USRLIST. Этот запрос применяется только для вставки *новой* записи; его нельзя использовать для редактирования имеющихся записей.

Чтобы понять, почему необходимо модифицировать исходный запрос USRLIST, предположим, что `USR_ID=13` является текущим идентификатором руководителя отдела ACTG. Если мы попытаемся *редактировать* запись отдела ACTG с помощью исходного запроса USRLIST, список "доступных сотрудников" будет содержать всех сотрудников, которых уже нет в таблице DEPARTMENT. Поэтому, поскольку руководитель отдела ACTG указан в столбце `USR_ID` таблицы DEPARTMENT, текущий руководитель отдела не появится в списке сотрудников, которые могут руководить отделом ACTG. Иначе говоря, если мы попытаемся редактировать (обновлять) запись с помощью исходного запроса USRLIST, мы будем вынуждены выбирать другого руководителя, а не текущего. А это не входит в наши планы!

Чтобы избежать такой проблемы, вложенный критерий запроса USRLIST нужно изменить так, чтобы исключить "предназначенный для редактирования" идентифи-

катор DEPT\_ID из подзапроса. Поэтому в строке 11 определено, что критерий вложенного запроса должен выглядеть так:

```
WHERE USR_ID > 0 AND DEPT_ID <> '#form.DEPT_ID#'
```

С учетом этой модификации, руководитель отдела ACTG (USR\_ID = 13) появится в списке допустимых идентификаторов сотрудников.

- В строках 20—40 создается пользовательская форма редактирования. Эта форма позволяет конечному пользователю модифицировать только два поля таблицы DEPARTMENT: DEPT\_DESC (описание отдела) и USR\_ID (руководитель отдела).

*Поскольку DEPT\_ID является первичным ключом таблицы, мы не можем позволить конечному пользователю модифицировать его значение.* Причина такого ограничения проста. Предположим, что конечный пользователь редактирует данные по отделу ACTG, т. е. DEPT\_ID = 'ACTG'. Если пользователь в форме DEPT\_ID изменит 'ACTG' на 'ACTNG', то данные по отделу будут переданы со значением поля DEPT\_ID = 'ACTNG' в сценарий обновления. К сожалению, значения DEPT\_ID = 'ACTNG' нет в таблице DEPARTMENT. Поэтому база данных возвратит код ошибки, указывающий на то, что пользователь пытается обновить запись, которой не существует, так это и есть на самом деле: 'ACTG' существует, а 'ACTNG' нет.

- В строке 26 создается переменная формы ввода с именем DEPT\_ID, которой присваивается значение '#DEPTDATA.DEPT\_ID#'. Другими словами, этот сценарий переходит в режим редактирования текущей записи. Обратите внимание, что эта переменная скрытая, поэтому на экране она не видна. Присвоение значения этой переменной гарантирует, что значение DEPT\_ID будет передано в сценарий 15.6b (помните, что все переменные формы, определенные с помощью тегов <INPUT> или <SELECT> формы, передаются в вызванную программу).
- Строка 27 обеспечивает представление текущего идентификатора отдела на экране. Обратите внимание, что конечный пользователь не может изменять это значение.
- В строке 28 конечному пользователю позволяет модифицировать описание отдела. Обратите внимание, что тег <INPUT> устанавливает "#DEPT\_DESC#" как значение по умолчанию для этого поля, тем самым гарантируя, что здесь будет отображено предыдущее содержимое поля. Пользователь может изменить отображенную информацию.
- В строках 29—35 конечному пользователю разрешается выбрать руководителя для данного отдела. Эти строки будут создавать список выбора, в котором перечисляются допустимые варианты для поля руководителя:
  - все сотрудники, которые не являются руководителями;
  - если у отдела есть руководитель, то отображается данный руководитель;
  - пустое значение, означающее, что для данного отдела руководитель пока не назначен.

С учетом этих вариантов пользователь может выбрать пустое значение поля руководителя, тем самым оставив текущего руководителя без изменения, или же выбрать другого руководителя. Если в редактируемом отделе уже имеется руково-

дитель, то имя этого руководителя должно по умолчанию появляться с параметром **SELECTED**.

- ❑ В строке 30 разрешается присвоить идентификатору руководителя пустое значение (обратите внимание, что **VALUE = ""**). В этой строке также имеется тег **<CFIF>** для оценки текущего значения поля отдела **USR\_ID**. Если значение **USR\_ID** пусто (""), т. е. в выбранном отделе нет руководителя, то это пустое значение и будет выбранным вариантом **SELECTED**. В противном случае этот вариант появится в списке вариантов, но не будет выбран по умолчанию. Наличие пустого значения гарантирует, что имеющийся руководитель отдела может быть удален.
- ❑ В строках 32—34 для создания ввода **OPTION** по каждому идентификатору сотрудника в запросе "USRLIST" используется тег **<CFOUTPUT>** (помните, что тег **<CFOUTPUT>** проверяет в цикле каждую запись в именованном запросе). Для каждой добавленной опции тег **<CFIF>** сравнивает имеющийся **USR\_ID** в таблице **DEPARTMENT (#DEPTDATA.USR\_ID#)** с добавленным **USR\_ID (#USRLIST.USR\_ID#)**. Если эти значения совпадают, то в тег **<OPTION>** добавляется ключевое слово **"SELECTED"**.

Когда пользователь нажмет кнопку **Update**, сценарий 15.6в вызывает сценарий 15.6в, передавая туда значения переменных.

#### Листинг 15.6в. Запрос обновления. Экран подтверждения

```

1: <HTML>
2: <HEAD><TITLE>Rob & Coronel - Chapter 15: Cold Fusion
 Examples</TITLE>
3: <CFUPDATE datasource="RobCor" Tablename="Department">
4: </HEAD>
5: <BODY BGCOLOR="LIGHTBLUE">
6: <H1>
7: <CENTER>UPDATE Query: Update Result Screen</BX/CENTER>
8: </H1>
9: <CFOUTPUT>
10: You have successfully Updated the following data
11: <PRE>
12: DEPARTMENT ID: #DEPT_ID#
13: DESCRIPTION: #DEPT_DESCf
14: MANAGER: #USR_ID#
15: </PRE>
16: <CFOUTPUT>
17: <FORM ACTION="chl5-Q.cfm" Method="post">
18: <input type="submit" value="Main Menu ">
19: </FORM>
20: </BODY>
21: </HTML>

```

Результат выполнения сценария 15.6в представлен на рис. 15.19.

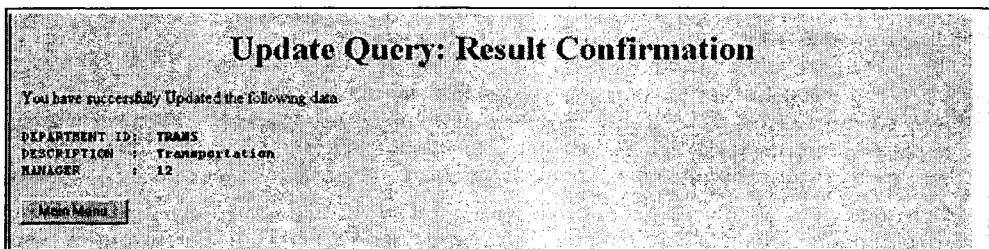


Рис. 15.19. Результат выполнения сценария 15.6в

В сценарии 15.6в для обновления таблицы DEPARTMENT используется тег <CFUPDATE>. Его параметры:

☐ DATASOURCE = "имя\_источника\_данных" — имя подключения к базе данных ODBC;

☐ TABLENAME = "имя\_таблицы" — имя таблицы, которую необходимо обновить.

В теге <CFUPDATE> для обновления именованной таблицы используются поля формы, переданные из вызывающей страницы (DEPT\_ID, DEPT\_DESC и USER\_ID). Как и в других cfm-страницах, в вызывающей странице необходимо именовать поля формы в соответствии с именами столбцов таблицы. Если не выполнить это соглашение об именовании, то это приведет к появлению сообщения об ошибке "variable not found" ("переменная не найдена").

Большинство основных процессов обработки Web-информации требуют, по крайней мере, трех действий: создание новой записи, модификация имеющейся записи и удаление записи. Мы обсудили первые две из этих операций, теперь исследуем третью — удаление.

### 15.5.9. Удаление данных

Запрос на удаление, который мы рассмотрим в этом разделе, позволяет удалять данные об отделе. Как и в случае с запросом обновления, запрос на удаление потребует создания трех страниц.

☐ Первая страница (листинг 15.7а) позволяет выбрать запись, которую необходимо удалить. Когда пользователь нажмет кнопку **Delete** (удалить), инициируется сценарий 15.7б и в него передается значение поля формы DEPT\_ID.

☐ Вторая страница (листинг 15.7б) считывает выбранную запись и отображает данные на экране. Этот запрос также выполняет проверку целостности на уровне ссылок, исключая возможность удаления отделов, в которых содержатся пользователи. Когда пользователь нажмет кнопку **Delete**, эта страница вызовет третью страницу, передавая ей значения полей формы DEPT\_ID.

- ❑ Последняя страница (листинг 15.7в) удаляет строку отдела из базы данных на основе значения поля формы DEPT\_ID, переданного из вызывающей программы (15.7б).

Рассмотрим первый из этих трех экранов.

**Листинг 15.7а. Экран выбора запроса на удаление записи**

```
1: <HTML>
2: <HEAD>
3: <TITLE>Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: <CFQUERY Name="Deptlist" DataSource="RobCor">
5: SELECT * FROM DEPARTMENT ORDER BY DEPT_ID
6: </CFQUERY>
7: </HEAD>
8: <BODY BGCOLOR="LIGHTBLUE">
9: <H1>
10: <CENTER>DELETE Query: Record Selection Screen</CENTER>
11: </H1>
12: <table align="CENTER" bgcolor="Silver">
13: <tr valign="TOP">
14: <td>
15: <FORM ACTION="chl5-7b.cfm" Method="post">
16: <SELECT NAME="DEPT_ID" SIZE=1>
17: <CFOUTPUT QUERY="Deptlisi">
18: <OPTION VALUE="#DEPT_ID#">[#DEPT_ID#] - #DEPT_DESC#
19: </CFOUTPUT>
20: </SELECT>
21: <input type=hidden name="DEPT_ID_required" value="DEPT_ID is
 required">
22: </TD>
23: <td>
24: <INPUT TYPE="submit" VALUE="Delete">
25: </FORM>
26: </TD>
27: <td>
28: <FORM ACTION="chl5-0.cfm" Method="post">
29: <input type="submit" value="Main Menu">
30: </FORM>
31: </TD>
```

```

32: </TR>
33: </table>
34: </BODY>
35: </HTML>

```

Результат выполнения сценария 15.7a приведен на рис. 15.20.

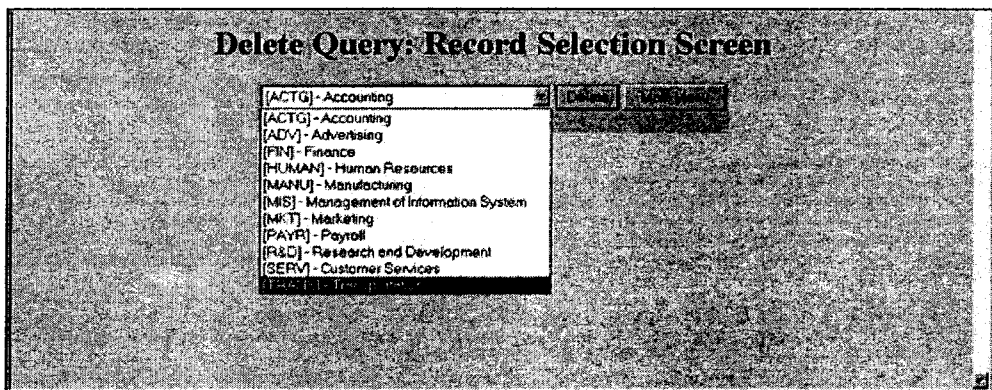


Рис. 15.20. Результат выполнения сценария 15.7a

В сценарии 15.7a пользователь может выбрать запись, которую необходимо удалить. Для того чтобы проследить эти операции, рассмотрим следующие строки.

- ☐ В строках 4—6 выполняется запрос ("Deptlist") для извлечения всех записей из таблицы DEPARTMENT. Результат запроса будет использоваться для отображения формы выбора записей.
- ☐ В строках 15—25 определяется форма выбора записи. Когда пользователь нажимает кнопку **Delete**, сценарий 15.7a вызывает сценарий, представленный в листинге 15.7б, и передает ему значение поля формы DEPT\_ID.
- ☐ В строках 16—20 создается элемент управления формой SELECT.
- ☐ В строках 17—19 используется тег <CFOUTPUT> для динамического создания списка OPTION.
- ☐ В строке 21 используется тег <INPUT> для определения поля DEPT\_ID в качестве обязательного поля. Эта команда будет использоваться системой ColdFusion для выполнения проверки данного поля на стороне сервера. Если это поле оставить пустым, ColdFusion вернет текст сообщения об ошибке, заданный в параметре VALUE. Поэтому в строке 21 гарантируется, что пользователь выбирает запись перед тем, как пытаться удалить ее (вы не можете удалить запись, предварительно ее не выбрав). Если пользователь, не выбирая записи, нажмет кнопку **Delete**, то возникнет сообщение об ошибке базы данных ODBC, указывающее на попытку удаления несуществующей записи — или, хуже того, при этом могут быть удалены все записи в таблице!



Второй сценарий (ch15-7b.cfm) выполняет две важные функции.

- ❑ Он считывает запись, которую необходимо удалить, и представляет данные на экране, чтобы пользователь мог подтвердить запись, которую необходимо удалить.
- ❑ Она выполняет проверку целостности на уровне ссылок. Помните, что таблицы DEPARTMENT и USER обслуживают связь 1:М, которую можно выразить как "в каждом отделе имеется один или более сотрудников". Поэтому пользователь не может удалить отдел, в котором имеются сотрудники.

С учетом вышеизложенного, рассмотрим подробнее сценарий 15.76.

#### Листинг 15.76. Запрос удаления. Экран представления записи

```

1: <HTML>
2: <HEAD>
3: <TITLE>Rob & Coronel - Chapter 15: Cold Fusion Examples</TITLE>
4: </HEAD>
5: <CFQUERY NAME="DeptData" DATASOURCE="RobCor">
6: SELECT * FROM DEPARTMENT WHERE
 (DEPARTMENT.DEPT_ID='#form.DEPT_ID#')
7: </CFQUERY>
8: <CFIF #deptdata.usr_id# is not "">
9: <CFQUERY NAME="Usrdata" DATASOURCE="RobCor">
10: SELECT USR_ID, USR_LNAME, USR_FNAME, USR_MNAME FROM USER
11: WHERE (USER.USR_ID = #deptdata .usr_id#)
12: </CFQUERY>
13: </CFIF>
14: <CFQUERY NAME="UsrTot" DATASOURCE = "RobCor">
15: SELECT COUNT(*) AS T1 FROM USER
16: WHERE (USER.DEPT_ID = '#form.DEPT_ID#')
17: </CFQUERY>
18: </HEAD>
19: <BODY BGCOLOR="LIGHTBLUE">
20: <H1>
21: <CENTER>DELETE Query: Show Record Screen</CENTER>
22: </H1>
23: <FORM ACTION="ch15-7c.cfm" method="post">
24: <CFOUTPUT Query="DeptData">
25: <input type="hidden" name="DEPT_ID" value="#deptdata.DEPT_ID#">
26: <input type="hidden" name="DEPT_DESC"
 value="#deptdata.DEPT_DESC#">

```

```

27: <input type="hidden" name="USR_ID" value="#deptdata.USR_ID#">
28: <table align="CENTER" bgcolor="Silver" bordercolor="Blue"
 bordercolorlight="Aqua" bordercolordark="Black">
29: <TR>
30: <TD>
31: <pre>
32: Department ID: #DEPT_ID#

33: Description : #DEPT_DESC#

34: Manage : <CFIF #deptdata.set_id# is not "">
 #Usrdata.USR_LNAME# #Usrdata.USR_FNAME# #Usrdata.USR_MNAME#</CFIF>
35: </CFOTPUT>
36: </pre>
37: </td>
38: <td valign="TOP">
39:
40: <CFIF #usrtot.tl# EQ 0>
41: <input type="submit" value=" Delete ">
42: <CFELSE>
43: <small>
44: We cannot delete this record

45: because there are dependent
users assigned to this
 department
46: </small>
47: </CFIF>
48: </FORM>
49: <FORM ACTION="chl5-0.cfm" Method="post">
50: <input type="submit" value="Main Menu">
51: </FORM>
52: </td>
53: </TR>
54: </table>
55: </BODY>
56: </HTML>

```

Результат выполнения этого сценария представлен на рис. 15.21.

Исследуем сценарий 15.76 подробнее, чтобы понять, как он работает.

- В строках 5—7 используется тег <CFQUERY> для считывания данных по выбранному отделу. Запрос использует поле формы #form.DEPT\_ID#, переданное из сценария 15.7a.

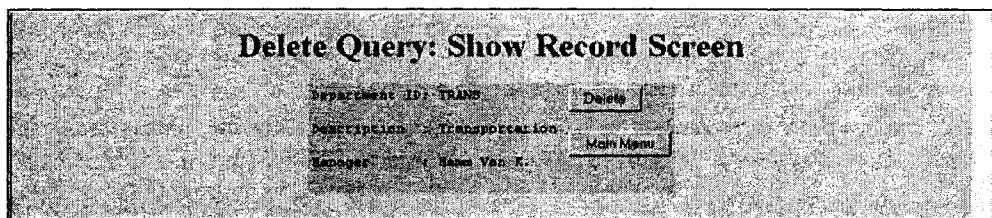


Рис. 15.21. Результат выполнения сценария 15.76

- ❑ В строках 8—13 из таблицы USER извлекаются данные о руководителе отдела. Поскольку это необязательное поле, мы, прежде всего, проверяем, не является ли поле идентификатора сотрудника пустым. Если это так, то данные о сотруднике считываются из таблицы USER с помощью значения #deptdata.usr\_id#. Если идентификатор сотрудника имеет пустое значение, то считывать данные нет необходимости.
- ❑ В строках 14—17 выполняется проверка целостности на уровне ссылок. Этот процесс начинается с выполнения запроса проверки факта, имеются ли сотрудники в отделе, который намечено удалить. Обратите внимание, что SQL-запрос в строках 15 и 16 рассчитывает число сотрудников, принадлежащих отделу (если рассчитанное значение больше нуля, это означает, что в отделе имеется, по крайней мере, один сотрудник). Расчетное значение хранится в переменной T1.
- ❑ В строках 23—47 определяется форма для отображения данных по отделу и подтверждения удаления записи. Когда пользователь нажимает кнопку **Delete**, вызывается сценарий 15.7в и ему передаются три переменные — DEPT\_ID, DEPT\_DESC и USR\_ID.
- ❑ В строке 25 поле DEPT\_ID формы определяется как скрытое ("hidden") и этому скрытому полю присваивается значение DEPT\_ID таблицы DEPARTMENT (хотя это скрытое поле ввода не видно на экране, оно передается на следующий экран).
- ❑ В строках 26 и 27 выполняются такие же действия, как в строке 25, т. е. оставшиеся поля формы (DEPT\_DESC и USER\_ID) определяются как скрытые ("hidden") и этим скрытым полям формы присваиваются соответствующие значения полей DEPARTMENT. Значения скрытых полей формы будут также передаваться в следующий сценарий.
- ❑ В строках 32—35 отображаются данные по отделу для записи, которую необходимо удалить. Это позволяет пользователю увидеть запись, чтобы он мог подтвердить, что именно эту запись собираются удалить. Обратите внимание, что поля, определенные в строках 32 и 33, используют источник "Deptdata", как определено в строке 24 тегом <CFOUTPUT>. В отличие от этого в строке 34 префикс имени поля указывает, что здесь используются поля из запроса "Usrdata".
- ❑ В строке 34 используется тег <CFIF> для проверки, имеются ли данные о сотруднике. Если #deptdata.usr\_id# имеет непустое значение, то данные о сотруднике выводятся на экран.
- ❑ В строках 39—46 проверяется, имеются ли в отделе сотрудники. Если записей (строк) сотрудников нет (#usrtot.t1 = zero#), то выводится кнопка **Delete** (еще раз

обратите внимание на строку 15, где указано условие проверки). Если значение не пустое, то в форме выводится сообщение о том, что в данном отделе имеются сотрудники, и кнопка **Delete** не выводится.

Если запись может быть удалена и пользователь нажал кнопку **Delete**, вызывается сценарий, представленный в листинге 15.7в, и ему передаются поля формы (скрытые).

#### Листинг 15.7в. Запрос на удаление. Экран подтверждения

```

1: <HTML>
2: <HEAD><TITLE>Rob & Coronel - Chapter 15: Cold Fusion
 Examples</TITLE>
3: <CFQUERY NAME="DeleteDept" Datasource="RDbCor">
4: DELETE FROM DEPARTMENT WHERE (DEPT_ID = '#FORM.DEPT_ID#')
5: </CFQUERY>
6: </HEAD>
7: <BODY BGCOLOR="LIGHTBLUE">
8: <H1>
9: <CENTER>DELETE Query: Delete Result Screen</CENTER>
10: </H1>
11: <CFOUTPUT>
12: You have successfully deleted the following data
13: <PRE>
14: DEPARTMENT ID: #DEPT_ID#
15: DESCRIPTION: #DEPT_DESC#
16: MANAGER : #USR_ID#
17: </PRE>
18: </CFOUTPUT>
19: <FORM ACTION="chl5-0.cfm" Method="post">
20: <input type="submit" value="Main Menu" >
21: </FORM>
22: </BODY>
23: </HTML>

```

Результат выполнения сценария 15.7в представлен на рис. 15.22

Сценарий 15.7в удаляет запись отдела из базы данных и отображает экран подтверждения. Рассмотрим, как эта задача решается в сценарии.

- В строке 3—5 выполняется запрос, который удаляет запись отдела из базы данных. Этот запрос выполняет SQL-оператор "Delete" с помощью поля формы DEPT\_ID, полученного от сценария 15.7б.

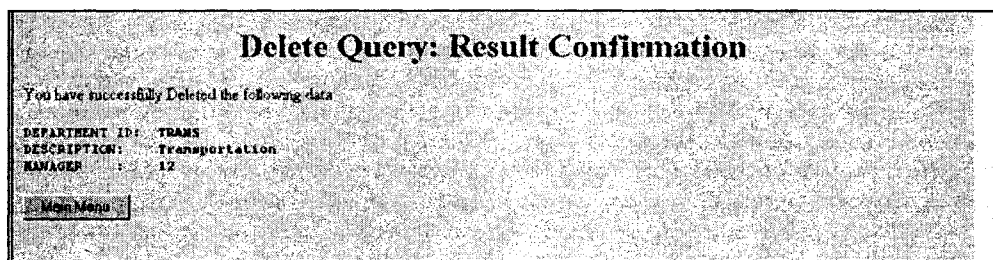
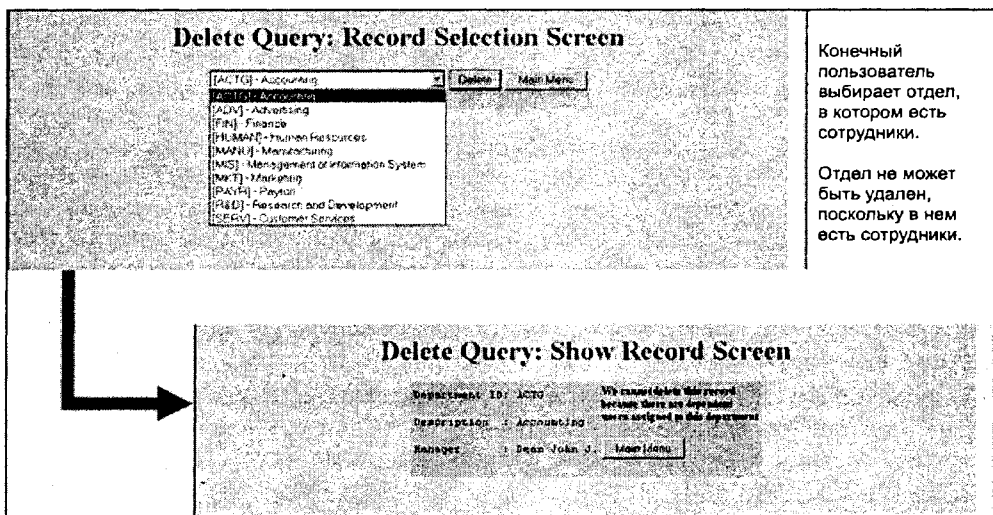


Рис. 15.22. Результат выполнения сценария 15.7в

- В строках 9—16 выполняется вывод подтверждения об удалении данных и отображаются удаленные данные.

На рис. 15.23 показан процесс удаления. На рисунке видно, что попытка удаления отдела, в котором все еще имеются сотрудники, потерпела неудачу. Также видно, что кнопка **Delete** не отображается на втором экране, поскольку отображать ее нет смысла, если данный отдел невозможно удалить.



Конечный пользователь выбирает отдел, в котором есть сотрудники.

Отдел не может быть удален, поскольку в нем есть сотрудники.

Рис. 15.23. Проверка удаления записи

### Примечание

Технология ColdFusion, описанная в этой главе, представляет собой лишь верхушку айсберга разработок Web-приложений, связанных с базами данных. Во время написания этой книги в ColdFusion предлагались сотни дополнительных тегов и функций, помогающих при профессиональной разработке Web-приложений. Хотя предыдущие примеры не претендуют на полноту, они, тем не менее, дают наглядное представление о мощи Web-интерфейса и его гибкости.

В нашу эпоху бурного развития коммерческих Web-приложений вряд ли можно обойтись без баз данных, связанных с Web. Учитывая очевидные конкурентные преимущества, предоставляемые доступом к базам данных через Web, очень заманчиво при разработке Web-БД уделить основное внимание именно Web-компоненту. Еще раз необходимо подчеркнуть, что Web-интерфейс к плохо спроектированной БД может привести к непоправимым последствиям. С другой стороны, хороший проект и реализация БД вкупе с хорошо проработанной техникой Web могут обеспечить большие тактические и стратегические преимущества, неограниченные профессиональные возможности и удовлетворение от проделанной работы.

## 15.6. Системы баз данных Интернета: замечания

Системы баз данных Интернета — это не только разработка Web-приложений с возможностью доступа к БД. Web-интерфейс также используется в качестве шлюза к корпоративным и учрежденческим базам данных. При этом приходится сталкиваться, например, с проблемами безопасности данных, управления транзакциями, проверки данных на стороне сервера и с множеством других организационных и управленческих проблем. Хотя многие из этих задач были подробно рассмотрены в предыдущих главах, условия Web-среды, которые требуют пересмотра некоторых подходов.

Идет ли разговор о базах данных в традиционных клиент/серверных средах или о новейшей среде Интернета, в любом случае разработка систем баз данных требует высококачественного дизайна и реализации. Система базы данных должна существовать в безопасной среде, должным образом разработанной для обслуживания хорошо защищенного доступа к данным, надежного управления транзакциями с поддержкой целостности данных и возможностью их восстановления. Наконец, с точки зрения конечных пользователей и управленцев база данных не имеет никакой практической ценности, если ее пользовательский интерфейс не является дружественным пользователю, информационно емким и не обладает всей полнотой поддержки транзакций пользователя.

Проектирование рабочей базы данных и информационного хранилища не подвержено влиянию (на концептуальном уровне) изменений со стороны традиционных клиент/серверных и интернет-клиент/серверных инфраструктур. Поэтому нет необходимости пересматривать основные процессы и процедуры проектирования этих баз данных. Однако разработка приложений баз данных Интернета сильно отличается от разработки традиционных клиент/серверных приложений. С учетом массового распространения Интернета такие проблемы разработки, как безопасность, резервное копирование и объемы транзакций, становятся даже более важными, чем они были в традиционных средах.

Очевидно, что одновременный доступ к данным от множества гетерогенных клиентов оказывает влияние на определение транзакций и управление ими. Поддержка множества источников данных и их типов, независимость от платформ, переносимость, распределение процессов и масштабируемость, а также открытые стандарты оказывают большое влияние на разработку, установку и управление приложениями.

Конечно же, Интернет оказал самое значительное влияние на разработку приложений баз данных. Свойства Интернета (в особенности Web) радикально изменили способ функционирования приложений. Глобальная Сеть, которая по своей природе не сохраняет информацию о состоянии (stateless system), сильно повлияла на представление и выполнение запросов к БД. Только представьте себе модель запросов/ответов Web и оцените ее отличие от традиционного представления программистов.

Если системы баз данных разработаны и управляются корректно, то администратор должен понимать среду интернет-бизнеса, чтобы успешно справляться с проблемами разработки, использования и управления интерфейсом Web-БД. Поскольку все начинается и заканчивается информацией, мы начнем с рассмотрения разнообразных типов данных, которые поддерживаются в Интернете.

### 15.6.1. Какие типы данных поддерживаются?

Web-разработка требует одновременного управления многими, достаточно отличающимися друг от друга типами данных. Обычно традиционные базы данных поддерживают такие типы данных, как юлианская дата, различные типы числовых данных (целый, с плавающей точкой, денежный) и текстовых (фиксированной и переменной длины). Большинство передовых производителей СУРБД поддерживают в традиционных БД расширенный набор типов данных, например, двоичный и OLE-объекты.

Поскольку интерактивные Web-сайты имеют тенденцию к интегрированию данных из различных источников (текстовые процессоры, электронные таблицы, презентации, изображения, фильмы, аудио и даже голографические изображения), некоторые проектировщики Web-БД используют расширенные типы данных для хранения компонентов страниц (в двоичном формате), которые впоследствии встраиваются в Web-страницы. Хотя страничная компоновка может обеспечить лучшую организацию данных с точки зрения БД, здесь имеются свои проблемы.

- ☐ Каким образом хранить и извлекать такие объекты данных, как документы, изображения и фильмы с помощью Web-обозревателя? Вспомните, что Web-клиент ожидает, что каждый компонент страницы является файлом, хранимым в каталоге Web-сервера. Поэтому СУБД и ППО Web-БД должны предоставлять специальные функции или подпрограммы, позволяющие динамически извлекать объекты из полей баз данных в каталог Web-сервера и наоборот.
- ☐ Как повлияет на общую нагрузку системы хранение в БД двоичных объектов? Насколько надежной должна быть СУБД для обработки транзакций двоичных объектов? Каковы ограничения по расширенным (или OLE) типам данных? Сколько полей расширенных (или OLE) типов данных может иметься в таблице?
- ☐ Поддерживает ли клиентский обозреватель тип данных объекта, доступ к которому требуется получить? Доступны ли необходимые подключаемые модули (plug-ins)? Имеется ли способ перевода документов из их естественного формата в формат HTML? Например, презентацию PowerPoint можно просматривать в Internet Explorer с помощью подключаемого модуля PowerPoint Viewer, а в Netscape Navigator это сделать нельзя.
- ☐ Наконец, хранение изображений или мультимедийных презентаций в базе данных может очень быстро увеличить размер БД. Поддерживает ли СУБД очень

большие базы данных? А как насчет скорости транзакций? Параллельный ввод данных и извлечение двоичных объектов в полях БД может оказать сильное влияние на производительность транзакций БД. Сколько пользователей собираются получить доступ к БД? Как часто они собираются это делать?

Проектирование интерфейса Web-БД должно устранить все эти проблемы и помочь найти верное решение, чтобы база данных не стала узким местом Web-системы.

## 15.6.2. Безопасность данных

Ранее отмечалось, что безопасность — одна из ключевых проблем при доступе к базе данных через Интернет. Большинство поставщиков СУБД предоставляют интерфейсы управления безопасностью базы данных. При создании Web-интерфейса БД безопасность может быть реализована на Web-сервере, в базе данных или в инфраструктуре сети. Во многих случаях создание нескольких брандмауэров является основой безопасности баз данных Интернета.

На уровне Web-сервера большинство Web-клиентов и серверов могут выполнять безопасные транзакции с помощью программ шифрования на уровне протокола TCP/IP. Клиенты и серверы могут обмениваться сертификатами безопасности для того, чтобы удостовериться, что клиенты и серверы — именно те, за кого они себя выдают. Поэтому нужно обеспечить необходимую регистрацию клиентов и серверов и наличие совместимых протоколов шифрования. Брандмауэр гарантирует, что за пределы компании будет отправляться только авторизованная информация.

Все поставщики СУБД предоставляют определенные механизмы безопасности на уровне базы данных, обеспечивая некоторые формы аутентификации пользователей, пытающихся получить доступ к базе данных. На уровне SQL для установки ограничений доступа к таблицам или специфическим SQL-командам администраторы могут использовать команды GRANT и REVOKE.

Поставщики ППО Web-БД обычно предлагают несколько механизмов безопасности, доступных в интерфейсе с базой данных. Например, при использовании источника данных ODBC администратор может ограничить доступ пользователей к некоторым операторам SQL, таким как SELECT, UPDATE, INSERT или DELETE, или к некоторым комбинациям таких операторов. И хотя Web-страницы работают по схеме запрос/ответ, использование Web-интерфейсов не мешает создавать алгоритмы, гарантирующие целостность данных и целостность на уровне ссылок. Мероприятия по обеспечению безопасности данных должны также включать журналы регистрации, связанные с действиями пользователей. Такие журналы гарантируют, что каждое обновление БД напрямую связано с авторизованным пользователем.

Безопасность должна включать в себя расширенную поддержку e-коммерции. Такая поддержка обеспечит возможность Web-сайта выполнять безопасные транзакции в Интернете. Предположим, вы хотите сделать заказ по Интернету с помощью кредитной карты. Процесс оформления заказа, связанный с обработкой данных рабочей БД, должен иметь надежный механизм обеспечения безопасности при обработке информации о кредитной карте. Кроме того, транзакция заказа должна обеспечивать безопасное взаимодействие с несколькими сайтами (дистрибьюторы и банки), гарантируя невозможность преднамеренного искажения информации или ее хищения.



### 15.6.3. Управление транзакциями

Хотя все предыдущие замечания направлены на проблемы управления транзакциями, связанные с успешным ведением e-коммерции, концепция транзакций базы данных не является внутренним делом Web. Вспомните, что модель "запрос/ответ" глобальной Сети предполагает взаимодействие Web-клиента и Web-сервера с помощью коротких сообщений. Эти сообщения ограничены запросом на доставку страниц и их компонентов (компоненты страницы могут включать в себя рисунки, файлы мультимедиа и т. д.). Противоречие Web-модели "запрос/ответ" состоит в следующем:

- ☐ Web не может поддерживать открытую линию связи между клиентом и сервером баз данных;
- ☐ механизм восстановления поврежденных и противоречивых транзакций БД *требует*, чтобы клиент поддерживал открытую линию связи с сервером баз данных.

Очевидно, что создание Web-приложений, зависящих от целевого назначения, обязательно требует поддержки управления транзакциями БД. Принимая во внимание только что описанное противоречие, *проектировщики должны обеспечить поддержку управления транзакциями на уровне сервера баз данных.*

Многие продукты ППО Web-БД обеспечивают поддержку управления транзакциями. Например, ColdFusion обеспечивает такую поддержку с помощью тега <CFTRANSACTION>. Если нагрузка транзакции очень велика, эту функцию можно возложить на независимый компьютер. При таком подходе Web-приложение и серверы баз данных высвобождаются для решения других задач, и вся нагрузка транзакции распределяется между несколькими процессорами.

### 15.6.4. Денормализация таблиц базы данных

Когда для взаимодействия с БД используется Web, проектировщики приложений должны иметь в виду, что Web-формы не могут использовать несколько строк записей данных, что типично для связей 1:M. А такие связи имеют большое значение в e-коммерции. Например, представьте себе заказ и строку заказа или счет-фактуру и строку счета-фактуры. Большинство пользователей знакомы с традиционными формами ввода, представленными в виде графического интерфейса (GUI), которые поддерживают многотабличные данные с помощью многокомпонентных структур, состоящих из главной формы и вложенных форм. Используя такие вложенные формы, конечный пользователь может заносить множество покупок с помощью единственного счета-фактуры. Все данные вводятся на одном экране.

К сожалению, Web-среда не поддерживает такую форму ввода информации. Все примеры, иллюстрирующие работу системы ColdFusion, показывают, что Web может легко обрабатывать ввод данных в одной таблице. Однако иногда необходимо выполнить ввод или обновление информации в нескольких таблицах (например, заказ и строки заказа, счет-фактура и строки счета-фактуры, бронирование номеров с несколькими строками и т. д.), в этом случае Web бессильна. Хотя реализация ввода данных предок/потомок в Web возможна, предлагаемые варианты реализаций не очень удобны, использовать их непросто и они зачастую становятся причинами ошибок.

Чтобы увидеть, как Web-разработчики работают с вводом данных по схеме предок/потомок, рассмотрим обработку отношения ORDER (заказ) и ORDER\_LINE

(строка заказа) при хранении заказов клиентов. Используя ППО Web-БД, например, ColdFusion, создавая Web-интерфейс для обновления заказов, можно использовать различные технические приемы:

- разработка системы HTML-фреймов (кадров), разделяющих экран на заголовок заказа и детализирующие строки. Дополнительный фрейм можно использовать для статусной информации или навигационного меню;
- использование рекурсивных обращений к страницам для обновления и отображения последних элементов, добавленных в заказ;
- создание временных таблиц или серверных массивов для хранения данных дочерних таблиц во время режима ввода данных. Эта технология обычно основана на восходящем подходе, при котором пользователь сначала выбирает товары, которые он хочет заказать. Когда выбор товара завершен, вводятся данные, связанные с заказом товара, например, идентификатор клиента, информация по доставке, сведения о кредитной карте. При такой технологии сведения о заказе хранятся во временных таблицах или массивах;
- использование хранимых процедур или триггеров для перемещения данных из временных таблиц или массивов в основные таблицы.

Хотя собственно Web не поддерживает напрямую ввод данных по схеме предок/потомок, для создания необходимого Web-интерфейса можно прибегнуть к языкам программирования Web, таким как Java, JavaScript или VBScript. Недостаток такого подхода — необходимость дополнительного обучения разработчиков и повышения уровня их мастерства. Кроме того, это означает, что данные программы будут храниться вне HTML-кода, используемого на вашем Web-сайте.

## Резюме

За последние несколько лет Интернет стал фактическим стандартом глобальных коммуникаций. Интернет — это глобальная сеть компьютеров, на которой выполняется набор протоколов TCP/IP. World Wide Web (WWW, Web) это одна из многих служб, работающих в Интернете; она обеспечивает графический интерфейс (GUI) для доступа к документам, форматированным в HTML. В настоящее время в Интернете размещены Web-сайты миллионов компаний и организаций, которые предоставляют различные типы сервисов. Отделы информационных систем рассматривают Интернет как новый подход к разработке систем, поскольку он предоставляет дополнительные преимущества, такие как универсальный доступ к данным, обобщенный графический интерфейс (GUI), низкие затраты на обучение пользователей и т. д.

Web-сервер предоставляет стандартные интерфейсы, такие как Common Gateway Interface (CGI) и программный интерфейс приложений (API). Все Web-серверы поддерживают CGI, но CGI увеличивает накладные расходы по обработке данных, поскольку использует внешние программы, которые загружаются многократно. Web API тесно интегрированы с программой Web-сервера, поэтому обеспечивают лучшую производительность. Однако Web API могут стать причиной выхода из строя Web-сервера, если они реализованы не совсем корректно. Другая проблема Web API в том, что они сильно зависят от производителя и операционной системы. Поэтому

существует множество Web API, например, ISAPI для Web-серверов Microsoft, NSAPI для Web-серверов Netscape и WSAPI для Web-сайтов O'Reilly.

Доступ к базам данных через Web достигается с помощью специального промежуточного программного обеспечения (ППО). Это ППО размещается между Web-сервером и ресурсом, доступ к которому необходимо получить. Его основная функция — обеспечить Web-доступ к другим внешним источникам, например, базам данных, системам электронной почты (e-mail), службам каталогов и FTP-сервисам.

ColdFusion представляет собой сервер Web-приложений, обеспечивающий доступ к базе данных (среди прочих других сервисов) через Web. ColdFusion может обеспечить доступ к базе данных с помощью ODBC или стандартных драйверов БД. ODBC предоставляет общий уровень абстракции для многих источников данных. В ColdFusion используются файлы сценариев, в которых содержатся выражения языка CFML (ColdFusion Markup Language), обеспечивающие доступ к базе данных. К самым распространенным тегам для работы с базами данных языка CFML относятся: CFQUERY, CFOUTPUT, CFTABLE, CFINSERT и CFUPDATE. ColdFusion также предоставляет поддержку обработки условий с помощью тега <CFIF>.

Разработка приложений для Интернет/интранет требует преодоления ограничений Web-интерфейса. Администраторы баз данных в современной интернет-среде должны понимать значимость Web-разработок для расширенной поддержки типов данных, безопасности баз данных, управления транзакциями БД и проектирования БД.

Оценивая вклад Интернета в инфраструктуру баз данных, необходимо обратить внимание, что он вызвал полный цикл обновления. То есть старое "большое железо" (большие машины, занимающие целые комнаты) заменили компактные недорогие Web-серверы, в сотни раз более производительные, чем устаревшие "большие" машины. На стороне клиента ужасные монохромные текстовые терминалы сменились недорогими компьютерами с графическим интерфейсом, производительность которых тоже в сотни раз выше, чем у старых машин. Все же, несмотря на огромные успехи микрокомпьютерной сферы, в некотором смысле произошел возврат к старой модели с мэйнфреймом, где обработка данных выполняется на стороне сервера.

## Основные термины

ActiveX

Java

JavaScript

VBScript

Web-обозреватель, Web-браузер — Web browser

Динамически подключаемая библиотека — dynamic-link library (DLL)

Интерфейс прикладного программирования — application programming interface (API)

Интерфейс прикладного программирования уровня вызовов — Call Level Interface (CLI)

Общий шлюзовой интерфейс — common gateway interface (CGI)

Открытый интерфейс доступа к базам данных — Open database Connectivity (ODBC)

Подключаемый модуль — plug-in

Промежуточное программное обеспечение Web для доступа к БД, ППО Web-БД — Web-to-database middleware

Сервер Web-приложений — Web application server

Серверное расширение — server-side extension

Система без сохранения информации о состоянии — stateless system

Сценарий — script

Тег системы ColdFusion или HTML — tag

Язык разметки гипертекста ColdFusion — ColdFusion Markup Language (CFML)

## Вопросы

1. Что означает выражение "Web является системой, не сохраняющей информацию о своем состоянии"? Какое влияние оказывает такая система на разработку приложений баз данных?
2. Для чего нужны интерфейсы Web-сервера? Приведите примеры.
3. Что такое служба ODBC и каковы ее функции?
4. Что такое сервер Web-приложений и каковы его функции с точки зрения БД?
5. Что такое сценарии и каковы их функции (дайте пояснение в терминах разработки приложений баз данных)?
6. Опишите основные службы (сервисы), предоставляемые сервером Web-приложений ColdFusion.
7. Найдите в Интернете серверы Web-приложений, отличные от ColdFusion, и подготовьте о них краткое сообщение в вашей группе.
8. Кратко опишите назначение следующих тегов ColdFusion: <CFQUERY>, <CFOUTOUT>, <CFTABLE>, <CFIF>, <CFINSERT> и <CFUPDATE>.
9. Расскажите, какие проблемы разработки Web-интерфейса связаны с типами данных, безопасностью, управлением транзакциями БД и денормализацией таблиц.
10. Опишите проблемы разработки Web-страниц, относящиеся к связям предок/родитель в БД.

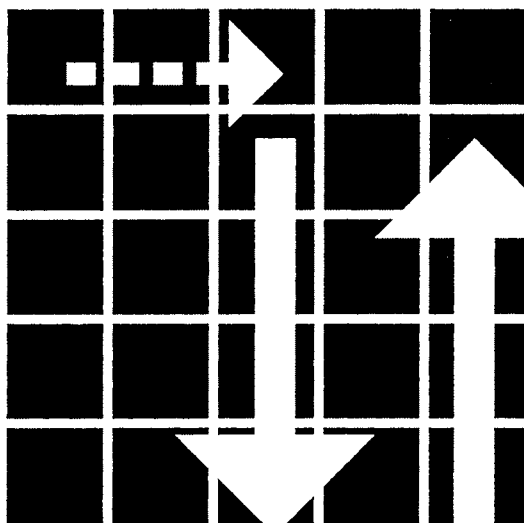
## Задачи

В следующих упражнениях студенты будут создавать сценарии ColdFusion. Преподаватель может назначить разработку сценария индивидуально каждому студенту или один сценарий нескольким студентам. После создания этих сценариев, разработайте основной, предназначенный для отображения записей и главных опций (для всех пяти сценариев по каждой таблице). Обсудите и опишите внешний ключ и бизнес-правила при создании сценариев.

1. Создайте сценарий ColdFusion для поиска, добавления, редактирования и удаления записей в таблице USER базы данных RobCor.

2. Создайте сценарий ColdFusion для поиска, добавления, редактирования и удаления записей в таблице INVTYPE базы данных RobCor.
3. Создайте сценарий ColdFusion для поиска, добавления, редактирования и удаления записей в таблице VENDOR базы данных RobCor.
4. Модифицируйте сценарии ввода данных (ch15-5a.cfm и ch15-5b.cfm) для таблицы DEPARTMENT таким образом, чтобы отделом могли управлять пользователи, работающие в этом отделе.
5. Создайте экран ввода данных заказа с помощью таблиц ORDER и ORDER\_LINE базы данных RobCor. Для этого необходимо использовать фреймы и другие дополнительные теги ColdFusion. Обратитесь к онлайн-руководству и просмотрите образцы приложений.





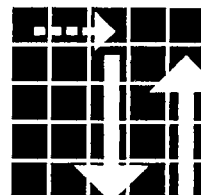
## ЧАСТЬ VI

# АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ





## Глава 16



# Администрирование баз данных

В этой главе мы обсудим:

- ☐ что информация представляет собой важную составляющую бизнеса и требует тщательного управления;
- ☐ почему база данных играет столь важную роль на предприятии;
- ☐ основные технологические и организационные последствия, которые влечет за собой внедрение СУБД на предприятии;
- ☐ организационную и техническую роль администратора базы данных;
- ☐ некоторые инструментальные средства администрирования баз данных;
- ☐ различные стратегии администрирования баз данных;
- ☐ как различные административно-технические задачи выполняются в среде Oracle.

## Обзор

В этой главе представлены основы успешной стратегии администрирования баз данных. Такая стратегия предполагает, что информация рассматривается как важнейший корпоративный ресурс, хранится и управляется как ценное корпоративное достояние.

Мы рассмотрим, каким образом база данных вписывается в схему предприятия, какие представления данных и требования к ней существуют на разных организационных уровнях и как СУБД обеспечивает эти представления и требования. Прежде чем реализовать стратегию администрирования на практике, необходимо понять и принять стратегию администрирования баз данных. Мы исследуем важнейшие проблемы управления данными и покажем организационную и техническую роль администратора базы данных (DBA, database administrator).

Технические аспекты администрирования баз данных будут дополнены обсуждением инструментальных средств администрирования, а также архитектуры данных в масштабах предприятия. Организационные аспекты администрирования баз данных будут проиллюстрированы на примере того, как функции администрирования БД вписываются в классические организационные схемы. Поскольку программное обеспечение Oracle в настоящее время лидирует на среднем и верхнем уровнях рынка корпоративных баз данных, для иллюстрации некоторых технических аспектов роли DBA в управлении средой БД мы будем использовать СУБД Oracle.

## 16.1. Информация как достояние корпорации

В гл. I было показано, что структурирование информации имеет очень большое значение для предприятия. Поэтому неудивительно, что информация в современной корпоративной среде рассматривается как весьма ценный ресурс, которым необходимо должным образом управлять.

Чтобы оценить значимость информации, только взгляните на то, что хранится в базе данных компании: данные о клиентах, поставщиках, инвентаре, операциях и т. д. Сколько возможностей будет упущено при утере этих данных? Как можно реально оценить убытки от утери данных? Например, бухгалтерская фирма, утратившая всю свою базу данных по причине выхода из строя жесткого диска или из-за заражения компьютерным вирусом, терпит огромные прямые и косвенные убытки. Проблемы фирмы будут еще большими, если данные утрачены именно в период расчета налогообложения! Утрата данных поставит компанию в затруднительное положение. Предприятие не сможет эффективно выполнять каждодневные операции и, возможно, от него уйдут клиенты, привыкшие к быстрому и эффективному обслуживанию, а новых клиентов такое положение дел вряд ли привлечет.

Если структурированная информация точна и своевременна, то она, скорее всего, приведет к повышению конкурентоспособности компании и улучшению ее благосостояния. На самом деле в любой организации постоянно действует цикл "данные — структурированные данные — принятие решений". *Пользователь* данных, используя свой интеллект, структурирует (упорядочивает) информацию должным образом, так что она становится основой знаний, используемых для принятия решений пользователем. Этот цикл представлен на рис. 16.1.

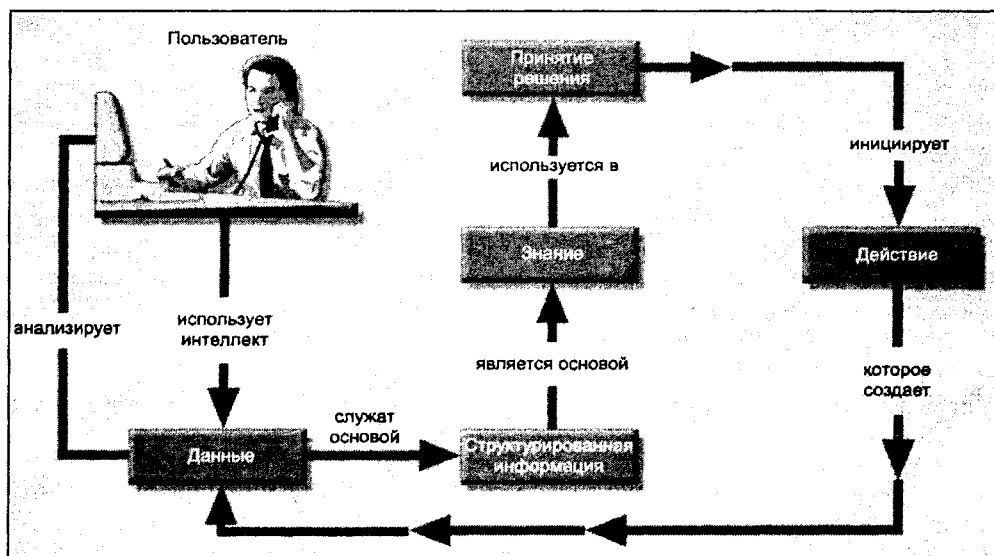


Рис. 16.1. Цикл преобразования информации

Если посмотреть на информационные потоки, представленные на рис. 16.1, то можно заметить, что решения, принимаемые руководством верхнего уровня, инициируют определенные действия на нижних уровнях предприятия. Эти действия создают дополнительную информацию, используемую для наблюдения за эффективностью работы компании. В свою очередь, в цикле "данные — структурированные данные — принятие решений" должна появляться дополнительная информация. Таким образом, данные составляют основу принятия решений, стратегического планирования и контролирования операций.

Важнейший фактор успеха организации — эффективное управление активами. Чтобы управлять информацией как корпоративным активом, руководители должны понимать значение упорядоченной, структурированной информации. На самом деле существуют компании (например, составляющие отчеты по кредитам), занимающиеся исключительно структурированием данных, и их успех целиком зависит от эффективности управления информацией.

## 16.2. Роль базы данных на предприятии

Данные используются различными людьми в различных подразделениях и в различных целях. Поэтому управление данными связано с концепцией разделения информации. В *гл. 1* было показано, что необходимость разделения данных практически неизбежно приводит к появлению СУБД. Если СУБД используется должным образом, то она обеспечит следующие преимущества:

- ☐ *интерпретация и представление* данных в удобном формате путем структурирования и упорядочивания необработанной ("сырой") информации;
- ☐ *распространение* информации среди нужных людей в нужное время;
- ☐ *защита* данных и *контроль* доступа к данным в соответствующие периоды времени;
- ☐ *контроль* дублирования данных и использование их как на внешнем, так и на внутреннем уровне.

Вне зависимости от типа организации основная роль базы данных состоит в *обеспечении поддержки принятия решений на всех уровнях предприятия*.

В организационной структуре предприятия можно выделить три уровня: высший, средний и оперативный (рабочий). Руководство высшего уровня принимает стратегические решения, руководство среднего уровня принимает тактические решения, а руководство рабочего уровня принимает каждодневные, оперативные решения.

СУБД должна предоставлять средства, обеспечивающие различное представление данных на разных уровнях и поддержку принятия решения на этих уровнях. Роль базы данных в обеспечении поддержки принятия решений будет более понятной, если мы подытожим действия, типичные для каждого организационного уровня.

- ☐ *Высшее исполнительное руководство*. На этом уровне база данных должна:
  - предоставлять информацию, необходимую для принятия стратегических решений, стратегического планирования, формирования политики и определения целей;

- обеспечивать доступ к внешним и внутренним данным для выявления возможности роста и направления этого роста (направление зависит от рода деятельности: является ли предприятие сервисным, промышленным или же комбинацией таких форм);
- предоставлять структуру для определения и проведения в жизнь политики предприятия (вспомните, что такая политика преобразуется в бизнес-правила на нижнем уровне предприятия);
- повышать вероятность позитивных изменений в инвестиционной деятельности компании за счет поиска новых способов снижения затрат и/или за счет повышения производительности;
- обеспечивать обратную связь, чтобы увидеть, достигает ли компания поставленных целей.

□ *Руководство среднего звена.* На этом уровне база данных должна обеспечить:

- предоставление необходимых сведений для принятия тактических решений и планирования;
- наблюдение и контроль размещения и использования ресурсов предприятия:
  - ◇ в частности, насколько эффективно размещаются и используются ресурсы? вспомните, что данные тоже являются ресурсом предприятия, поэтому база данных должна предоставлять необходимые сведения по наблюдению и контролю использования информации. Например, отдел информационных систем должен иметь возможность правильно интегрировать все компоненты базы данных с помощью БД;
  - ◇ какие проблемы возникают при обработке данных? насколько хорошо среда БД позволяет определить и изучить альтернативные решения? такие решения часто требуют проработки сценария и детальной прогностической информации;
  - ◇ оценка производительности различных подразделений;
- предоставление структуры для реализации поддержки защиты и секретности данных в БД. *Безопасность* (security) означает защиту данных от их случайного или преднамеренного использования неавторизованными пользователями. *Секретность*, или *конфиденциальность* (privacy) связана с правами отдельных пользователей и организации в целом для определения того, "кто, что, когда, где и как" может использовать данные.

□ *Оперативное руководство.* На этом уровне база данных должна обеспечивать:

- представление и поддержку операций компании, насколько это возможно. Модель данных должна быть достаточно гибкой, чтобы в нее можно было ввести все необходимые представления и ожидаемые данные;
- получать результаты запросов на определенном уровне исполнения. Помните, что требования производительности ужесточаются для более низких уровней управления. Поэтому БД должна обеспечивать быструю обработку большого числа транзакций на оперативном уровне;

- расширять возможности краткосрочных операций предприятия, предоставляя:
  - ◊ периодическое информирование клиентов;
  - ◊ поддержку разработки приложений и вычислительных операций.

Основная цель базы данных состоит в обеспечении бесперебойного управления потоками информации в рамках предприятия.

Базу данных компании также называют *корпоративной базой данных (corporate database)*, или *базой данных предприятия (enterprise database)*. Базу данных предприятия можно определить как представление данных предприятия, обеспечивающее поддержку деятельности предприятия в настоящем и будущем. Большинство современных успешных компаний зависят от возможностей базы данных предприятия по обеспечению поддержки всей их деятельности — от проектирования до реализации, продажи и обслуживания, а также от стратегического планирования до принятия каждодневных решений.

### 16.3. Внедрение базы данных: анализ проблем

Наличие компьютеризованной системы управления базой данных еще не гарантирует того, что данные будут должным образом использоваться для принятия руководителями верных решений. СУБД представляет собой лишь средство управления данными и для получения желаемого результата должно эффективно использоваться. Точно так же молоток в руках столяра помогает создавать мебель, а в руках ребенка может стать средством разрушения. Решение проблем компании заключается не в установке компьютерной системы и базы данных, а в эффективном использовании этих средств.

Внедрение СУБД представляет собой трудную задачу и оказывает на предприятие в целом сильное влияние, которое может быть как позитивным, так и негативным, в зависимости от того, как им распорядиться. Например, одно из важных соображений — необходимость адаптации СУБД к условиям предприятия, а не адаптация предприятия к требованиям СУБД. Во главу угла следует ставить потребности предприятия, а не технические возможности СУБД. Тем не менее, внедрение СУБД оказывает воздействие на предприятие в целом. Только лишь обилие новой информации, которую помогает создать СУБД, оказывает огромное влияние на методы работы предприятия и культуру его производства.

Внедрение СУБД на предприятии представляет собой процесс, в рамках которого решаются три основных блока проблем<sup>1</sup>:

- *технологические*: программное и аппаратное обеспечение СУБД;
- *организационные*: административные действия;
- *интеллектуальные*: внутреннее противодействие сотрудников корпорации любым нововведениям.

---

<sup>1</sup> John P. Murray, "The Managerial and Cultural Issues of a DBMS", 370/390 Database Management I(8), сентябрь 1991, стр. 32—33.

Технологические аспекты включают в себя выбор, установку и наблюдение за СУБД для обеспечения условий хранения данных, доступа к ним и эффективной безопасности. Лица, ответственные за указанные технологические аспекты установки СУБД, должны иметь определенную специализацию, необходимую для обслуживания всех пользователей СУБД: программистов, управленцев и конечных пользователей. Поэтому укомплектование штата администрации БД — ключевая организационно-техническая задача на этапе внедрения СУБД. Отобранный персонал должен обладать и техническими, и руководящими навыками для обеспечения работы новой структуры совместного использования данных.

Нельзя сбрасывать со счетов организационный аспект внедрения СУБД. Самая хорошая СУБД не гарантирует получения хорошей информационной системы, как наличие великолепной гоночной машины не гарантирует выигрыша в гонке при отсутствии хороших механиков и водителей.

Внедрение СУБД на предприятии требует тщательного планирования по созданию надлежащей организационной структуры, включающей в себя всех лиц, ответственных за администрирование СУБД. Эта организационная структура также должна хорошо управляться и контролироваться. Администраторы должны обладать отличными навыками работы с людьми и коммуникабельностью в совокупности с хорошим пониманием специфики предприятия и сферы его деятельности. Высшее руководство должно активно участвовать в разработке новой системы администрирования данных и должно определять и поддерживать ее административные функции, цели и роль в рамках всего предприятия.

Воздействию, которое оказывает внедрение СУБД на умы и настроения сотрудников, а также на культуру производства, необходимо уделить самое пристальное внимание. Внедрение СУБД, скорее всего, окажет влияние на людей, их функции и взаимоотношения. Например, возможно появление новых сотрудников, имеющиеся сотрудники, возможно, станут выполнять иные функции, а производительность сотрудников может оцениваться по новым стандартам.

Интеллектуальный шок тоже вполне вероятен, поскольку в системах баз данных информационные потоки контролируются и структурируются гораздо более строго. Руководители подразделения, привыкшие работать с собственной информацией, должны смириться с тем, что перестанут быть исключительными собственниками информации, которая станет доступной и другим сотрудникам. Прикладные программисты должны изучить новые стандарты проектирования и разработки и следовать их требованиям. Руководители могут столкнуться с достаточно большой информационной перегрузкой, и им может потребоваться некоторое время для адаптации к новым условиям.

Когда новая база данных войдет в строй, сотрудники могут с большой неохотой использовать информацию, предоставляемую системой, и подвергать сомнению ее ценность и точность (многие из них удивятся и, возможно, огорчатся, обнаружив, что полученная информация не соответствует их представлениям и понятиям). Отдел администрирования БД должен быть готов ответить на вопросы пользователей, выслушать их размышления, по возможности ответить на них и обучить пользователей навыкам работы с системой и использования всех ее преимуществ.

## 16.4. Эволюция функций администратора базы данных

Администрирование данных берет свое начало в старом децентрализованном мире систем файлов. Затраты на информацию и дублирование управленческих функций в таких системах файлов привели к возникновению централизованного аппарата, который именовался *отделом электронной обработки данных (EDP, electronic data processing)* или *отделом обработки данных (DP, data processing)*. В задачу отдела DP входило объединение всех вычислительных ресурсов для поддержки всех отделов на оперативном уровне с минимальным дублированием информации. В компетенцию DP-администрации входило управление существующими системами файлов компании, а также разрешение информационных и организационных конфликтов, возникающих при дублировании или неправильном использовании информации.

Появление СУБД с ее идеологией совместного использования данных — это новый уровень управления информацией, который привел к преобразованию отдела DP в *отдел информационных систем*, или *отдел ИС (information systems department, IS)*. В сферу ответственности этого отдела входят:

- ❑ *сервисные* функции по обеспечению конечных пользователей активной поддержкой в области управления данными;
- ❑ *производственные* функции по обеспечению пользователей специфическими решениями в соответствии с их информационными потребностями с помощью интеграции приложений или информационных систем управления.

Функциональная ориентация отдела ИС нашла отражение и во внутренней организационной структуре. Структура отдела ИС обычно выглядела так, как это представлено на рис. 16.2. По мере роста требований к разработке приложений сегмент разработок отдела ИС подразделялся по типам поддерживаемых систем: бухгалтерские, складской учет, сбыт и т. д. Однако это означало, что функции администраторов баз данных также разделялись. Сегмент разработки приложений отвечал за сбор требований к БД и логическое проектирование, в то время как сегмент операций с БД отвечал за реализацию, наблюдение и управление операциями СУБД.

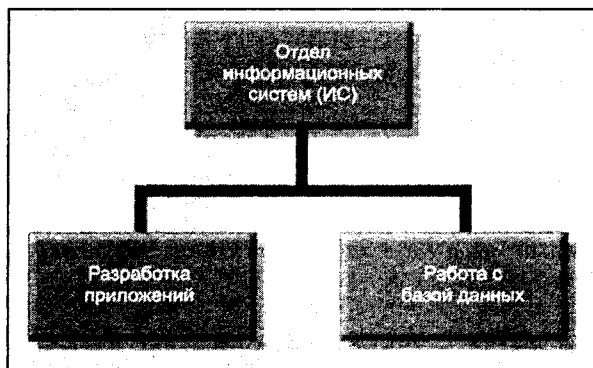


Рис. 16.2. Внутренняя структура отдела информационных систем

По мере роста приложений баз данных управление данными становилось все более сложной задачей, что привело к разработке *функций администрирования баз данных (database administration function)*. Лицо, ответственное за управление централизованной и распределенной базой данных, называется *администратором баз данных (DBA, database administrator)*.

Масштаб и роль DBA при введении его в организационную схему варьируется от компании к компании. В схеме организации должность DBA может быть либо штатной, либо консультирующей. Консультирующая должность DBA обычно предполагает создание консультационной инфраструктуры, и в этом случае DBA могут продумывать стратегию администрирования данных, но не имеют права проводить ее в жизнь или разрешать возможные конфликты<sup>2</sup>.

Штатный администратор DBA отвечает за планирование, определяет, реализует и проводит в жизнь политику, стандарты и процедуры, необходимые администратору. Два варианта функционирования DBA представлены на рис. 16.3.

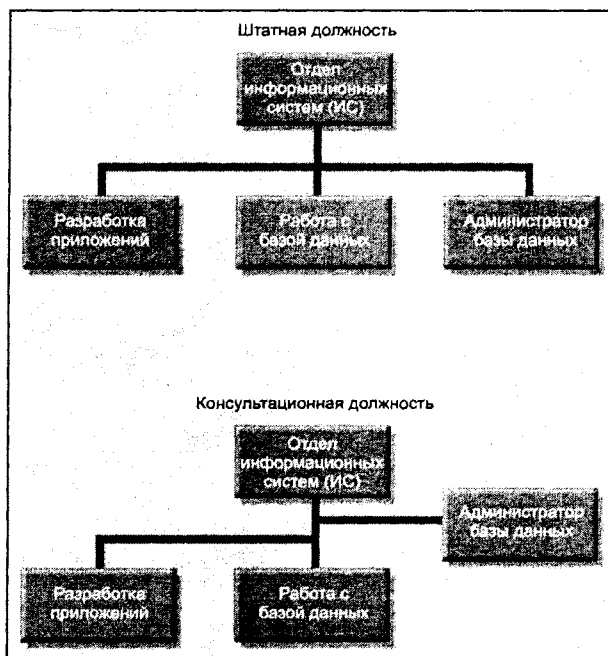


Рис. 16.3. Место DBA в структуре предприятия

<sup>2</sup> Если вы хотите изучить историю развития функций DBA и организацию деятельности DBA на предприятии, авторы рекомендуют работу "Jay-Louise Weldon's classic Data Base Administration" (New York: Plenum Press, 1981). Хотя, судя по дате публикации, она может показаться вам устаревшей, очень многие темы этой работы являются весьма актуальными для современного состояния рабочих баз данных. (Вспомните, например, что "тонкий" клиент в клиент/серверной архитектуре отражает многие концепции "устаревшей" схемы с мэйнфреймом, а концептуальная структура объектной модели во многом напоминает идеи, существовавшие в концептуальной структуре иерархической модели).



Решение о роли DBA на предприятии зависит от стиля руководства компании и от таких факторов, как сложность и масштаб деятельности предприятия и географическое распределение служб компании. Эти факторы помогают определить внутреннюю структуру функций DBA.

По всей видимости, стандартов на внутреннюю организацию DBA не существует. Недостаток стандартизации отчасти объясняется тем, что функции DBA сами по себе являются очень динамичными по сравнению с другими организационными функциями. На самом деле, быстрые изменения в технологии СУБД определяют изменение организационного стиля. Например:

- ☐ развитие распределенных баз данных может способствовать децентрализации функций администрации данных на предприятии. Распределенные БД требуют, чтобы системные DBA несли ответственность за действия каждого локального DBA, что требует от системных DBA выполнения достаточно сложных координирующих функций;
- ☐ возрастающее использование баз данных, ориентированных на Интернет, и объектно-ориентированных БД и растущее число приложений хранилищ данных, вероятно, потребуют от DBA новых действий в плане моделирования и проектирования, что также расширяет сферу деятельности администраторов;
- ☐ возрастающая сложность и мощь программного обеспечения СУБД, предназначенного для микрокомпьютерных систем, предоставляет удобную платформу для разработки дружественных пользователю, экономичных и эффективных решений для специфических потребностей подразделений. Но такая структура тоже ведет к дублированию информации, не говоря уже о проблемах, связанных с человеческим фактором, например, о недостаточной квалификации для разработки высококачественных проектов баз данных. Иначе говоря, новая микрокомпьютерная среда требует от DBA новых технических и организационных навыков.

Хотя в настоящее время на этот счет нет стандартов, стало общепринятой практикой определять функции DBA, разделяя деятельность DBA в соответствии с фазами *жизненного цикла БД (database life cycle, DBLC)*. При таком подходе необходимо, чтобы деятельность DBA охватывала следующие направления:

- ☐ планирование базы данных, включая определение стандартов и процедур, а также проведение их в жизнь;
- ☐ сбор требований к БД и концептуальное проектирование;
- ☐ логическое проектирование БД и разработка транзакций;
- ☐ физическое проектирование БД и ее реализация;
- ☐ тестирование и отладка БД;
- ☐ операции с БД и ее обслуживание, включая установку, конвертирование и миграцию;
- ☐ обучение работе с БД и поддержка пользователей.

Если в качестве модели административной деятельности DBA используется именно такая схема, то в этом случае примерная функциональная организация DBA выглядит так, как это представлено на рис. 16.4.

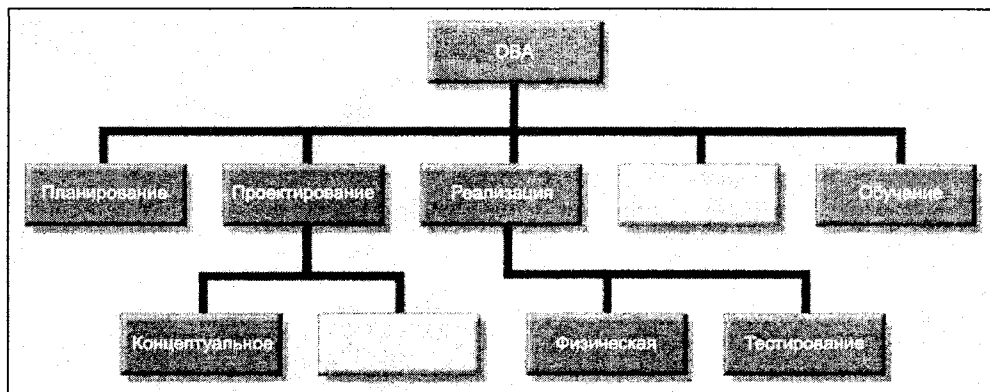


Рис. 16.4. Функциональная организация DBA

Следует иметь в виду, что у компании может быть несколько различных и несовместимых СУБД, установленных для обеспечения различных операций. Например, может случиться так, что для ежедневных операций используется иерархическая СУБД, а для обслуживания нерегламентированных запросов на уровне руководства высшего и среднего звена используется реляционная база данных. Предприятие может иметь несколько микрокомпьютерных СУБД, установленных в различных подразделениях. При такой структуре компания может назначать DBA на каждую отдельную СУБД. Генеральным координатором деятельности всех DBA, как правило, является администратор, обычно называемый *системным администратором (SYSADM)*, место которого в общей структуре продемонстрировано на рис. 16.5.

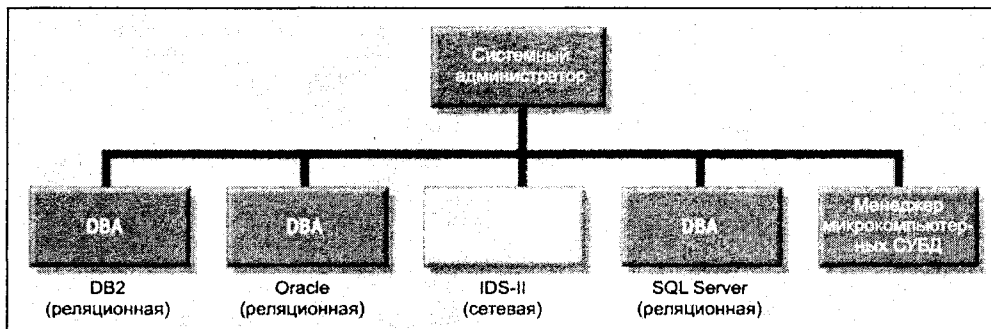


Рис. 16.5. Несколько администраторов баз данных (DBA) в структуре предприятия

В настоящее время намечается тенденция к специализации функций управления данными. Например, в схеме предприятия, которая часто используется в больших корпорациях, разделяются DBA и *администратор данных (DA, data administrator)*. DA, которого еще называют *менеджером информационных ресурсов (information resource manager, IRM)*, обычно напрямую подотчетен высшему руководству и имеет более

высокие полномочия и степень ответственности, чем DBA, хотя их роли частично перекрываются.

DA отвечает за управление всеми ресурсами данных предприятия, как компьютерными, так и рукописными. Поэтому поле деятельности DA шире, чем у DBA, поскольку DA отвечает за контроль не только компьютерных ресурсов, но и информации, находящейся вне поля зрения СУБД. Место DBA внутри расширенной структуры предприятия может варьироваться от компании к компании. В зависимости от структуры, DBA могут быть подотчетны DA, руководству отдела информационных систем или напрямую президенту компании.

## 16.5. Человеческий фактор в среде базы данных

Основная часть материала этой книги посвящена проектированию и реализации реляционных баз данных. Здесь также исследованы свойства и возможности СУБД. Говоря кратко, все внимание было сосредоточено на самых важных технических аспектах баз данных. Однако есть еще одна немаловажная составляющая среды баз данных, которая еще ждет своего исследования: даже самые тщательно проработанные системы баз данных не могут работать без человека. Поэтому необходимо наконец узнать, каким образом люди выполняют работу по администрированию данных, делая, в конечном счете, хорошо спроектированную БД действительно полезной.

Эффективное администрирование данных требует как технических, так и организационных навыков. В предыдущем разделе этой главы было сказано, что в том случае, если в структуре предприятия имеются как DBA, так и DA, их функции зачастую перекрываются. Тем не менее, можно установить некоторые общие правила. Например, работа DA обычно имеет организационную направленность в рамках всего предприятия. В отличие от этого работа DBA больше связана с техническими проблемами и ограничена рамками СУБД. Однако DBA также должен хранить сведения о специализации сотрудников, поскольку и DA, и DBA выполняют функции "работы с людьми", общие для всех отделов предприятия. Например, и DA, и DBA управляют подбором кадров и их обучением внутри соответствующих подразделений. Естественно, мы сейчас говорим в основном об их роли в управлении данными.

В табл. 16.1 сравниваются основные свойства этих должностей и представлены направления их деятельности. Все функциональные обязанности, вытекающие из свойств, представленных в табл. 16.1, будут выполняться только DBA, если в структуре организации должности DA и DBA не разделяются.

**Таблица 16.1. Сопоставление деятельности DBA и DA и их характеристики**

| Администратор данных (DA)      | Администратор базы данных (DBA)                |
|--------------------------------|------------------------------------------------|
| Стратегическое планирование    | Управление и контроль                          |
| Определение долгосрочных целей | Исполнение планов для достижения целей         |
| Выработка политики стандартов  | Проведение в жизнь политик и процедур          |
|                                | Проведение в жизнь стандартов программирования |

Таблица 16.1 (окончание)

| Администратор данных (DA)      | Администратор базы данных (DBA)                                    |
|--------------------------------|--------------------------------------------------------------------|
| Широкие масштабы деятельности  | Узкие рамки деятельности                                           |
| Долгосрочные перспективы       | Краткосрочные перспективы (основное внимание ежедневным операциям) |
| Организационная направленность | Техническая направленность                                         |
| Независимость от СУБД          | Привязка к СУБД                                                    |

В табл. 16.1 обратите внимание на то, что DA отвечает за разработку глобальной и четкой административной стратегии, касающейся всех данных предприятия. Иначе говоря, планы администрирования данных DA должны всегда простирались на весь спектр данных. Поэтому DA отвечает за консолидацию и непротиворечивость как рукописной, так и компьютеризованной информации.

DA также должен определить цели администрирования данных, которое должно решать следующие задачи:

- ☐ "разделяемость" (возможность совместного использования) данных и готовность их к использованию;
- ☐ непротиворечивость и целостность данных;
- ☐ безопасность и конфиденциальность данных;
- ☐ рамки и характер использования информации.

Естественно, список этих задач можно расширить в соответствии с потребностями предприятия. Независимо от способа управления данными (и независимо от того, какие полномочия предоставлены DA или DBA по определению способа управления данными), DA и DBA не являются собственниками информации. Наоборот, функции DA и DBA определяются так, чтобы подчеркнуть, что информация является общедоступным достоянием компании.

Все предшествующие рассуждения не должны привести вас к убеждению, что для DA и DBA есть общепринятые административные стандарты. На самом деле, стиль, ответственность, место в структуре предприятия и внутренняя структура этих должностей варьируются от компании к компании. Например, многие компании распределяют обязанности DA между DBA и руководителем отдела информационных систем. Для упрощения и чтобы избежать путаницы, мы будем использовать обозначение DBA как общее, куда включены все необходимые административные функции. Тогда мы можем проиллюстрировать роль DBA как посредника между пользователями и информацией.

Посредничество во взаимодействии между двумя наиболее важными ценностями предприятия (люди и информация) определяет место DBA в динамической среде, представленной на рис. 16.6.

На рис. 16.6 обратите внимание, что DBA находится в центре взаимодействия пользователей и информации. DBA определяет и реализует процедуры и стандарты, используемые программистами и конечными пользователями в работе с СУБД. DBA

также следит за тем, чтобы доступ программистов и конечных пользователей к данным соответствовал требованиям стандартов безопасности.

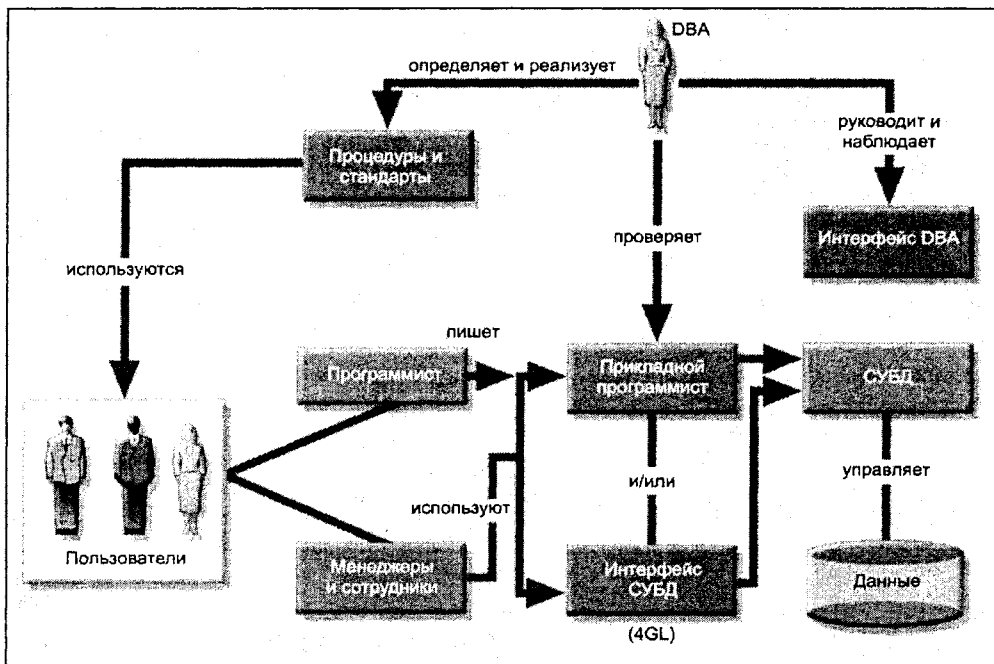


Рис. 16.6. Общее направление деятельности DBA

Пользователей базы данных можно классифицировать:

- ☐ по уровню поддержки принятия решений (пользователи оперативного уровня, тактического или стратегического);
- ☐ по уровню знаний в области информатики (новичок, опытный пользователь или профессионал);
- ☐ по частоте доступа (редкий доступ, периодический или частый).

Типы этой классификации не являются единственно возможными и зачастую перекрывают друг друга. Например, оперативный пользователь может быть профессионалом с редким доступом к базе данных. Типичный руководитель верхнего звена является стратегическим пользователем и новичком в компьютерной сфере с периодическим доступом к БД. Прикладного программиста баз данных можно рассматривать как пользователя с оперативным уровнем принятия решений и как профессионала-компьютерщика с частым доступом к базе данных. Поэтому каждое предприятие принимает на работу людей, уровни специализации которых в области БД охватывают весь спектр этой классификации. DBA должен уметь взаимодействовать со всеми этими людьми, понимать их различные потребности, отвечать на вопросы любого уровня сложности и эффективно сотрудничать с ними.

Деятельность DBA, представленная на рис. 16.6, предполагает наличие очень широкого диапазона навыков. В больших компаниях эти обязанности обычно распределяются между несколькими сотрудниками, выполняющими роль DBA. В небольших компаниях эти обязанности могут выполняться и одним сотрудником. Диапазон профессиональных навыков DBA широк, разнообразен и тесно связан с характером его работы. Можно классифицировать эти профессиональные навыки по двум разным категориям: организационные и технические. Подобная классификация представлена в табл. 16.2.

**Таблица 16.2. Профессиональные навыки DBA**

| Организационные                                                     | Технические                                                                                                                                                                           |
|---------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Широкие познания в области бизнеса                                  | Широкие познания в области обработки данных                                                                                                                                           |
| Умение координировать деятельность                                  | Знания в области жизненного цикла разработки систем                                                                                                                                   |
| Аналитические навыки                                                | Знание структурированных методик: <ul style="list-style-type: none"> <li>• Схемы информационных потоков</li> <li>• Структурные диаграммы</li> <li>• Языки программирования</li> </ul> |
| Умение разрешать конфликты                                          | Знания в области жизненного цикла баз данных                                                                                                                                          |
| Умение общаться (устно и письменно)                                 | Навыки моделирования и проектирования баз данных: <ul style="list-style-type: none"> <li>• Концептуального</li> <li>• Логического</li> <li>• Физического</li> </ul>                   |
| Умение вести переговоры                                             | Оперативные навыки: реализация баз данных, управление словарем базы данных, безопасность и т. д.                                                                                      |
| Опыт работы: 2–5 лет по обработке информации на большом предприятии |                                                                                                                                                                                       |

Изучая табл. 16.2, необходимо помнить, что DBA исполняет две различные роли. Организационная роль DBA сосредоточена на управлении персоналом и взаимоотношениях с конечными пользователями. Техническая роль DBA охватывает деятельность, связанную с использованием СУБД (проектирование БД, разработка и реализация), а также разработку и использование прикладных программ. В следующих разделах мы подробно исследуем организационные и технические функции DBA.

### 16.5.1. Организационные функции DBA

Как руководитель, DBA должен сосредоточиться на аспектах управления и планирования при администрировании БД. Поэтому DBA отвечает за следующие направления:

- координация, наблюдение и распределение административных ресурсов БД: людей и информации;

- определение целей и формулирование стратегических планов деятельности по администрированию БД.

Конкретные направления, за которые несет ответственность DBA, приведены в табл. 16.3.

Таблица 16.3. Деятельность DBA

| Направление деятельности DBA |  | Сервис DBA                                            |
|------------------------------|--|-------------------------------------------------------|
| Планирование                 |  | Поддержка конечных пользователей                      |
| Организационные мероприятия  |  | Политики, процедуры и стандарты                       |
| Тестирование                 |  | Безопасность, конфиденциальность и целостность данных |
| Текущий контроль             |  | Резервное копирование и восстановление данных         |
| Доставка                     |  | Распределение и использование данных                  |

Табл. 16.3 показывает, что DBA в основном отвечает за планирование, организацию, тестирование, текущий контроль и обеспечение большого числа сервисов. Эти сервисы может выполнять DBA или, что более вероятно, сотрудники информационного отдела. Для упрощения, а также для того, чтобы наши рассуждения можно было применить в более широких масштабах, мы не делаем различия между сервисами DBA и сервисами, обеспечиваемыми персоналом информационного отдела под управлением DBA. Вместо этого мы обозначаем аббревиатурой DBA всех исполнителей сервисов в рамках функций администрирования БД. Именно с этой точки зрения мы и будем детально исследовать сервисы DBA.

## Поддержка конечных пользователей

DBA взаимодействует с конечными пользователями, предоставляя им сервисы информационной поддержки в подразделениях предприятия. Поскольку у конечных пользователей обычно совершенно разные уровни компьютерной грамотности, сервисы конечных пользователей обычно включают в себя следующие компоненты.

- *Сбор требований пользователей.* DBA должен работать совместно с пользователями для сбора информации, необходимой для выявления и описания проблем пользователей. Коммуникабельность DBA на этом этапе имеет очень большое значение, поскольку у тех, с кем работает DBA, очень разные уровни знаний и стили общения. При сборе требований от DBA требуется:
  - точно понимать потребности пользователей и их представления. Если DBA не составит точную картину проблемы, то СУБД не сможет обеспечить надлежащее и эффективное решение. Помните, что неправильно поставленная задача обычно приводит к плохому конечному результату;
  - выявить текущие и будущие информационные потребности. Конечный пользователь видит только свою часть общей информационной картины. На са-

мом деле, пользователь видит только верхушку айсберга. В отличие от него, DBA со своей позиции видит схему данных организации в целом.

Успех администрирования данных на предприятии определяется тем, насколько полно удовлетворены потребности пользователя. Если DBA заранее предвидит проблемы и предугадывает информационные потребности пользователя, то тем самым он создает хорошие предпосылки для повышения доверия пользователей к деятельности DBA.

- *Разрешение конфликтов и проблем.* Решение проблем конечных пользователей одного подразделения может инициировать конфликт с другими подразделениями. У пользователей обычно имеются свои специфические требования к информации, которые отличаются от требований других пользователей и, скорее всего, они не желают знать, как их информация влияет на другие подразделения организации. Когда возникает конфликт, DBA имеет право (и обязан) выяснить причины конфликта и разрешить его.
- *Поиск решений в области удовлетворения информационных потребностей.* Возможность и право DBA разрешать конфликты в сфере информации позволяют ему разрабатывать решения, которые хорошо вписываются в существующую структуру управления. Основная цель DBA состоит в удовлетворении информационных потребностей пользователя. Учитывая возрастающее влияние Интернета, решение этой задачи, скорее всего, потребует внедрения и поддержки Web-обозревателей для взаимодействия с базой данных. На самом деле, резкий рост е-коммерции требует использования динамических интерфейсов для обеспечения интерактивного заказа товаров и его продажи (см. гл. 14).
- *Обеспечение качества и целостности приложений и данных.* После того как решение найдено, его необходимо должным образом реализовать и использовать. Поэтому DBA работают совместно как с прикладными программистами, так и с конечными пользователями для обучения их стандартам и процедурам доступа к данным и манипулирования ими. DBA должен также гарантировать, что транзакции БД не оказывают неблагоприятного воздействия на качество информации в БД. Сертификация качества прикладных программ, получающих доступ к БД, является одной из важнейших функций DBA. Особое внимание должно уделяться интерфейсам СУБД с Интернетом, поскольку такие интерфейсы не предоставляют средств управления транзакциями, обычно имеющихся в среде баз данных, управляемых средствами СУБД. Например, если "внутренняя" транзакция БД, основанная на некотором приложении, требует инициализации триггера, то этот триггер должен также быть инициализирован, если транзакция осуществляется через интернет-интерфейс.
- *Создание доверительных отношений с конечными пользователями.* Поиск наилучшего решения проблем пользователей повышает их доверие к деятельности DBA.
- *Организация обучения и поддержка пользователей СУБД.* Пожалуй, больше всего времени отнимает у DBA обучение конечных пользователей надлежащей работе с БД. DBA должен обеспечить, чтобы все пользователи, получающие доступ к БД, имели основные понятия о функциях и правилах пользования программным обеспечением СУБД. DBA координирует и наблюдает за всей деятельностью, связанной с обучением пользователей.



## Политики, процедуры и стандарты

Основной компонент успешной стратегии администрирования данных — постоянное проведение в жизнь политик, процедур и стандартов по корректному созданию данных, их использованию, распространению и удалению в рамках БД. DBA должен определить, документировать и довести до сведения пользователей все политики, процедуры и стандарты до начала проведения их в жизнь.

- ☐ *Политики (policies)* — основные руководящие инструкции или действия, соответствующие целям DBA.
- ☐ *Стандарты* более детализированы и специфичны, чем политики, и описывают минимальные требования по данному направлению деятельности DBA. В действительности стандарты представляют собой правила, которые используются для оценки деятельности. Например, стандарты определяют принципы структурирования прикладных программ, соглашения об именовании, которые должны использовать программисты, и т. д.
- ☐ *Процедуры* — тексты инструкций, описывающие последовательность действий, которые необходимо выполнять в рамках какого-либо мероприятия. Процедуры должны разрабатываться в рамках существующих режимов эксплуатации и поддерживать и улучшать существующую среду БД.

Чтобы продемонстрировать различие между политиками, стандартами и процедурами, рассмотрим следующие примеры.

### Политики

- ☐ Все пользователи должны иметь пароли.
- ☐ Пароли должны меняться каждые шесть месяцев.

### Стандарты

- ☐ Пароль должен содержать не меньше пяти символов.
- ☐ Пароль должен содержать не больше двенадцати символов.
- ☐ Нельзя использовать в качестве пароля номер социального страхования и дату рождения.

### Процедуры

Для создания пароля выполняются следующие действия.

1. Конечный пользователь посылает DBA письменный запрос на создание учетной записи.
2. DBA утверждает запрос и пересылает его оператору.
3. Оператор создает учетную запись, присваивает временный пароль и отправляет информацию об учетной записи пользователю. Копия этой информации отправляется DBA.
4. Пользователь меняет временный пароль на постоянный.

Стандарты и процедуры, определенные DBA, используются всеми конечными пользователями, которые хотят извлечь максимальную пользу из БД. Стандарты и про-

цедуры должны дополнять друг друга и расширять политики администрирования данных. Процедуры должны облегчать работу пользователей и DBA. DBA определяющие, доводят до сведения и проводят в жизнь процедуры, которые затрагивают следующие сферы деятельности.

- ❑ *Сбор требований конечных пользователей к базе данных.* Какая документация необходима? Какие формы нужно использовать?
- ❑ *Проектирование и моделирование базы данных.* Какую методологию нужно использовать при проектировании БД (нормализация, объектно-ориентированные технологии и т. д.)? Какие инструментальные средства необходимо использовать (CASE-средства, словари данных, ER-диаграммы и т. д.)?
- ❑ *Соглашение о документировании и именовании.* Должно использоваться при определении всех элементов данных, наборов и программ, получающих доступ к базе данных.
- ❑ *Разработка, кодирование и тестирование прикладных программ БД.* DBA должен определить стандарты кодирования прикладных программ, документации и тестирования. Стандарты и процедуры, утвержденные DBA, передаются прикладным программистам, а DBA должен следить за использованием этих стандартов.
- ❑ *Выбор программного обеспечения БД.* Выбором пакетов программного обеспечения БД и любых других программных продуктов, имеющих отношение к БД, необходимо управлять. Например, DBA может потребовать, чтобы программное обеспечение должным образом взаимодействовало с существующим программным обеспечением, чтобы оно имело возможности, необходимые для предприятия, и чтобы оно могло быстро окупиться. В сегодняшней среде Интернета DBA должны взаимодействовать с Web-администраторами для обеспечения корректной связи глобальной Сети с базой данных.
- ❑ *Обеспечение безопасности и целостности данных.* DBA должен определить политики, управляющие безопасностью и целостностью. Безопасность БД имеет особое значение. Необходимо четко определить стандарты безопасности и строго их выполнять. Процедуры по обеспечению безопасности должны разрабатываться для обработки множества сценариев безопасности, обеспечивая максимальную возможную безопасность. Хотя не существует абсолютно безопасных схем, процедуры безопасности должны разрабатываться в соответствии с самыми строгими стандартами. Растущее использование интернет-интерфейсов к БД открывает двери для новых гораздо более серьезных и значительных угроз, чем те, которые встречаются в традиционных интерфейсах. Поэтому DBA должны работать в тесном сотрудничестве со специалистами по безопасному доступу в Интернет, чтобы обеспечить надежную защиту базы данных от внешних атак (как случайных, так и осознанных), совершаемых неавторизованными пользователями.
- ❑ *Резервное копирование и восстановление баз данных.* Процедуры резервного копирования и восстановления БД должны включать в себя информацию, необходимую для надлежащего управления резервным копированием.
- ❑ *Обслуживание базы данных.* Ежедневные операции СУБД должны быть надлежащим образом документированы. Операторы должны хранить регистрационные журналы, в которых записываются инструкции и примечания. Эти примечания очень полезны при определении точного местонахождения причин ошибок и

разрешении проблем. Операционные процедуры должны также включать в себя точную информацию, касающуюся процедур резервного копирования и восстановления.

- *Обучение пользователей.* На предприятии должна быть принята полноценная программа обучения, а процедуры управления обучением должны быть точно определены. Цель — четко определить, *кто* должен выполнять, *что* и как. Каждый пользователь должен ознакомиться с видом и формой доступного обучения.

Процедуры и стандарты должны ревизоваться, по крайней мере, ежегодно, чтобы соответствовать современным требованиям. Периодическая проверка стандартов и процедур гарантирует, что предприятие может быстро адаптироваться к изменениям рабочей среды, вызванным внутренними или внешними причинами. Разумеется, внедрение нового программного обеспечения СУБД, выявление нарушений безопасности и целостности информации, реорганизация компании и тому подобные изменения требуют пересмотра всех стандартов и процедур.

## **Безопасность, конфиденциальность и целостность данных**

Безопасность, конфиденциальность и целостность данных в БД в высшей степени касается DBA, управляющего текущей СУБД. Технология указывает способ для повышения производительности с помощью управления информацией. Современная технология также обеспечивает возможность распространять информацию по многим сайтам, затрудняя обеспечение контроля, безопасности и целостности данных. Информационная среда, состоящая из нескольких сайтов, обязывает DBA использовать механизмы обеспечения безопасности и целостности, предоставляемые СУБД для реализации политик администрирования БД, определенных так, как это было показано в предыдущем разделе. Кроме того, для предотвращения возможных внешних атак DBA должны совместно с экспертами по безопасности Интернета разрабатывать брандмауэры, прокси-серверы и другие механизмы безопасности.

Обеспечение безопасности и конфиденциальности данных в БД — функция руководства, обладающего соответствующими полномочиями. Полномочное руководство определяет процедуры защиты, гарантирующие безопасность и конфиденциальность информации в БД. Эти процедуры включают в себя (но не ограничиваются только этим) управление доступом пользователей, определение представлений, управление доступом к утилитам СУБД и наблюдение за использованием СУБД.

- *Управление доступом пользователей.* Эта функция предназначена для ограничения доступа к базе данных и должна включать, по крайней мере, следующие процедуры.
  - *Определение каждого пользователя в базе данных.* Это достигается на двух уровнях: на уровне операционной системы и на уровне СУБД. На уровне операционной системы DBA может потребовать создания регистрационного имени пользователя (login user ID), которое разрешает пользователю регистрироваться в системе. На уровне СУБД DBA может либо создать другое регистрационное имя пользователя, либо использовать для доступа к СУБД то же самое имя.
  - *Назначение пароля каждому пользователю.* Это также может быть выполнено как на уровне операционной системы, так и на уровне СУБД. Назначенный

пароль может иметь определенный срок действия. Это позволяет DBA периодически экранировать пользователя от БД и напоминать пользователю о необходимости периодической смены паролей, что затрудняет неавторизованный доступ к БД.

- *Определение групп пользователей.* Сортировка пользователей по различным группам в соответствии с общими требованиями по доступу к БД облегчает работу DBA по контролю и управлению привилегиями доступа отдельных пользователей.
  - *Назначение привилегий доступа.* DBA назначает привилегии доступа или права доступа к определенной БД для отдельных пользователей. Привилегии доступа описывают тип авторизованного доступа. Права доступа, например, могут быть ограничены только чтением, авторизованный доступ может включать привилегии READ (чтение), WRITE (запись) и DELETE (удаление). Привилегии доступа в реляционных базах данных назначаются с помощью команд SQL GRANT и REVOKE (см. гл. 5 и 8).
  - *Контроль физического доступа.* Физическая безопасность может защитить от несанкционированного доступа неавторизованных пользователей непосредственно к установке СУБД и оборудованию. Некоторые общеизвестные методы использования физической безопасности в больших базах данных включают в себя: безопасный вход, рабочие станции, защищенные паролем, персональные электронные идентификационные карточки, скрытую видеосъемку и средства распознавания голоса.
- *Определение представлений.* DBA должен определить представления данных для защиты и управления областью данных, доступной авторизованному пользователю. СУБД должна предоставлять инструментальные средства, позволяющие определять представления, состоящие из одной или более таблиц, и назначать пользователям или группам пользователей права доступа. Для определения представлений используется команда SQL CREATE VIEW.
- *Утилиты СУБД по управлению доступом.* Доступ к базе данных можно контролировать установкой ограничений на использование инструментальных средств СУБД по созданию запросов и отчетов. DBA должен гарантировать, что такой инструментарий будет использован должным образом и только авторизованными лицами.
- *Наблюдение за использованием СУБД.* DBA должен контролировать использование информации в БД. Некоторые пакеты СУБД имеют средства, позволяющие создавать *контрольный журнал* (audit log), в который автоматически записывается краткое описание операций с БД, выполняемых всеми пользователями. Такие сведения позволяют DBA определять нарушения прав доступа. Контрольный журнал можно настроить на запись всех попыток доступа к базе данных или только неудачных попыток.

### Примечание

Джеймс Мартин (James Martin) предложил замечательный перечень и описание желательных атрибутов стратегии безопасности базы данных, который остается актуальным и по сей день (James Martin, "Managing the Database Environment", Englewood Cliffs, NJ:

Prentice-Hall, 1977). Стратегия безопасности по Мартину основана на "семи основных элементах безопасности базы данных":

|                  |                      |
|------------------|----------------------|
| <b>Данные:</b>   | <b>Пользователи:</b> |
| Защищены         | Идентифицируемы      |
| Реконструируются | Авторизованы         |
| Контролируются   | Наблюдаемы           |
| Отказоустойчивы  |                      |

Бреши в защите могут привести БД в состояние, при котором ее целостность либо сохранена, либо нарушена.

- ❑ *Целостность БД сохранена.* Необходимы действия для предотвращения возникновения таких проблем, но восстановление данных может и не потребоваться. На самом деле, большинство случаев нарушения безопасности происходит по причине неавторизованного и незафиксированного доступа в целях извлечения какой-то информации из БД, а такое хищение данных не повреждает саму базу данных.
- ❑ *Целостность БД нарушена.* Необходимы действия по предотвращению подобных ситуаций, и базу данных придется приводить в устойчивое состояние. Нарушение безопасности, повлекшее за собой повреждение БД, может возникнуть при проникновении компьютерных вирусов или взломах защиты "хакерами", действия которых имеют своей целью разрушить или полностью изменить БД.

Целостность базы данных может быть утеряна из-за внешних факторов, находящихся вне контроля DBA. Например, база данных может быть повреждена из-за разрушения здания, пожара или землетрясения. В любом случае из-за угрозы повреждения базы данных задача создания резервных копий и восстановления БД становится для DBA критически важной.

## Резервное копирование данных и восстановление

В начале главы подчеркивалось, что информация имеет очень важное значение для предприятия и должна управляться с той же тщательностью, как и другие его активы. Если к данным невозможно получить доступ, то это может иметь губительные последствия для компании. Поэтому резервное копирование данных и процедуры восстановления являются очень важными для всех баз данных, а DBA должен гарантировать, что данные в БД могут быть полностью восстановлены в случае их физического повреждения или нарушения целостности БД.

Утеря данных может быть как тотальной, так и частичной. Частичная утеря данных может быть вызвана физической утерей части базы данных или нарушением целостности части базы данных. Тотальная утеря данных может означать, что несмотря на существование БД, ее целостность полностью нарушена — база данных полностью физически утеряна. В любом случае резервное копирование и восстановление — самые дешевые из всех возможных страховок.

Управление безопасностью БД, целостностью, резервным копированием и восстановлением настолько важно, что многие подразделения DBA создают специальную должность — *инспектор безопасности базы данных (database security officer, DSO)*. Единственная задача DSO — обеспечение безопасности и целостности базы

данных. В больших базах данных предприятия деятельность DSO зачастую называют управлением чрезвычайными ситуациями.

*Управление чрезвычайными ситуациями* (disaster management) включает в себя все действия DBA, направленные на обеспечение готовности данных при физических катастрофах или нарушениях целостности базы данных. Управление чрезвычайными ситуациями включает в себя разработку и тестирование плана действий в случае непредвиденных обстоятельств, а также проверку процедур восстановления. Мероприятия по резервному копированию и восстановлению должны включать в себя, по крайней мере, следующие действия.

- ❑ *Периодическое резервное копирование данных и приложений.* Некоторые СУБД имеют в своем составе инструментальные средства обеспечения резервного копирования и восстановления данных в БД. DBA должен использовать эти средства для автоматического резервного копирования и восстановления. Такие продукты, как, например, DB2 компании IBM, позволяют выполнять различные типы резервного копирования: полное, инкрементальное и параллельное. (При *полном резервном копировании* (full backup), которое также называется *дампом базы данных*, создается полная копия всей БД. *Инкрементальное копирование* (incremental backup) создает резервную копию всех данных, начиная с даты последнего резервного копирования. *Параллельное резервное копирование* (concurrent backup) выполняется в процессе работы пользователей с базой данных.)
- ❑ *Правильная маркировка резервной копии.* Резервные копии должны однозначно маркироваться с помощью детального описания и указания даты их создания, что позволяет DBA обеспечить использование нужной копии при восстановлении БД. Чаще всего в качестве носителя резервной копии используется магнитная лента; место хранения и маркировка таких магнитных лент должны тщательно продумываться оператором, а DBA должен контролировать использование и местоположение магнитных лент.
- ❑ *Удобное и безопасное хранение резервных копий.* Необходимо иметь несколько резервных копий с одинаковой датой создания, и каждая такая копия должна храниться в отдельном месте. Места хранения должны находиться как внутри предприятия, так и за его пределами (размещение различных резервных копий в одном месте, прежде всего, не соответствует требованию наличия нескольких резервных копий). Место хранения должно быть тщательно подготовлено и может представлять собой, например, огнеупорный, защищенный от землетрясений сейф, снабженный датчиками влажности и температуры. DBA должен разработать политику, позволяющую ответить на два вопроса: 1) *Где хранится резервная копия?* 2) *Сколько времени должна храниться резервная копия?*
- ❑ *Физическая защита программного и аппаратного обеспечения.* Такая защита может включать в себя установку компьютеров в помещениях с ограниченным доступом, а также оборудование места установки компьютеров кондиционерами, устройствами резервного электропитания и средствами противопожарной безопасности. Физическая защита также включает в себя оборудование резервного компьютера и СУБД, которые можно использовать при аварийной ситуации.
- ❑ *Персональный контроль доступа к программному обеспечению базы данных.* Для достоверной идентификации авторизованных пользователей или ресурсов необ-

ходимо использовать многоуровневые пароли и привилегии, а также аппаратные и программные средства аутентификации с запросом и подтверждением.

- ❑ *Страхование данных в БД.* DBA или инспектор по безопасности БД должны проводить страховую политику, чтобы обеспечить финансовую защиту в случае выхода БД из строя. Такая страховка может быть достаточно дорогой, но это дешевле, чем утрата всего созданного массива данных.

Необходимо упомянуть еще два момента.

- ❑ План восстановления данных в случае непредвиденных обстоятельств должен быть тщательно протестирован и оценен, его также необходимо периодически проверять на практике. Так называемые "пожарные учения" не стоит рассматривать как некое унижение, и высшее руководство должно оказывать всяческую поддержку таким мероприятиям.
- ❑ Программа резервного копирования и восстановления, скорее всего, не сможет охватить все компоненты информационной системы. Поэтому разумно подумать о приоритетах процесса восстановления данных.

## Распределение и использование данных

Информация значима, только если она предоставляется нужным пользователям в нужное время. DBA отвечает за доставку данных нужным пользователям в нужное время и в правильном формате. Задача распределения данных может отнимать у DBA много времени, особенно если доставка данных основана на программном обеспечении, когда пользователи зависят от программистов, поставляющих программы доступа к базе данных. Хотя Интернет и его расширения интранет и экстранет открыли базы данных для корпоративных пользователей, их применение создает множество новых проблем для DBA.

Современная теория распределения данных позволяет упростить доступ авторизованных пользователей к БД. Один из способов решения этой задачи состоит в использовании нового поколения более сложных инструментов запросов и новых интерфейсных программ Web. Это позволяет DBA научить пользователей получать необходимую информацию независимо от прикладных программистов. Естественно, DBA должен обеспечить применение соответствующих стандартов и процедур.

Такая идеология распределения данных в настоящее время считается общепринятой и, вероятно, будет получать дальнейшее распространение по мере развития технологии баз данных. Такая структура устраивает конечных пользователей. Очевидно, что получение пользователями относительной самостоятельности при использовании данных может привести к более эффективному использованию информации в процессе принятия решений. И все же такая "демократия данных" имеет побочные эффекты. Разрешая конечным пользователям управлять своими наборами данных на микроуровне, мы можем нарушить связь между пользователями и администрацией. Работа DBA в этих условиях может стать достаточно сложной, и может возникнуть угроза эффективности работы DBA. Может вновь буйно расцвести дублирование данных без какого-либо контроля на уровне предприятия по обеспечению уникальности элементов данных. При этом конечные пользователи, не до конца понимающие природу и источник информации, могут неправильно использовать элементы данных.

## 16.5.2. Технические функции DBA

Технические функции DBA требуют всестороннего понимания функций СУБД, конфигурации, языков программирования, моделирования данных и методологии проектирования, а также других проблем, связанных с использованием СУБД. Например, техническая деятельность DBA включает в себя выбор, установку, работу, обслуживание и обновление СУБД и программных утилит, а также проектирование, разработку и обслуживание прикладных программ взаимодействия с базой данных.

Большая часть технических функций DBA является логическим продолжением его организационной деятельности. Например, DBA имеет дело с безопасностью и целостностью БД, резервным копированием и восстановлением, обучением персонала и поддержкой пользователей. Поэтому двойственная роль DBA может быть концептуально представлена в виде капсулы, ядро которой, отражающее технические функции DBA, покрыто оболочкой организационной работы.

Технические аспекты работы DBA затрагивают следующие сферы деятельности:

- ☐ развитие СУБД и утилит, их установка и выбор;
- ☐ проектирование и реализация баз данных и приложений;
- ☐ тестирование и оценка баз данных и приложений;
- ☐ работа СУБД, утилит и приложений;
- ☐ обучение и поддержка пользователей;
- ☐ обслуживание СУБД, утилит и приложений.

В следующем разделе эти направления деятельности будут рассмотрены подробно.

### Развитие СУБД и утилит, их установка и выбор

Одна из наиболее важных сторон технической деятельности DBA — выбор системы управления базой данных, программных утилит и аппаратуры поддержки, которая будет использоваться на предприятии. Поэтому DBA должен разработать и выполнять план сравнительной оценки и приобретения СУБД, утилит и оборудования. Оценка и приобретение должны базироваться в основном на потребностях предприятия, а не на специфических возможностях программного и аппаратного обеспечения. DBA должен понимать, что необходимо искать решение проблем, а не компьютеры или программное обеспечение СУБД. Проще говоря, СУБД это инструмент управления, а не техническая игрушка.

При составлении плана оценки и приобретения, прежде всего, необходимо выявить потребности компании. Чтобы выявить полную картину этих потребностей, DBA должен убедиться, что все пользователи, включая руководство высшего и среднего уровней, привлечены к решению этой задачи. После того как потребности определены, цель DBA становится вполне ясной и можно определить критерии выбора СУБД и ее возможностей.

Чтобы возможности СУБД соответствовали потребностям предприятия, DBA должен разработать контрольный список желательных возможностей СУБД. В этом списке должны найти отражение, по крайней мере, следующие вопросы.

- ☐ *Модель СУБД.* Какая модель СУБД лучше всего соответствует потребностям компании — реляционная, объектно-ориентированная или объектно-реляционная?



Если это будет приложением хранилища данных, нужно ли при этом использовать реляционную или многомерную СУБД?

- ❑ *Емкость СУБД.* Какие необходимо предусмотреть максимальные размеры дискового пространства и БД? Сколько пакетов дисков нужно поддерживать? Сколько устройств записи на магнитную ленту следует приобрести?
- ❑ *Поддержка разработки приложений.* Нужна ли поддержка языков программирования 3GL и 4GL? Какие доступны инструментальные средства разработки приложений (проектирование схемы БД, словарь данных, наблюдение за производительностью, палитры экранов и меню)? Имеются ли инструментальные средства создания запросов конечных пользователей? Обеспечивает ли СУБД доступ к данным с помощью Web-интерфейса?
- ❑ *Безопасность и целостность.* Поддерживает ли СУБД правила целостности на уровне ссылок и сущностей, права доступа и т. д.? Поддерживает ли СУБД контрольные журналы для учета ошибок и нарушений безопасности? Можно ли изменять размер контрольного журнала?
- ❑ *Резервное копирование и восстановление.* Обеспечивает ли СУБД автоматическое создание резервной копии и восстановление? Поддерживает ли СУБД резервное копирование на магнитные ленты, оптические диски или CD? Может ли СУБД автоматически создавать резервные копии журналов транзакций?
- ❑ *Контроль параллельного выполнения.* Поддерживает ли СУБД работу нескольких пользователей? Какой уровень изолирования предлагает СУБД (таблица, страница, строка)? Как много ручного кодирования необходимо при разработке прикладных программ?
- ❑ *Производительность.* Сколько транзакций в секунду может выполнять СУБД? Нужны ли дополнительные процессоры транзакций?
- ❑ *Средства администрирования БД.* Предлагает ли СУБД различные типы интерфейсов для DBA? Какие типы сведений предоставляет интерфейс DBA? Обеспечивает ли СУБД выдачу предупреждающих сообщений DBA при возникновении ошибок или нарушении безопасности?
- ❑ *Способность к взаимодействию и распределение данных.* Может ли СУБД работать с другими типами СУБД в той же среде? Какой обеспечивается уровень совместного использования и взаимодействия? Поддерживает ли СУБД операции READ и WRITE с другими СУБД? Поддерживает ли СУБД клиент/серверную архитектуру?
- ❑ *Переносимость и стандартизация.* Может ли СУБД выполняться на различных платформах и в разных операционных системах? Может ли СУБД выполняться на мэйнфреймах, мини-компьютерах и персональных компьютерах? Могут ли приложения СУБД выполняться без дополнительной модификации на всех платформах? Каким национальным и промышленным стандартам соответствует СУБД?
- ❑ *Оборудование.* Какое оборудование необходимо для установки и работы СУБД?
- ❑ *Словарь данных.* Имеется ли в СУБД словарь данных? Если да, то какая информация в нем хранится? Имеется ли интерфейс СУБД со словарем данных? Поддерживает ли СУБД CASE-инструментарий?

- ❑ *Поддержка и обучение со стороны поставщика.* Предлагает ли поставщик обучение на своем оборудовании? Какой тип и уровень поддержки обеспечивает поставщик? Хорошо ли составлена техническая документация по СУБД? Какова политика обновления у поставщика?
- ❑ *Доступные инструментальные средства сторонних организаций.* Инструментальные средства какого типа предлагаются сторонними поставщиками для данной СУБД (средства создания запросов, словари данных, управление доступом, управление хранением и т. д.)?
- ❑ *Стоимость.* Сколько стоит программное обеспечение и оборудование? Сколько дополнительных работников потребуется и каков должен быть их уровень подготовки? Каков размер периодических издержек? Каков предполагаемый срок окупаемости?

Необходимо в процессе выбора взвесить все "за" и "против" различных решений. Доступных вариантов обычно немного, поскольку программное обеспечение должно быть совместимо с существующими компьютерными системами предприятия. Необходимо также помнить, что СУБД это только часть решения: она требует вспомогательного оборудования, прикладного программного обеспечения и программных утилит. Например, использование СУБД, скорее всего, ограничивается доступным центральным процессором, внешним процессором, вспомогательными устройствами хранения, устройствами коммуникации, операционной системой, системой обработки транзакций и т. д. Стоимость вспомогательных аппаратных и программных компонентов также должна быть включена в общую оценку.

В процессе выбора необходимо учитывать стоимость подготовки места размещения. Например, DBA должен учесть как единовременные, так и постоянные расходы по подготовке и обслуживанию компьютерного кабинета.

DBA должен наблюдать за установкой всего программного и аппаратного обеспечения, предназначенного для поддержки стратегии администрирования; должен иметь исчерпывающее представление об установленных компонентах; должен быть знаком с процедурами установки, конфигурирования и запуска этих компонентов. Процедуры установки включают такие сведения, как местоположение журналов резервного копирования и транзакций, конфигурационная информация локальной сети, сведения о физических свойствах устройств хранения и т. д.

Установка и конфигурирование зависят от конкретной СУБД, поэтому их невозможно здесь описать. Все необходимые сведения нужно искать в разделе установки и конфигурирования руководства по администрированию вашей СУБД.

## **Проектирование и реализация баз данных и приложений**

Функции DBA также предоставляют сервисы проектирования и моделирования данных для конечных пользователей. Такие сервисы часто координируются с группой разработки приложений внутри отдела обработки информации. Поэтому одно из основных направлений деятельности DBA — определение и реализация необходимых стандартов и процедур. Как только стандарты и процедуры для всей структуры определены, DBA должен обеспечить, чтобы моделирование БД и работы по проек-

тированию выполнялись именно в этой структуре. DBA затем предоставляет необходимую помощь и поддержку в процессе проектирования БД на концептуальном, логическом и физическом уровнях (вспомните, что концептуальное проектирование не зависит от СУБД и оборудования, логическое проектирование зависит от СУБД и не зависит от оборудования, а физическое проектирование зависит как от оборудования, так и от СУБД).

DBA, как правило, требует, чтобы к моделированию и проектированию БД допускались определенные лица. Такие люди должны организоваться в группы в соответствии со сферой действия приложения. Например, персонал, занимающийся проектированием и моделированием БД, может быть приписан к производственным системам, финансовым и организационным системам, системам поддержки решений и их исполнения и т. д. DBA планирует проектные работы для координации проектирования и моделирования данных. Такая координация может потребовать перераспределения доступных ресурсов на основе внешних приоритетов.

DBA также обеспечивает проектирование транзакций БД и предоставляет прикладным программистам службы обеспечения качества и целостности данных. Такая поддержка включает исследование проектов приложений БД с целью обеспечения следующих условий при выполнении транзакций.

- ☐ *Корректность.* Транзакции должны отражать события реального мира.
- ☐ *Эффективность.* Транзакции не должны перегружать СУБД.
- ☐ *Соответствие стандартам и требованиям целостности.*

Эти действия требуют от персонала определенных навыков проектирования БД и программирования.

Для реализации приложений нужна физическая реализация БД. Поэтому DBA должны обеспечить помощь и контроль в течение физического проектирования, включая определение и выделение необходимой памяти для хранения данных, загрузку данных, конвертирование и службы перемещения данных. В задачи реализации DBA также входят создание, компиляция и хранение плана доступа к приложениям. *План доступа (access plan)* представляет собой хранимую процедуру, в процессе компиляции предопределяющую способ, которым приложение будет получать доступ к базе данных во время работы системы (для того чтобы создавать и проверять план доступа, пользователь должен иметь соответствующие права доступа к БД).

Перед запуском приложения DBA должен разработать, протестировать и реализовать рабочие процедуры, необходимые новой системе. Эти рабочие процедуры включают в себя планы обучения, обеспечения безопасности, а также резервного копирования и восстановления и, кроме того, назначение ответственных за контроль и обслуживание БД. Наконец, DBA должен авторизовать пользователей приложения на доступ к базе данных, из которой приложение извлекает необходимую информацию.

Добавление новой базы данных может потребовать точной настройки и/или переконфигурирования СУБД. Помните, что СУБД помогает всем приложениям, управляя корпоративным складом информации общего доступа. Поэтому если добавляются или модифицируются структуры данных, то СУБД могут потребоваться новые ресурсы для эффективного обслуживания новых пользователей.

## Тестирование и оценка баз данных и приложений

ДБА должен также обеспечить тестирование и оценку служб для всех приложений баз данных и конечных пользователей. Эти службы представляют собой логические расширения служб проектирования, разработки и реализации, описанных в предыдущем разделе. Очевидно, процедуры и стандарты тестирования должны иметься в наличии до того, как прикладная программа будет одобрена для использования на предприятии.

Хотя службы тестирования и оценки тесно связаны со службами проектирования и реализации базы данных, они обычно обслуживаются независимо от них. Причина такого разделения кроется в том, что прикладные программисты и проектировщики зачастую слишком тесно привязаны к задачам, которые необходимо исследовать на наличие ошибок и недостатков.

Тестирование обычно начинается с загрузки тестовой модели БД. Такая база данных содержит тестовые данные для приложений, и цель ее — проверка определений данных, соблюдения правил целостности данных в БД и приложениях.

Тестирование и оценка приложений баз данных охватывают все аспекты системы — от простого накопления информации до ее использования и удаления. Процесс оценки включает в себя следующие направления.

- ☐ *Технические аспекты*, как приложений, так и баз данных. Необходимо оценить возможности резервного копирования и восстановления, безопасности и целостности, а также эффективности приложения.
- ☐ *Оценка документации* позволяет убедиться в том, что документация и процедуры в точности соответствуют друг другу.
- ☐ *Соблюдение стандартов* именования, документирования и кодирования.
- ☐ *Конфликты дублирования* данных с имеющимися данными.
- ☐ *Реализация всех правил* проверки данных.

После проведения тестирования всех приложений, баз данных и процедур система объявляется работоспособной и становится доступной конечным пользователям.

## Работа СУБД, утилит и приложений

Работу СУБД можно подразделить на четыре основных направления:

- ☐ поддержка системы;
- ☐ наблюдение за производительностью и настройка;
- ☐ резервное копирование и восстановление;
- ☐ контроль и наблюдение за безопасностью.

Действия по *поддержке системы* охватывают все задачи, напрямую связанные с ежедневными операциями СУБД и ее приложений. Диапазон этих задач — от заполнения рабочих журналов до смены магнитных лент для проверки состояния компьютерного оборудования, дисковых пакетов, резервных источников питания и т. д. Действия, связанные с поддержкой работоспособности системы, включают в себя периодические задачи, такие как запуск специальных программ и конфигурирование ресурсов для новых и/или обновленных версий приложений баз данных.

*Наблюдение за производительностью и настройка* требуют от DBA внимания и времени. Эти действия предназначены для обеспечения удовлетворительного уровня производительности работы утилит СУБД и приложений. Для выполнения наблюдения за производительностью и задач настройки DBA должен:

- определить показатели производительности СУБД;
- наблюдать за СУБД, оценивая соответствие ее производительности заданным значениям показателей;
- изолировать проблему и найти альтернативное решение (если производительность не соответствует требованиям);
- реализовать выбранное решение по повышению производительности.

В СУБД часто включаются средства наблюдения за производительностью, которые позволяют DBA запрашивать информацию об использовании БД. Если в СУБД нет таких средств, их можно получить из других источников — утилит СУБД, предоставляемых третьими фирмами, либо входящих в состав операционной системы или процессора транзакций. Большинство средств наблюдения за производительностью позволяют DBA выявить узкие места системы. Наиболее общими способами повышения производительности СУБД являются использование индексов, алгоритмов оптимизации запросов и управление ресурсами хранения.

Поскольку неверный выбор индекса может повлиять на производительность системы, в большинстве СУБД придерживаются строго определенного плана создания и определения индексов. Такой план позволяет DBA определять процедуры, гарантирующие надлежащее определение и использование индекса; такой план особенно важен в реляционных базах данных.

DBA, как правило, тратит много времени, пытаясь обучить программистов и конечных пользователей надлежащему использованию SQL-операторов для обеспечения лучшей производительности. Обычно "Руководство по программированию СУБД" и "Руководство по администрированию" содержат полезные сведения о производительности и примеры, демонстрирующие правильное использование SQL-операторов, как в режиме командной строки, так и в приложениях. Поскольку реляционные системы не позволяют пользователю индексировать выбор внутри запроса, СУБД выбирает индекс для пользователя. Поэтому DBA должен создавать индексы, которые могут повышать производительность системы (см. примеры по повышению производительности системы в *гл. 8*).

Программы, повышающие производительность системы, обычно интегрируются в дистрибутивный пакет СУБД, поэтому в них доступно много опций настройки. Программы оптимизации запросов ориентированы на улучшение одновременного доступа к БД. Некоторые базы данных позволяют DBA определять параметры для задания желательного уровня параллельного выполнения транзакций. На параллельный доступ также влияют типы блокировок, используемые в СУБД и устанавливаемые приложениями. Поскольку проблема одновременного доступа очень важна для эффективной работы системы, DBA должен быть знаком с факторами, влияющими на параллельный доступ (подробную информацию о параллельном доступе см. в *гл. 9*).

Доступные ресурсы хранения, как в первичной, так и во вторичной памяти, также должны приниматься во внимание во время настройки производительности СУБД.

Размещение ресурсов хранения определяется при конфигурировании СУБД. Параметры конфигурации хранения используются для определения:

- ☐ числа одновременно открытых баз данных;
- ☐ числа одновременно работающих прикладных программ или пользователей;
- ☐ размера первичной памяти (буфера), назначаемой процессу каждой базы данных;
- ☐ размера и местоположения файлов журналов (Вспомните, что эти файлы используются при восстановлении БД. Файлы журналов могут быть размещены на отдельном томе для уменьшения перемещения головок диска и увеличения производительности.)

Проблемы наблюдения за производительностью связаны с используемой СУБД. Поэтому DBA должен быть знаком с руководством по использованию СУБД для выяснения технических подробностей, связанных с этой задачей.

Поскольку утрата данных, скорее всего, разрушительна для предприятия, *работы по резервному копированию и восстановлению* являются первостепенными при эксплуатации СУБД. DBA должен установить распорядок резервного копирования базы данных и файлов журналов в определенные интервалы времени. Частота резервного копирования зависит от типа приложения и от относительной значимости данных. Необходимо периодически создавать резервные копии всех важнейших компонентов системы (базы данных, приложений базы данных и журналов транзакций).

Все большее число дистрибутивных пакетов СУБД включают утилиты, автоматически выполняющие резервное копирование БД, как полное, так и инкрементальное. Хотя инкрементальное резервное копирование выполняется конечно же быстрее, чем полное резервное копирование, все же при инкрементальном копировании необходимо наличие полной копии, которую можно использовать для восстановления.

Восстановление БД после выхода из строя устройств или системы требует использования журнала транзакций для корректировки копии БД. DBA должен планировать, реализовать, тестировать и выполнять "пуленепробиваемые" процедуры создания резервных копий и восстановления.

*Контроль и наблюдение за безопасностью* предполагают корректное назначение прав доступа и надлежащее использование таких привилегий доступа программистами и конечными пользователями. Технические аспекты проверки безопасности и наблюдения за ней находят свое отражение в присвоении пользователям прав пользования, в использовании SQL-команд по назначению, отмене прав доступа пользователям и объектам данных, использовании контрольных журналов для выявления нарушений безопасности или попыток их нарушения. DBA должен периодически выполнять отчеты на основе информации контрольных журналов для выявления реальных попыток нарушения безопасности и, при их наличии, для выяснения того, с какого рабочего места это было сделано и, по возможности, кем.

## Обучение и поддержка пользователей

Обучение персонала пользованию СУБД и ее инструментальными средствами также входит в обязанности DBA. Кроме того, DBA обеспечивает обучение прикладных программистов использованию СУБД и ее утилит. Обучение прикладных програм-

мистов охватывает изучение применения инструментария СУБД, а также процедур и стандартов, необходимых для программирования в среде баз данных.

Незапланированное обучение пользователей и программистов по требованию также входит в обязанности DBA. Для обеспечения такой поддержки разрабатываются процедуры разрешения технических проблем. Техническая процедура может включать в себя разработку технической базы данных, которая нужна для поиска решений самых распространенных технических проблем.

Частично поддержка обеспечивается взаимными соглашениями с поставщиком СУБД. Установление хороших отношений с поставщиками программного обеспечения — одна из гарантий того, что компания всегда будет иметь надежный внешний источник помощи. Поставщики всегда предоставляют самую свежую информацию о новых продуктах и необходимости переподготовки персонала. Хорошие взаимоотношения между компанией и поставщиками, вероятно, дадут предприятию возможность определить направление будущего развития разработок в области баз данных.

## **Обслуживание СУБД, утилит и приложений**

Поддержка системы является частью повседневных обязанностей DBA. Поддержка системы направлена на сохранение структуры СУБД.

Периодическое обслуживание СУБД включает в себя управление физическими или вторичными устройствами хранения. Одно из самых распространенных направлений этой деятельности — реорганизация размещения данных в БД (выполняется, как правило, во время настройки СУБД). Реорганизация базы данных должна быть выполнена так, чтобы дисковые страницы располагались подряд для повышения производительности. Процесс реорганизации также должен высвободить дисковое пространство, связанное с удаленными данными, что позволит разместить больше новой информации.

Работы по обслуживанию включают в себя обновление СУБД и программного обеспечения. При обновлении могут потребоваться установка новых версий СУБД или интерфейса Интернета, а также создание дополнительных шлюзов по доступу к СУБД, выполняющейся на другом хост-компьютере. Службы шлюзов СУБД очень часто используются в распределенных СУБД, выполняющихся в клиент/серверной среде. Также БД нового поколения включают в себя средства работы с пространственными данными, создание хранилищ данных и средств запросов по схеме "звезда", а также поддержку интерфейса Java-программирования для обеспечения доступа к Интернету.

Довольно часто компании сталкиваются с потребностью обмена данными различных форматов или между базами данных. Действия DBA включают в себя обеспечение служб миграции и преобразования данных несовместимых форматов или различных СУБД. Такие условия часто встречаются при обновлении системы до новой версии или когда существующая СУБД полностью заменяется новой. К преобразованию базы данных также относится загрузка данных из основной (размещенной, например, на мэйнфрейме) СУБД на персональные компьютеры, чтобы их пользователи могли выполнять различные действия — анализ с помощью электронных таблиц, создание диаграмм, статистическое моделирование и т. д. Службы перемещения и преобразования могут выполняться на логическом уровне СУБД (в зависимости от СУБД или ее программного обеспечения) или на физическом (в зависимости от устройств хранения или операционной системы).

## 16.6. Инструментальные средства администрирования баз данных

Вряд ли будет преувеличением считать словарь данных одним из основных инструментальных средств DBA. Хотя мы уже описали функции словаря данных в гл. 2 и проиллюстрировали его использование в примерах к гл. 2 и 8, его использование в административных целях необходимо обсудить дополнительно. В этом разделе мы исследуем словарь данных как средство администрирования, а также CASE-инструментарий, используемый DBA для проведения анализа БД во время ее проектирования.

### 16.6.1. Словарь данных

В гл. 1 было определено, что словарь данных представляет собой компонент СУБД, в котором хранятся определения свойств данных и связей. Вспомните, что такие "данные о данных" называются *метаданными*. Словарь данных СУБД обеспечивает возможность самоописания свойств СУБД. На самом деле, словарь данных напоминает рентген для всего набора данных компании и является важнейшим элементом административных функций.

Существуют два основных типа словарей данных: *интегрированный* и *самостоятельный*. Интегрированный словарь данных включается в СУБД. Например, в состав всех реляционных СУБД (РСУБД) входит встроенный словарь данных или системный каталог, доступ к которому и его обновление осуществляются РСУБД. Другие СУБД, особенно старые, не имеют встроенных словарей данных, вместо этого DBA может использовать системы самостоятельных словарей баз данных сторонних фирм.

Словари данных подразделяются на активные и пассивные. *Активный словарь данных* автоматически обновляется СУБД в каждом случае доступа к данным, что позволяет постоянно обновлять информацию. *Пассивные словари* данных автоматически не обновляются, это выполняется периодически с помощью пакетных процедур.

Основная функция словарей данных — хранение описаний всех объектов, взаимодействующих с базой данных. В интегрированных словарях данных информация, как правило, ограничивается метаданными, которые управляются СУБД. Системы самостоятельных словарей данных обычно более гибки и позволяют DBA управлять всей информацией предприятия и описывать ее не только в компьютерной, но и в любой другой форме. Каков бы ни был формат словаря данных, его существование предоставляет проектировщикам баз данных и конечным пользователям самые передовые способы коммуникации. Кроме того, словарь данных помогает DBA разрешать конфликты данных.

Хотя не существует стандартного формата для хранения информации в словаре данных, есть некоторые общие подходы. Например, в словаре данных обычно хранится следующая информация.

- Элементы данных, определенные во всех таблицах и во всех базах данных. Конкретно в словаре данных хранятся имена, типы данных, форматы представления, форматы внутреннего хранения и правила проверки. Словарь данных указывает на способ использования элемента данных, кем он используется и т. д.



- ❑ Таблицы, определенные во всех базах данных. Например, в словаре данных, скорее всего, хранятся имя создателя таблицы, дата ее создания, авторизация доступа и число столбцов.
- ❑ Индексы, определенные в каждой таблице базы данных. Для каждого индекса СУБД хранит, по крайней мере, имя индекса, атрибуты, используемые в индексе, местоположение, специфичные свойства индекса и дату создания.
- ❑ Созданные базы данных: кто создал ту или иную БД, дата ее создания, местоположение базы данных, кто является DBA и т. д.
- ❑ Конечные пользователи и администраторы базы данных.
- ❑ Программы, получающие доступ к базе данных, включая экранные формы, отчетные формы, прикладные программы, SQL-запросы и т. д.
- ❑ Авторизация доступа для всех пользователей всех баз данных.
- ❑ Связи между элементами данных, включая все вовлеченные в них элементы независимо от того, являются они обязательными или нет, связности и мощности связей и т. д.

Если словарь базы данных можно приспособить для включения внешних данных, он становится особенно удобным средством для общего управления корпоративными ресурсами. Управление таким расширенным словарем данных позволяет использовать и размещать всю информацию предприятия невзирая на то, берет она свое начало в БД или нет. Вот почему некоторые руководители рассматривают словарь данных как ключевой элемент управления информационными ресурсами. И вот почему словарь данных часто называют *словарем информационных ресурсов (information resource dictionary)*.

Метаданные, хранящиеся в словаре данных, часто считаются основой для наблюдения за состоянием БД, для применения и назначения прав доступа пользователям базы данных. Для хранения информации в словаре данных обычно используется формат реляционных таблиц, что позволяет DBA обращаться к базе данных с помощью SQL-запросов. Например, SQL-команды могут использоваться для извлечения информации о пользователях отдельных таблиц или о правах доступа отдельных пользователей. Мы продемонстрируем такие запросы с помощью следующих таблиц системного каталога IBM OS/2 Database Manager v1.3.

- ❑ SYSTABLES — хранит одну строку для каждой таблицы или представления.
- ❑ SYSCOLUMNS — хранит одну строку для каждого столбца таблицы или представления.
- ❑ SYSTABAUTH — хранит одну строку для каждой авторизации пользователя для данной таблицы или представления в БД.

## Примеры использования словаря данных

### Пример 1

Получение списка имен и дат создания всех таблиц, созданных пользователем JONESVI в текущей базе данных.

```
SELECT NAME
FROM SYSCOLUMNS
WHERE TBCreator = 'JONESVI';
```

### *Пример 2*

Получение списка имен столбцов для всех таблиц, созданных пользователем JONESVI в текущей базе данных.

```
SELECT NAME
FROM SYSCOLUMNS
WHERE TBCreator = 'JONESVI';
```

### *Пример 3*

Получение списка имен всех таблиц, для которых у пользователя JONESVI есть право на удаление (DELETE).

```
SELECT TTNAME
FROM SYSTABAUTH
WHERE GRANTEE = 'JONESVI' AND DELETEAUTH = 'Y';
```

### *Пример 4*

Получение списка всех пользователей, имеющих какие-либо права доступа к таблице INVENTORY.

```
SELECT DISTINCT GRANTEE
FROM SYSTABAUTH
WHERE TTNAME = 'INVENTORY';
```

### *Пример 5*

Получение списка имен пользователей и таблиц для всех пользователей, которые могут изменять структуру базы данных для любой таблицы в БД.

```
SELECT GRANTEE, TTNAME
FROM SYSTABAUTH
WHERE ALTERAUTH = 'Y'
ORDER BY GRANTEE, TTNAME;
```

Из приведенных примеров следует, что словарь данных можно использовать в качестве средства наблюдения за безопасностью данных в БД, проверяя назначение прав доступа к информации. Хотя предыдущие примеры ориентированы на таблицы баз данных и пользователей, из словаря данных можно также извлечь информацию о приложениях, получающих доступ к БД.

DBA может использовать словарь данных для выполнения анализа данных и проектирования. Например, DBA может создать отчет, в котором перечислены все элементы данных, используемые в определенных приложениях; список всех пользователей, получающих доступ к отдельным программам; отчет, в котором проверяются избыточность данных, дублирование, использование омонимов и синонимов — и другие отчеты, в которых представлены пользователи информации, доступ к данным и структуры данных. Словарь данных можно использовать для обеспечения использования прикладными программистами стандартов именования элементов данных в БД и корректного применения проверки данных. Таким образом, словарь данных

можно использовать для обеспечения широкого спектра работ по администрированию данных.

Ранее в этом разделе было упомянуто, что словари данных используются для проектирования и реализации информационных систем. Интегрированные словари данных также предназначены для использования в инструментальных средствах CASE.

## 16.6.2. Инструментальные средства CASE

CASE — аббревиатура от Computer-Aided Software Engineering (*система автоматизированной разработки программ*). CASE-инструментарий предоставляет автоматизированную структуру для жизненного цикла разработки систем (systems development life cycle, SDLC). Средства CASE основаны на использовании структурированных технологий мощного графического интерфейса. Автоматизируя многие рутинные действия системного проектирования и реализации, CASE-инструментарий играет очень важную роль в разработке информационных систем.

CASE-инструментарий обычно можно классифицировать в соответствии с поддержкой, какую он оказывает SDLC. Например, CASE-инструментарий для создания интерфейса пользователя предоставляет поддержку для этапов планирования, анализа и проектирования, CASE-инструментарий для серверных приложений обеспечивает поддержку на этапах кодирования и реализации. К преимуществам CASE можно отнести:

- ☐ сокращение времени и снижение затрат на разработку;
- ☐ автоматизацию SDLC;
- ☐ стандартизацию технологии разработки систем;
- ☐ простоту обслуживания прикладных систем разработки с помощью CASE.

Один из основных компонентов CASE-инструментария — расширенный словарь данных, который отслеживает все объекты, созданные проектировщиками системы. Например, в словаре данных CASE хранятся описания схем информационных потоков, структурные схемы, все внутренние и внешние сущности, склады информации, отчетные формы, экранные формы и т. д. В словаре данных CASE также описываются связи между компонентами системы.

Некоторые средства CASE обеспечивают интерфейсы с СУБД. Эти интерфейсы позволяют CASE-инструментарии хранить информацию словарей данных с помощью СУБД. Такое взаимодействие CASE/СУБД демонстрирует независимость разработки систем и разработки базы данных и помогает создавать полностью интегрированную среду разработки.

В среде разработки CASE проектировщики баз данных и приложений используют CASE-инструментарий для хранения описаний схемы БД, элементов данных, прикладных процессов, экранов, отчетов и других данных, связанных с процессом разработки. CASE-инструментарий интегрирует всю информацию разработки системы в едином архиве, который DBA может проверять на непротиворечивость и точность.

Кроме того, CASE-инструментарий расширяет и улучшает качество взаимодействия между DBA, прикладными программистами и конечными пользователями. DBA может взаимодействовать с CASE-инструментарием для проверки определе-

ния схемы данных для приложений, наблюдения за исполнением соглашения об именовании, дублированием элементов данных, проверкой использования правил для элементов данных, а также выполнять ведущие функции по разработке и поддержке других объектов управления. Когда с помощью CASE выявляются конфликты, нарушения правил и непротиворечивости, с его помощью проще выполнить исправления. Более того, средства CASE каскадом передают исправления по всей инфраструктуре приложения, что значительно упрощает работу DBA и прикладных проектировщиков.

## Коммерческие инструментальные средства CASE

Обычно инструментальные средства CASE состоят из пяти компонентов:

- ☐ графические средства, предназначенные для разработки структурных диаграмм, таких, например, как схемы информационных потоков и ER-диаграммы;
- ☐ блоки раскраски и создания отчетов для разработки информационных системных форматов ввода/вывода (например, интерфейса конечного пользователя);
- ☐ интегрированный архив для хранения и перекрестных ссылок данных проекта системы. Этот архив включает в себя обширный словарь данных;
- ☐ сегмент анализа для обеспечения полностью автоматизированной проверки непротиворечивости системы, ее синтаксиса и полноты;
- ☐ генератор программной документации.

Рис. 16.7 иллюстрирует, как можно использовать CASE-инструментарий Visio для разработки ER-диаграмм.

Одно из инструментальных средств CASE, система PLATINUM Erwin корпорации PLATINUM Technology, позволяет создавать полностью документированные ER-диаграммы, которые можно отображать на различном уровне абстракции. Кроме того, Erwin позволяет выполнять детальный реляционный дизайн: пользователь просто определяет атрибуты и первичные ключи для каждой сущности и описывает связи. Система Erwin затем назначает внешние ключи на основе заданных связей между сущностями. Изменения в первичных ключах всегда автоматически обновляются во всей системе.

Основные поставщики СУБД, такие как Oracle, предоставляют полностью интегрированный CASE-инструментарий для собственного программного обеспечения СУБД, а также для РСУБД других производителей. CASE-инструментарий Oracle можно совместно использовать, например, с DB2 компании IBM, SQL/DS, SQL Server и Rdb и RMS корпорации Dec для выполнения полностью документированных проектов БД. Некоторые поставщики берут за основу не реляционные СУБД, разрабатывают их схемы и автоматически производят эквивалентный реляционный проект.

Несомненно, CASE повышает производительность проектировщиков базы данных и прикладных программистов. Но вне зависимости от изощренности CASE-инструментария его пользователи должны быть хорошо знакомы с идеями концептуального проектирования. В неопытных руках CASE-инструментарий может всего лишь представить плохой проект в красивой обертке.

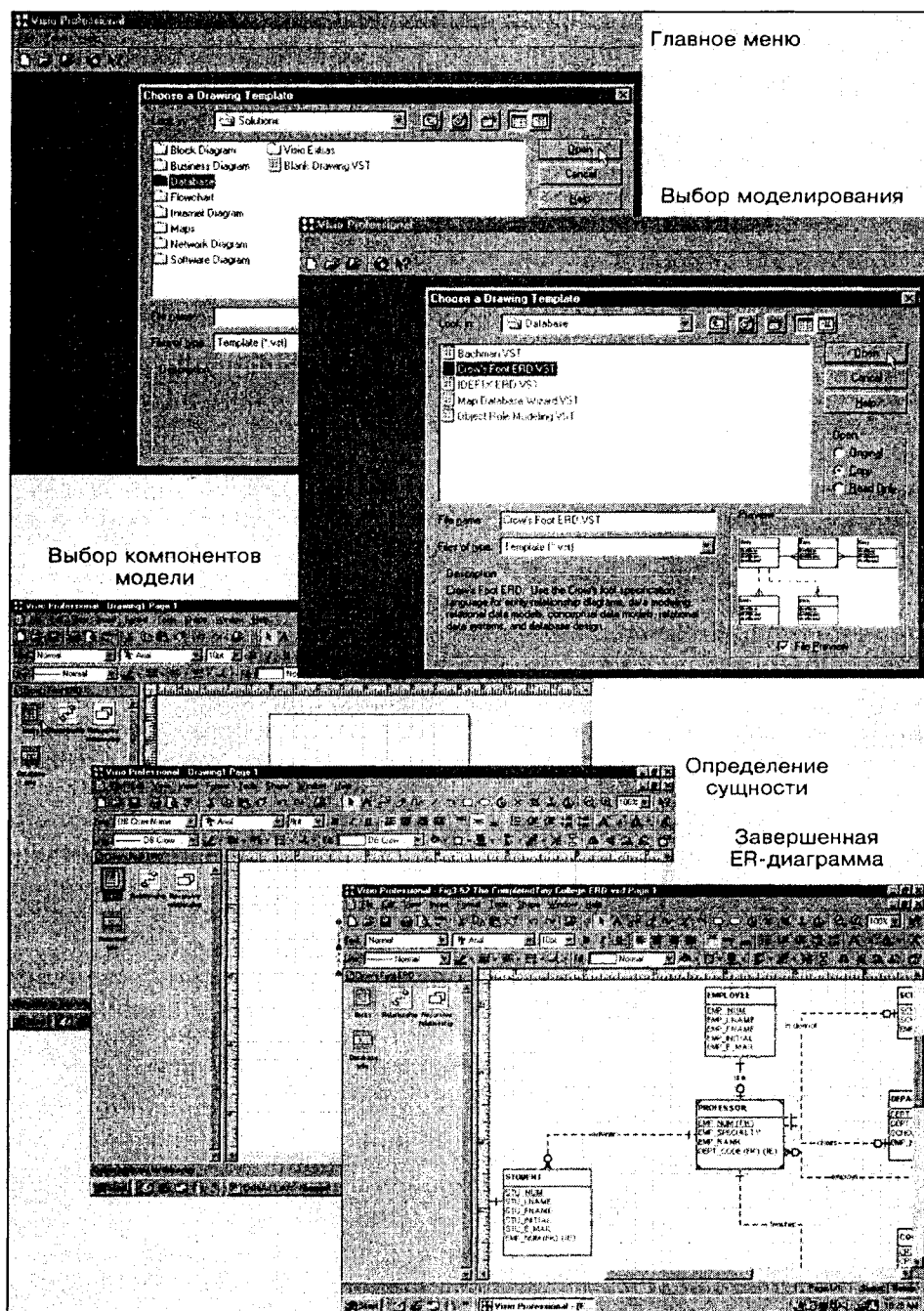


Рис. 16.7. Пример использования инструментальных средств CASE: VISIO

## 16.7. Разработка стратегии администрирования данных

Для успеха компании ее деятельность должна соответствовать главному предназначению. Поэтому независимо от размеров предприятия важный шаг для любой организации — обеспечение поддержки информационной системой стратегических планов для каждой сферы деятельности компании.

Стратегия администрации базы данных не должна противоречить планам информационной системы, поскольку планы информационной системы получены на основе детального анализа целей компании, ее положения или состояния и бизнес-потребностей. Некоторые методологии позволяют обеспечить совместимость данных администрирования и планов информационных систем для управления разработкой стратегического плана. Наиболее часто используемая методология называется *информационным инжинирингом*, или *информационной техникой*, *инфотехникой* (information engineering, IE).

Информационный инжиниринг (IE) позволяет транслировать стратегические цели предприятия в данные приложения, помогающие компании достигнуть поставленных целей. Логическое объяснение возникновения IE просто: типы бизнес-данных остаются достаточно стабильными и не сильно изменяются за время их существования. В отличие от них, процессы меняются часто и потому требуют частой модификации существующей системы. Сосредоточив внимание на данных, IE помогает уменьшить воздействие на систему при изменении процессов.

Результат работы IE-процесса — *архитектура информационной системы (ISA, information systems architecture)*, которая служит основой для планирования, разработки и управления будущими информационными системами. На рис. 16.8 показаны силы, оказывающие влияние на разработку ISA.

Реализация IE-технологий в организации — процесс дорогостоящий, требующий тщательного планирования, привлечения ресурсов, выполнения договоренностей, корректной постановки целей, выявления критических факторов и контроля. ISA предоставляет структуру, в которую включено использование компьютеризованных, автоматизированных и интегрированных средств, таких как СУБД и CASE.

Успех всей стратегии информационной системы, а значит, стратегии администрирования данных, зависит от нескольких важных факторов. Знание этих факторов помогает DBA разработать успешную стратегию администрирования корпоративных данных. Важнейшие факторы успеха включают в себя организационные, технологические и корпоративные проблемы. Особое внимание следует обратить на следующие показатели.

- ❑ *Управление обязательствами.* Управление договоренностями на высшем уровне необходимо для проведения в жизнь стандартов, процедур, планов и полномочий. Высшее руководство должно быть тому примером.
- ❑ *Исчерпывающий анализ положения компании.* Для понимания положения компании и получения ясного представления о дальнейших действиях необходимо проанализировать текущее положение дел в сфере администрирования данных. Например, как управлять анализом, проектированием, документированием, реа-

лизацией, стандартизацией, кодированием и другими проблемами? Нужно идентифицировать потребности и задачи, а затем расставить их по приоритетам.

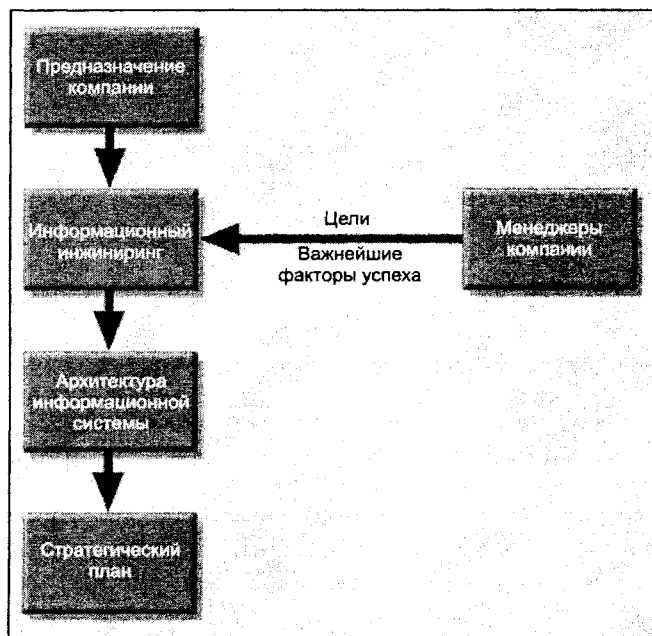


Рис. 16.8. Силы, влияющие на разработку архитектуры информационной системы (ISA)

- ❑ *Привлечение конечных пользователей.* Привлечение конечных пользователей — еще один важный аспект успеха стратегии администрирования данных. Каков уровень организационных изменений? Успешные организационные изменения предполагают, что люди смогут приспособиться к этим изменениям. Пользователю нужно предоставить открытый канал связи с руководством верхнего уровня для гарантии успешной реализации. Хорошие каналы взаимодействия являются ключевым моментом всего процесса.
- ❑ *Определение стандартов.* Аналитики и программисты должны быть знакомы с соответствующими методологиями, процедурами и стандартами. Если их познания в этой области недостаточны, то они могут пройти обучение по использованию этих процедур и стандартов.
- ❑ *Обучение.* Поставщик может обучить персонал DBA использованию СУБД и других средств. Конечных пользователей нужно обучить работе с этими инструментальными средствами, стандартами и процедурами для получения максимальных преимуществ при их использовании, тем самым повышая доверие пользователей к этим средствам. Прежде всего, необходимо обучить самых ответственных сотрудников, а после они смогут обучить остальных.

- *Небольшой экспериментальный проект.* Рекомендуется разработать небольшой проект, чтобы проверить работу СУБД на возможность ее использования в данной компании, а также, что выходные данные соответствуют требованиям, а персонал достаточно обучен.

Этот список факторов далеко не полон и не может быть полным. Тем не менее, он обеспечивает первоначальную схему для разработки успешной стратегии. Однако независимо от того, насколько полон список факторов успеха компании, он должен основываться на том положении, что разработка и реализация успешной стратегии администрирования данных тесно связаны с общими работами на предприятии по планированию информационной системы.

## 16.8. Работа DBA: использование СУБД Oracle для администрирования базы данных

До сих пор мы обсуждали рабочую среду DBA и его задачи в общем виде. Цель состояла в том, чтобы дать вам общее представление о широком поле организационной деятельности профессиональных администраторов БД. В этом разделе мы хотим, чтобы вы получили детальное представление о том, как DBA может решать следующие технические задачи в конкретной СУБД:

- создание и расширение структуры хранения БД;
- управление объектами базы данных, такими, как таблицы, индексы, триггеры, процедуры и т. д.;
- управление конфигурацией пользователя БД, включая тип и уровень доступа к БД;
- конфигурирование начальных параметров БД.

Многие из этих задач требуют от DBA применения инструментальных программных средств и утилит, обычно поставляемых с программным обеспечением БД. На самом деле, все поставщики СУБД предлагают набор программ для взаимодействия с базой данных и выполнения широкого круга административных задач.

Для иллюстрации выбранных задач DBA мы рассмотрим СУБД, которая обычно используется в достаточно больших организациях и имеет достаточно сложную инфраструктуру БД, требующую (и организация сможет себе это позволить) наличия DBA. Нам нужна СУБД с достаточным весом на рынке, чтобы ее можно было использовать в качестве основы для типовой инфраструктуры рабочей БД. Наконец, мы хотим использовать СУБД, доступную по цене даже для небольших колледжей и университетов. Поскольку Oracle отвечает всем этим требованиям, в качестве СУБД для наших примеров мы выбрали Oracle Workgroup 2000 для Windows NT<sup>3</sup>.

---

<sup>3</sup> Oracle предоставляет широкомасштабную политику изучения ее продуктов, что делает развитую структуру Oracle еще более доступной. Во время написания этой книги фирма Oracle предоставляла учебным институтам двухгодичную лицензию на использование своих инструментальных средств



### Примечание

Хотя Microsoft Access является превосходной СУБД, она, как правило, используется в небольших организациях или в организациях и подразделениях, оперирующих относительно небольшими объемами данных. MS Access предоставляет прекрасную среду для создания прототипа БД, а простой графический интерфейс делает быструю разработку интерфейсных приложений очень удобной. К тому же Access является одним из компонентов пакета программ MS Office, что обеспечивает сравнительно простую интеграцию приложений конечного пользователя. Наконец, Access предоставляет некоторые мощные инструментальные средства администрирования. Однако в структурах БД, основанных на MS Access, DBA используются сравнительно редко, поэтому MS Access не соответствует задачам этого раздела, что и определило выбор Oracle.

Имейте в виду, что с большей частью задач, описанных в этом разделе, DBA сталкивается независимо от используемой СУБД или операционной системы. Поэтому если вы используете DB2 Universal Database компании IBM или Microsoft SQL Server, то сможете адаптировать представленные здесь процедуры к вашей СУБД. Поскольку в этих примерах СУБД выполняется под управлением операционной системы Windows NT, некоторые из задач администрирования специфичны именно для этой системы. Следовательно, если вы используете другую операционную систему, вы должны адаптировать процедуры, представленные в этом разделе, к вашей операционной системе.

Наконец, необходимо уточнить, что данный раздел не может служить руководством по администрированию баз данных. Напротив, это всего лишь краткое описание способов решения некоторых типичных задач DBA, которые должны выполняться в Oracle. Перед тем как показать использование Oracle для решения этих специфических административных задач, в следующих двух разделах в общих чертах мы описали инструментальные средства администрирования Oracle и правила регистрации в системе.

## 16.8.1. Инструментальные средства администрирования Oracle

Все поставщики БД обеспечивают набор средств администрирования. Oracle обычно предоставляет главное меню, в котором отображаются основные средства администрирования базы данных. Это главное меню, представленное на рис. 16.9, называется *Oracle Administrator Toolbar* (панель инструментов администратора Oracle) или *Database Administrator Console* (консоль администратора базы данных).

На панели инструментов, представленной на рис. 16.9, показаны пять основных инструментов администрирования:

- ☐ *Security Manager (менеджер безопасности)* позволяет DBA создавать пользователей и присваивать им права доступа к базе данных;
- ☐ *Schema Manager (менеджер схем)* позволяет DBA управлять объектами базы данных, такими как таблицы, индексы, триггеры, процедуры и функции;
- ☐ *Storage Manger (менеджер хранения)* позволяет пользователю управлять базой данных и компонентами физического уровня, такими как область размещения таблиц, файлы данных и сегменты для отката (мы определим эти термины Oracle далее в этой главе);



Рис. 16.9. Панель инструментов администратора Oracle

- ☐ *SQL Worksheet* (*рабочая таблица SQL*) позволяет DBA вводить SQL-команды и видеть результат их действия;
- ☐ *Instance Manager* (*менеджер экземпляров БД*) позволяет DBA запускать и останавливать базу данных и конфигурировать начальные параметры БД.

## 16.8.2. Регистрация по умолчанию

Чтобы выполнить любую административную программу, необходимо подключиться к БД как пользователь с административными привилегиями (DBA) (процесс подключения называется регистрацией, *login*). На рис. 16.10 показано, что при регистрации используется имя SYSTEM, поскольку пользователь с таким именем имеет привилегии администратора. Выбрав любую опцию на панели инструментов (см. рис. 16.9), вы увидите окно регистрации (рис. 16.10).

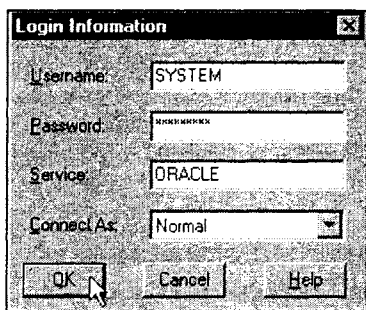


Рис. 16.10. Окно регистрации Oracle

В окне регистрации, представленном на рис. 16.10, DBA должен ввести следующую информацию:

- ☐ допустимое имя пользователя и пароль к базе данных, которую нужно открыть;
- ☐ имя базы данных, которую нужно открыть (вводится в поле **Service**);
- ☐ поле **Connect As** по умолчанию содержит значение **Normal**; мы рекомендуем не изменять его.

Одна из приятных возможностей панели инструментов администратора состоит в том, что она позволяет настраивать информацию по умолчанию. Такая настройка позволяет запускать административные программы без необходимости выполнения процедуры регистрации каждый раз при запуске административной программы. Регистрационная информация по умолчанию будет использоваться для каждого приложения.

При создании новой базы данных Oracle создает двух пользователей с именами SYSTEM и SYS. Оба пользователя имеют привилегии администратора (DBA). Имейте в виду, что имена пользователей и пароли зависят от базы данных. Поэтому в каждой базе данных могут быть различные имена пользователей и пароли. Первое, что необходимо сделать, — изменить пароли для двух пользователей по умолчанию (SYSTEM и SYS). Сразу после того как это сделано, можно создавать пользователей и назначать им привилегии доступа к БД.

### 16.8.3. Автоматический запуск РСУБД

Одна из основных задач DBA состоит в обеспечении автоматического запуска БД после запуска компьютера. Процедуры запуска различаются в разных операционных системах. Поскольку в примерах мы используем Oracle Workgroup 2000 для Windows NT Server, мы будем использовать оснастку *Services* (*Сервисы*), расположенную в панели управления Windows. На экране, появляющемся после ее запуска, показаны все сервисы, выполняющиеся на компьютере. (*Сервис* — это термин Windows NT для специальных программ, которые выполняются как часть операционной системы. Такая программа обеспечивает доступность соответствующего сервиса для системы и конечных пользователей локального компьютера или для локальной сети.) На рис. 16.11 показаны пять сервисов Oracle, автоматически запускаемых при запуске операционной системы Windows NT.

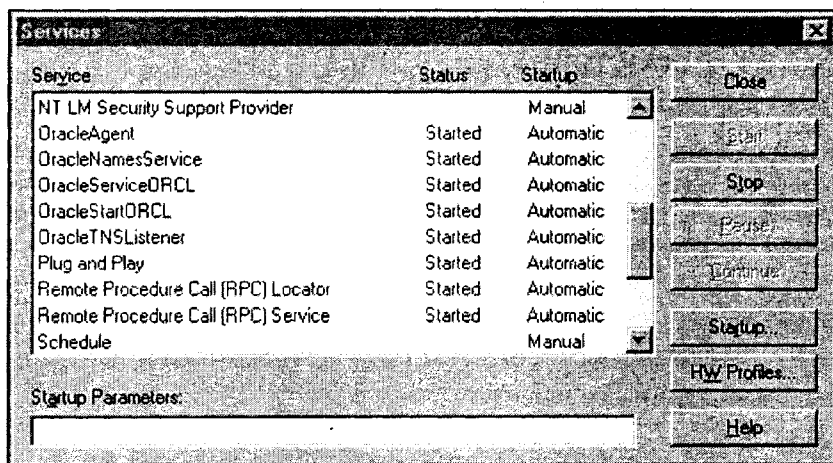


Рис. 16.11. Проверка запуска сервисов Oracle

На рис. 16.11 обратите внимание на следующие сервисы Oracle.

- ❑ Сервис *OracleNamesService* обеспечивает *разрешение имен* (процесс преобразования имен в конкретный сетевой адрес) для базы данных. Каждая база данных Oracle должна иметь уникальное имя, которое идентифицирует его в локальной сети. Такая идентификация очень важна при одновременной работе нескольких баз данных в распределенной среде (имена, используемые для идентификации

базы данных в сети, должен создавать DBA). Сервис `OracleNameService` позволяет каждому пользователю ссылаться на каждую базу данных по имени, без необходимости определять местонахождение БД. Говоря кратко, этот сервис обеспечивает независимость местоположения БД (если нужно, вспомните материал *гл. 10*, посвященной независимости местоположения).

- ❑ Сервис *OracleTNSListener* это процесс, который "слушает" (listens) и обрабатывает запросы на подключение конечных пользователей к БД по локальной сети. Например, когда SQL-запрос на подключение вида "connect userid/password @ORALAB" передается по сети, сервис *OracleTNSListener* получает запрос, проверяет его и устанавливает подключение.
- ❑ Сервисы *OracleServiceORCL* и *OracleStartORCL* — это часть базы данных, создаваемой по умолчанию при установке Oracle. Суффикс "ORCL" используется для экземпляра базы данных по умолчанию. *Экземпляр базы данных (database instance)* связан с процессами Oracle в памяти, относящимися к конкретной базе данных. Можно представлять себе экземпляр базы данных как отдельный участок памяти, зарезервированный для запуска базы данных. Поскольку у вас может иметься несколько баз данных (и, следовательно, несколько экземпляров), выполняющихся в памяти одновременно, необходимо уникально идентифицировать каждый экземпляр БД с помощью различных суффиксов. Сервис *OracleStartORCL* автоматически запускает экземпляр БД во время загрузки операционной системы.

#### 16.8.4. Использование менеджера хранения для создания пространства таблиц и файлов данных

В *гл. 1* база данных была определена как интегрированная компьютерная структура, в которой размещаются связанные данные — т. е. данные конечных пользователей и метаданные. Ясно, что эти данные должны храниться на диске в виде файлов. Это важное замечание вызывает два вопроса.

- ❑ Хранится ли база данных в виде файла?
- ❑ Чем такое хранение файла отличается от системы файлов, исследованной в *гл. 2*?

Очень важно знать ответы на эти вопросы, если вы хотите понять некоторые важнейшие проблемы управления базой данных.

Ответ на первый вопрос — да. На физическом уровне база данных хранится в одном или более файлах. Однако ответ на второй вопрос требует провести различие между файлом, используемым базой данных для хранения информации, и файлом, описанным в *гл. 1*.

- ❑ Файл системы файлов создается для хранения информации конечного пользователя о единственной сущности, и программист может получить непосредственный доступ к такому файлу. Но доступ к файлу требует от конечного пользователя знания структуры данных, которые хранятся в файле.
- ❑ В СУБД файлы, в которых хранится база данных, создаются СУБД, а не конечным пользователем. Поскольку все операции с данными в файле выполняются

СУБД, конечному пользователю нет необходимости знать структуру файла базы данных.

- ❑ На логическом уровне СУБД представляет базу данных для конечного пользователя как единую логическую структуру. Пользователю нет дела до физических подробностей хранения данных в файлах.

Каждая СУБД управляет хранением данных по-разному. В данном примере мы будем использовать РСУБД Oracle, чтобы проиллюстрировать, как БД управляет хранением данных на логическом и физическом уровне. В Oracle:

- ❑ база данных *логически* состоит из одного или более пространств таблиц. *Пространство таблиц (tablespace)* это логическое пространство хранения. Пространства таблиц, прежде всего, предназначены для логического группирования связанных данных.
- ❑ Данные пространства таблиц *физически* хранятся в одном или более файлах данных (datafile). Файлы данных хранят данные БД физически. Каждый файл данных связан с одним или более пространством таблиц. Но каждый *файл данных (datafile)* может размещаться в другом каталоге жесткого диска или даже на другом жестком диске.

С учетом этого описания пространства таблиц и файлов данных мы можем сделать вывод, что база данных имеет связь 1:М с пространством таблиц, а пространство таблиц имеет связь 1:М с файлами данных. Это множество иерархических связей 1:М изолирует конечного пользователя от всех физических деталей хранения. Однако *DBA должен быть знаком с этими деталями для того, чтобы правильно управлять базой данных.*

Для решения задач управления хранением в БД (например, создание и управление пространствами таблиц и файлами данных) DBA использует административную программу Oracle Storage Manager (менеджер хранения), представленную на рис. 16.12.

Когда DBA создает базу данных, Oracle автоматически создает пространства таблиц и файлы данных, как показано на рис. 16.12.

- ❑ Пространство таблиц SYSTEM предназначено для хранения словаря данных. Пространство таблиц связано с файлом данных SYS1ORCL.ORA.
- ❑ Пространство таблиц USER\_DATA предназначено для хранения таблицы и индексов данных, созданных конечным пользователем. Это пространство таблиц связано с файлом данных USR1ORCL.ORA.
- ❑ Пространство таблиц TEMPORARY\_DATA предназначено для хранения временных таблиц и индексов, созданных во время выполнения SQL-операторов. Например, временные таблицы создаются, если оператор SQL содержит выражения ORDER BY, GROUP BY и HAVING. Это пространство таблиц связано с файлом данных TMP1ORCL.ORA.
- ❑ Пространство таблиц ROLLBACK\_DATA предназначено для хранения информации по восстановлению транзакций базы данных. Если по какой-то причине транзакция необходимо отменить (в целях сохранения целостности), сегмент отката (rollback) используется для хранения информации об отмене. Это пространство таблиц связано с файлом данных RBS1ORCL.ORA.

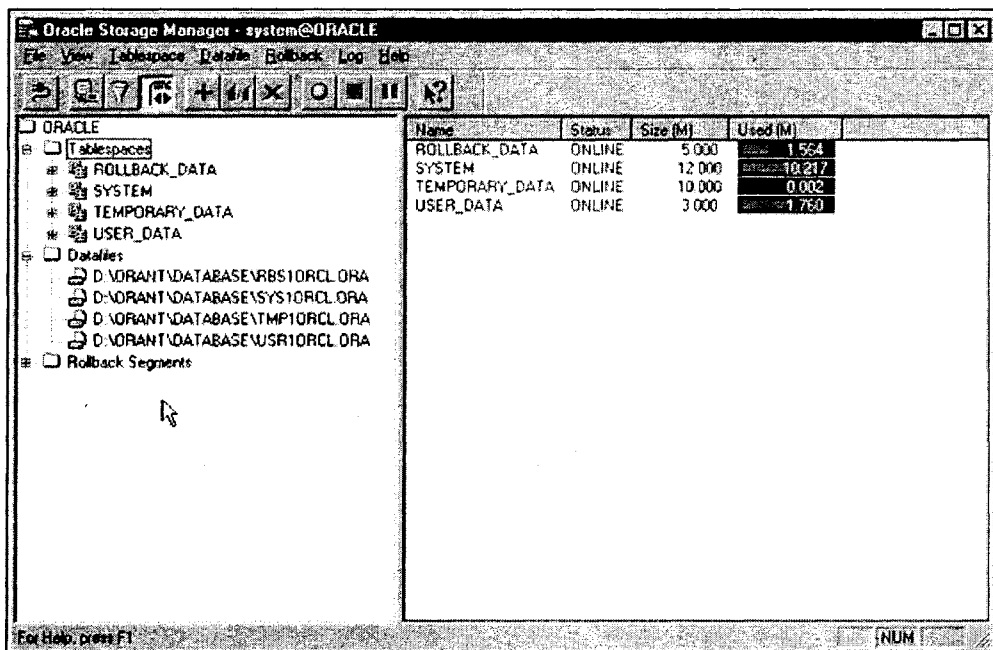


Рис. 16.12. Окно программы Oracle Storage Manager

С помощью Storage Manager DBA может решать следующие задачи.

- ❑ Создать дополнительное пространство таблиц для организации данных в БД. Следовательно, если у базы данных сотни пользователей, можно создать несколько пользовательских пространств таблиц для сегментов архива данных для различных типов пользователей. Например, можно создать пространство таблиц для преподавателей и для студентов.
- ❑ Создать дополнительное пространство таблиц для организации различных подсистем, которые могут иметь место в БД. Например, можно создать различные пространства таблиц для данных отдела кадров, информации по платежным ведомостям, информации расчетного отдела, данных производственного отдела и т. д. На рис. 6.13 показано диалоговое окно, которое было использовано при создании нового пространства таблиц с названием ROBCOR\_DATA для хранения таблиц к этой книге. Это пространство таблиц хранится в файле D:\ORANT\DATABASE\RCDATA.ORA, начальный размер которого равен 10 Мбайт.

На рис. 16.13 обратите внимание, что пространство таблиц будет немедленно доступно пользователям для размещения данных. Обратите также внимание, что в нижней части диалогового окна можно посмотреть SQL-код, который генерирует Oracle при создании пространства таблиц. (Фактически все задачи DBA можно выполнить с помощью SQL-команд. На самом деле, некоторые твердолобые DBA предпочитают писать собственный SQL-код, а не использовать "простой выход из положения" с помощью GUI).

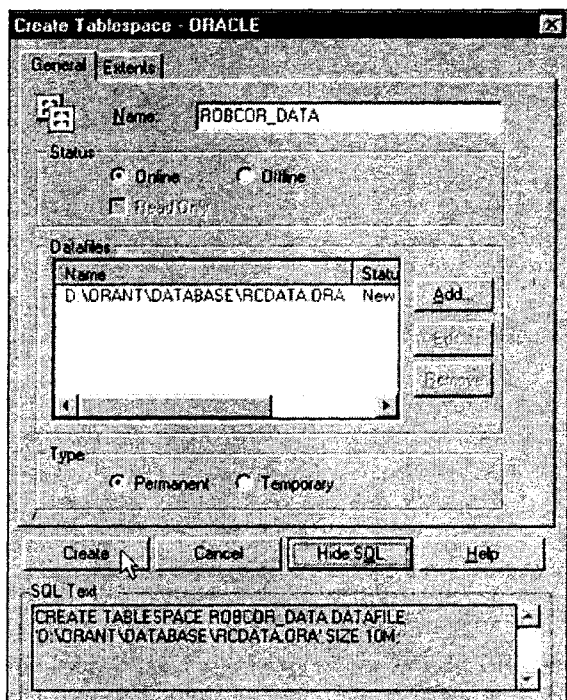


Рис. 16.13. Создание нового пространства таблиц

- Увеличивать емкость хранения пространства таблиц, создавая дополнительные файлы данных. Если использовать полосу прокрутки, показанную в нижней части поля **Datafiles** на рис. 16.13, то вы увидите размер памяти для каждого пространства таблиц в правой части этого поля. Помните, что файлы данных могут храниться в одном каталоге или в различных каталогах жестких дисков в целях повышения производительности. Например, можно увеличить эффективность хранения и доступа к таблице `USER_DATA`, создав новый файл данных на другом устройстве (например, `E:\ORANT\DATABASE\USR2ORCL.ORA`).

### 16.8.5. Управление объектами базы данных: таблицы, представления, триггеры и процедуры

Другой важный аспект управления базой данных — наблюдение за созданными объектами БД. Менеджер схем Oracle Schema Manager предоставляет DBA графический интерфейс для создания, редактирования, просмотра и удаления объектов БД. *Объект базы данных (database object)* это любой объект, созданный конечным пользователем, например, таблица, представление, индекс, хранимая процедура или триггер. На рис. 16.14 показаны различные типы объектов, перечисленные в Oracle Schema Manager.

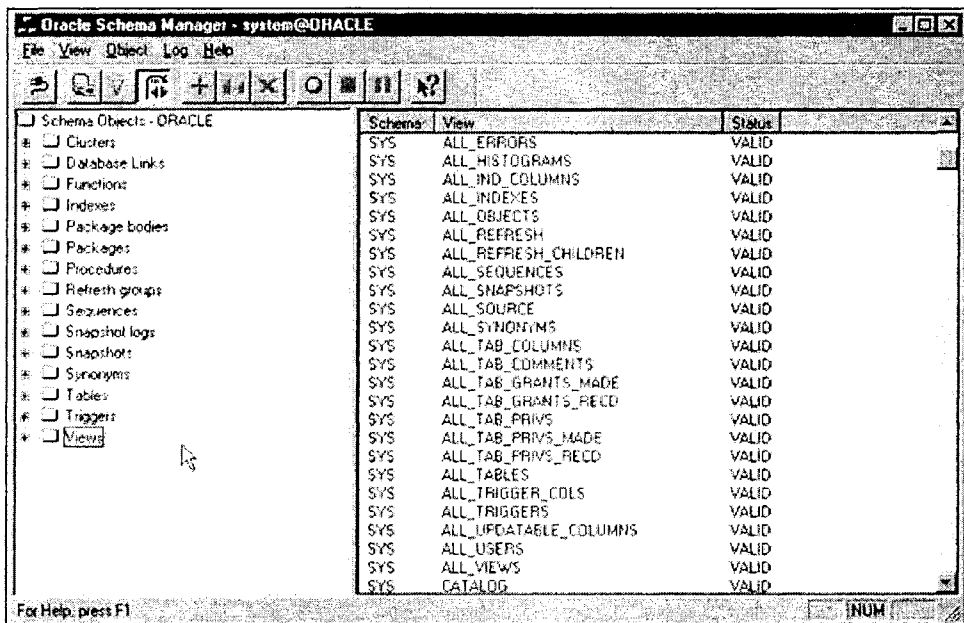


Рис. 16.14. Окно программы Oracle Schema Manager

Схема Oracle это логический раздел базы данных, который принадлежит данному пользователю, и такая схема определяется именем пользователя. Например, если пользователь с именем SYSTEM создает таблицу VENDOR, то эта таблица будет принадлежать схеме SYSTEM. В Oracle имя таблицы получает префикс, совпадающий с именем пользователя. Поэтому таблица VENDOR схемы SYSTEM будет в Oracle называться SYSTEM.VENDOR. Точно так же если пользователь PEROB создает таблицу VENDOR, то эта таблица будет создана в схеме PEROB и получит имя PEROB.VENDOR.

Внутри схемы пользователь авторизуется только по доступу к объектам, принадлежащим к своей схеме. Пользователь может, конечно, получить доступ и к другим объектам, если его полномочия будут установлены надлежащим образом. На самом деле, все пользователи с полномочиями DBA имеют доступ ко всем объектам всех схем базы данных.

Как видно на рис. 16.14, Schema Manager (менеджер схем) дает упорядоченное представление всех объектов схемы базы данных. С помощью этой программы DBA может создавать, редактировать, просматривать и удалять таблицы, индексы, представления, функции, триггеры, процедуры и другие специальные объекты.

На рис. 16.14 также представлены некоторые таблицы БД, формирующие словарь данных Oracle. Например, DBA может запросить таблицу ALL\_VIEWS, чтобы получить список всех заданных представлений, или таблицу ALL\_USERS, чтобы получить список всех пользователей БД (полное описание всех таблиц словаря данных можно найти в "Справочном руководстве Администратора базы данных" — Database Administrator's Guide).



## 16.8.6. Управление пользователями и безопасность

Одно из наиболее распространенных действий администратора БД — создание и управление пользователями базы данных. Для выполнения этой задачи все СУБД предлагают программы администрирования, позволяющие создавать пользователей базы данных, управлять ими или удалять их. (Фактически создание идентификатора пользователя — это первый компонент хорошо спланированных работ по обеспечению безопасности. Как уже было сказано в этой главе, безопасность БД — важнейшая задача администрирования базы данных.)

Менеджер безопасности Oracle (Security Manager) — это административная утилита, позволяющая DBA создавать пользователей, их роли и профили.

- ❑ *Пользователь (user)* — уникально идентифицируемый объект, который позволяет данному лицу регистрироваться в базе данных. DBA назначает привилегии доступа объекта к базе данных. В рамках назначения привилегий DBA может определить набор ограничений на ресурсы БД, которые может использовать данное лицо.
- ❑ *Роль (role)* — именованный набор привилегий доступа к базе данных, который авторизует пользователя при его подключении к БД и при использовании ресурсов БД. Примеры ролей:
  - CONNECT — разрешает пользователю подключаться к базе данных, а также создавать и модифицировать таблицы, представления и другие объекты данных;
  - RESOURCE — разрешает пользователю создавать триггеры и процедуры и другие объекты управления;
  - DBA — предоставляет пользователю БД полномочия администратора.
- ❑ *Профиль (profile)* — именованный набор установок, который управляет количеством ресурсов БД, доступных данному пользователю (если вы представите себе случай, когда неконтролируемый запрос может заблокировать БД и она перестанет отвечать на запросы пользователя, вы поймете, почему так важно ограничивать доступ к ресурсам БД!). Определяя профили, DBA может ограничить размер пространства хранения, доступный пользователю, длительность подключения пользователя к БД, время бездействия, после которого пользователь отключается от БД и т. д. В идеале все пользователи должны иметь неограниченный доступ ко всем ресурсам в любое время, но в действительности мы понимаем, что такой доступ невозможен, да и не нужен.

На рис. 16.15 показано основное окно Oracle Security Manager. Обратите внимание, что в его левой части отображается графическое представление основных типов объектов — пользователей, ролей и профилей — а в правой части будут показаны реальные объекты, принадлежащие каждому типу, после того как будет выбран элемент в левой части.

Создать нового пользователя DBA можно с помощью диалогового окна **Create User**, представленного на рис. 16.16.

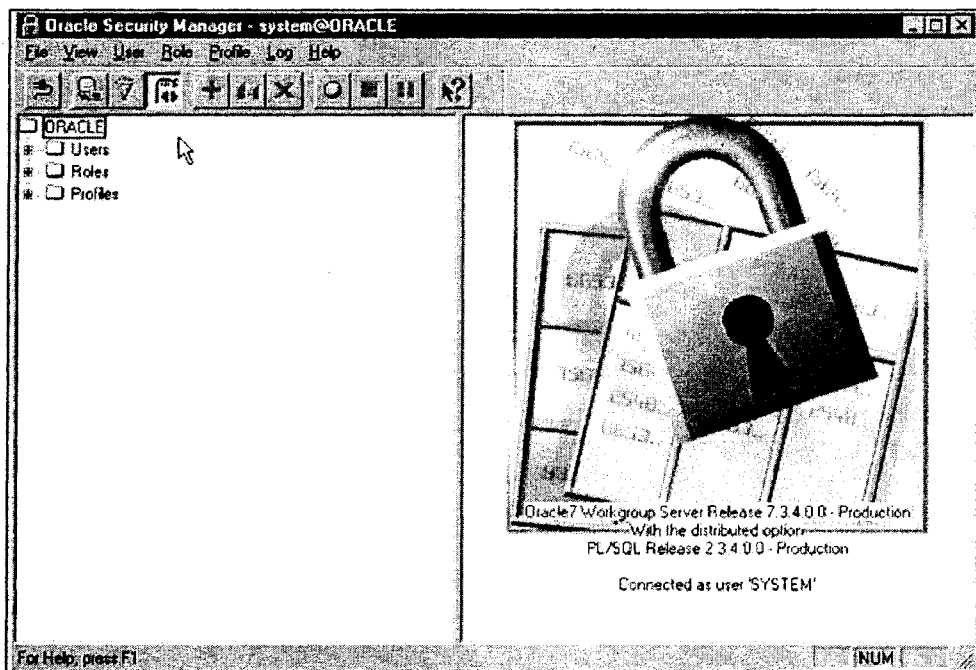


Рис. 16.15. Окно программы Oracle Security Manager

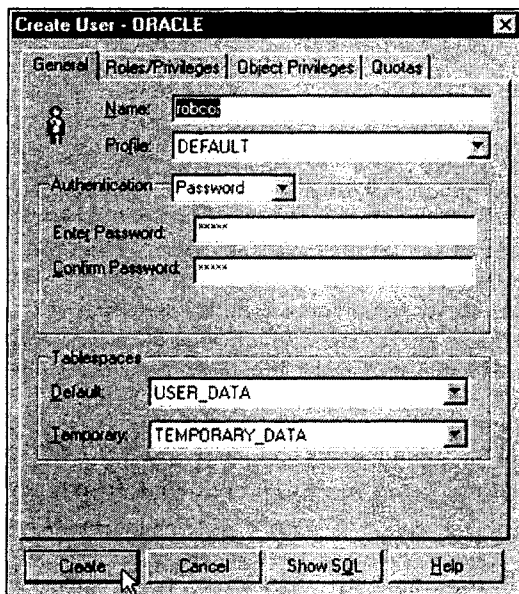


Рис. 16.16. Диалоговое окно Create User

Диалоговое окно **Create User** содержит четыре вкладки.

- ☐ На вкладке **General DBA** можно задать имя, пароль и профиль нового пользователя. Также в этом окне DBA может определить пространство таблиц по умолчанию, используемое для хранения табличных данных, а также временное пространство таблиц для временных данных.
- ☐ На вкладке **Roles/Privileges** DBA может назначить роли пользователям.
- ☐ На вкладке **Object Privileges** DBA может назначить специфические права доступа к другим объектам БД.
- ☐ На вкладке **Quotas** DBA может определить максимальный размер памяти, доступной пользователю для каждого назначенного им пространства таблиц.

### 16.8.7. Настройка начальных параметров базы данных

Хорошая настройка базы данных — еще одна важная задача DBA. Эта задача обычно требует модификации некоторых конфигурационных параметров БД, часть из которых можно изменять во время работы системы с помощью SQL-команд, в то время как для изменения других потребуется остановить БД и запустить ее снова. Поэтому очень важно, чтобы DBA был знаком с параметрами конфигурации БД, особенно с теми, которые влияют на производительность БД.

Каждая база данных Oracle имеет связанный с ней файл инициализации, где хранятся параметры конфигурации времени выполнения. Этот файл инициализации представляет собой обычный текстовый файл ASCII и обычно называется `INITXXXX.ORA`, где `XXXX` — четырехсимвольный идентификатор экземпляра. Файл инициализации считывается при запуске экземпляра и устанавливает рабочую среду базы данных. Oracle Instance Manager (менеджер экземпляров Oracle) позволяет DBA запускать, останавливать<sup>1</sup> и просматривать/редактировать параметры конфигурации (хранящиеся в файле инициализации) экземпляра БД. Программа Oracle Instance Manager предоставляет GUI для модификации текстового файла, содержащего параметры конфигурации, как показано на рис. 16.17.

Одна из важнейших функций, предоставляемых параметрами инициализации, показанными в правой части окна на рис. 16.17, — резервирование ресурсов, используемых базой данных во время выполнения. Один из этих ресурсов — первичная память, необходимая для кэширования транзакций базы данных, которое требуется для настройки производительности базы данных. Например, параметр `shared_pool_size` устанавливает предельный объем памяти, отведенной под кэширование транзакций базы данных. Этот параметр нужно установить в значение, достаточное для поддержки всех параллельных транзакций.

После настройки параметров инициализации может потребоваться перезапуск базы данных. Это можно выполнить с помощью опции **Database** панели инструментов в окне менеджера экземпляров (см. рис. 16.17).

Из приведенного краткого описания можно понять, что DBA отвечает за задачи весьма широкого спектра. Высококачественные и совершенные инструментальные средства администрирования, доступные DBA, в значительной степени облегчают его работу. Но даже при этих условиях очень важно, чтобы DBA был знаком с инст-

рументальными средствами и техническими деталями РСУБД, чтобы надлежащим образом выполнять свою работу.

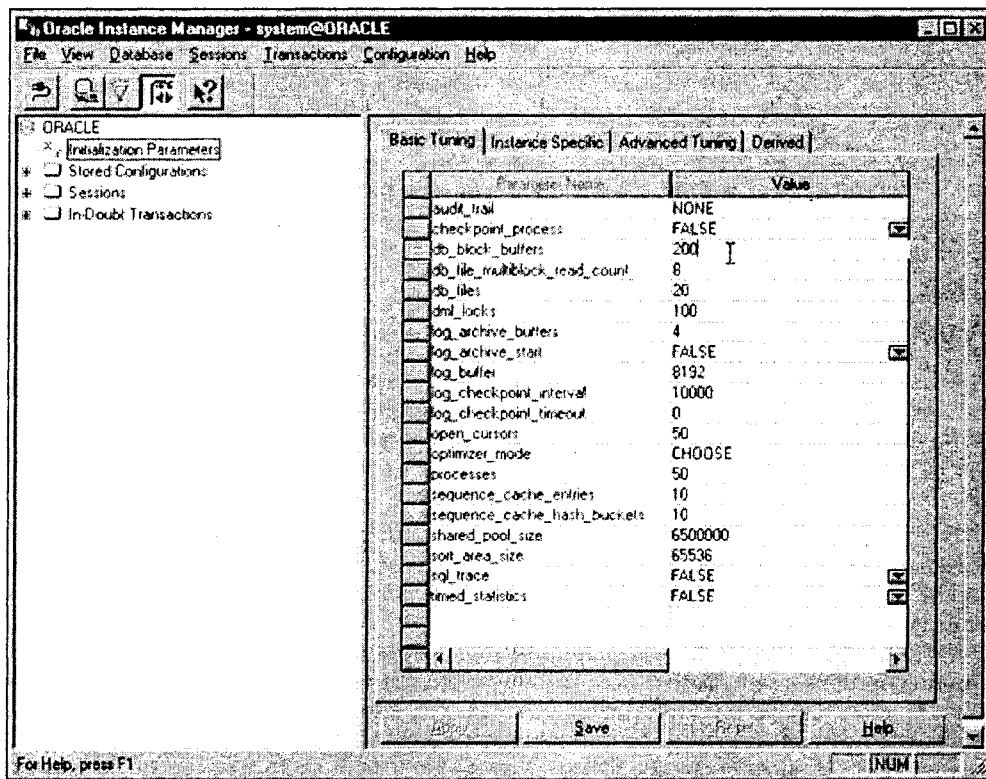


Рис. 16.17. Окно программы Oracle Instance Manager

### 16.8.8. Создание новой базы данных

Хотя общий порядок создания новой базы данных практически универсален, его выполнение зависит от специфики СУБД. Ведущие поставщики РСУБД предлагают DBA возможность создания новой БД вручную, с помощью SQL-команд или с помощью графического интерфейса. Выбор зависит от предпочтений DBA и стиля его работы.

В Oracle создание базы данных с помощью SQL-запросов потребует выполнения следующих действий.

1. Создать новый файл инициализации Oracle. Этот файл автоматически считывается при запуске базы данных. В нем содержатся важные параметры настройки и загрузки БД в память.

Вот некоторые из этих параметров:

- **DB\_NAME** — имя новой базы данных; должно быть уникальным и включать до восьми символов. Например:

**DB\_NAME=ROBCOR**

- **CONTROL\_FILES** (управляющие файлы). Управляющие файлы содержат важную информацию о базе данных. Можно определить как имя управляющего файла, так и его местоположение. Рекомендуются, по крайней мере, два управляющих файла. Например, обратите внимание на управляющие файлы с именами **CTL1ROBC.ORA** и **CTL2ROBC.ORA**, расположенные на двух разных устройствах в отдельных каталогах:

```
CONTROL_FILES=(D:\ORACLE\ROBCOR\CTL1ROBC.ORA,
 E:\ORACLE\ROBCOR\CTL2ROBC.ORA)
```

- **DB\_FILES** — максимальное число файлов данных в базе данных. Например:

**DB\_FILES=15**

При именовании файла инициализации необходимо придерживаться следующего синтаксиса: **init????.ora**, где **????** — четыре символа, уникально идентифицирующие вашу базу данных. Oracle рекомендует скопировать файл инициализации по умолчанию (**initordcl.ora**) и модифицировать его в соответствии с требованиями новой базы данных.

2. Создать и запустить новый сервис Oracle для управления новой базой данных, которую мы собираемся создать. Этот шаг зависит от операционной системы и здесь используется файл инициализации, сформированный на шаге 1. Например:

```
ORADIM73 -NEW -SID ROBCOR -PFILE -D:\ORACLE\ROBCOR\INITROBC.ORA
```

Эта команда запустит новый экземпляр Oracle для управления новой базой данных.

3. Выполнить SQL-оператор **CREATE DATABASE** с помощью программы SQL Plus. Прежде чем выполнить эту SQL-команду, DBA должен зарегистрироваться в базе данных Oracle с административными полномочиями по умолчанию. Оператор **CREATE DATABASE** должен соответствовать значениям, указанным в файле инициализации, например:

```
CREATE DATABASE ROBCOR
```

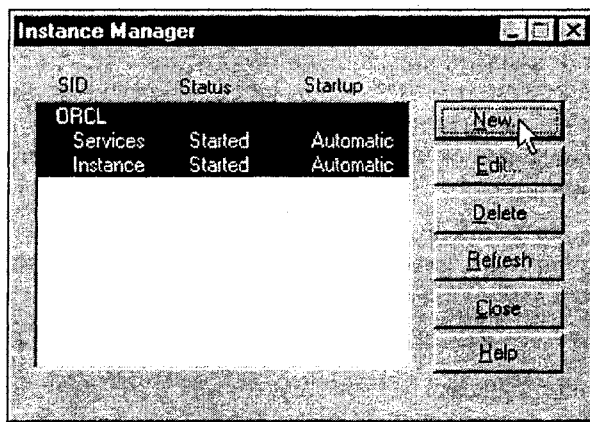
```
LOGFILE 'D:\ORACLE\ROBCOR\LOG1ROBC.ORA' SIZE 1M,
 'D:\ORACLE\ROBCOR\LOG2ROBC.ORA' SIZE 1M
 'D:\ORACLE\ROBCOR\LOG3ROBC.ORA' SIZE 1M
 'D:\ORACLE\ROBCOR\LOG4ROBC.ORA' SIZE 1M
MAXDATAFILES 15
```

```
DATAFILE 'D:\ORACLE\ROBCOR\SYS1ROBC.ORA' SIZE 20M
```

На этом шаге физически создаются файлы базы данных, в которых хранится структура БД. В файле данных `syslogbcs.ora` будут содержаться таблицы словаря данных, создаваемые на шаге 5.

4. Выполнить дополнительные SQL-команды для создания дополнительных пространств таблиц и сегментов отката для новой базы данных.
5. Выполнить SQL-сценарии для создания таблиц словаря данных и других вспомогательных таблиц.
6. Использовать SQL-команду `ALTER USER` для изменения пароля для пользователей по умолчанию (`SYS` и `SYSTEM`).

Другой способ создания новой базы данных в Oracle состоит в использовании графического интерфейса Oracle NT Instance Manager. Эта программа автоматизирует большую часть задач, выполняемых при создании новой БД. Начните с нажатия кнопки **New** в окне Instance Manager (рис. 16.18).



**Рис. 16.18.** Создание новой базы данных с помощью Oracle NT Instance Manager

Откроется окно **New Instance** (Новый экземпляр) (рис. 16.19), где вы можете ввести идентификатор `SID`. Идентификатор `SID` — это уникальное имя экземпляра базы данных длиной 4 символа. Идентификатор `SID` должен соответствовать `SID` в файле инициализации. Затем введите пароль, который `DBA` будет использовать в новой базе данных. Нажмите кнопку **Advanced** для отображения окна **Advanced Parameters** (Дополнительные параметры) (рис. 16.20).

Один из недостатков использования графического интерфейса при создании БД — отсутствие каких-либо записей в форме SQL-сценариев, документирующих создание БД. Как только БД создана, `DBA` должен выполнить шаги 4, 5 и 6 предыдущего алгоритма. Иначе говоря, даже с помощью GUI процесс создания базы данных требует полного понимания основной структуры БД и ее компонентов.

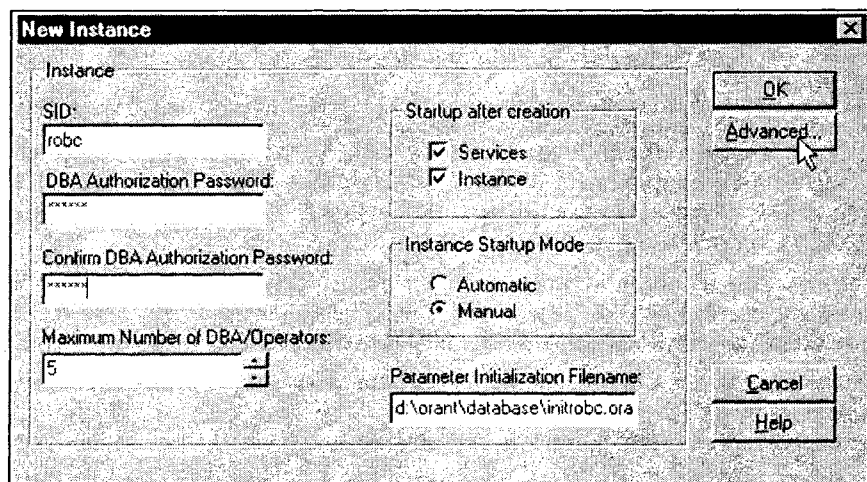


Рис. 16.19. Окно New Instance

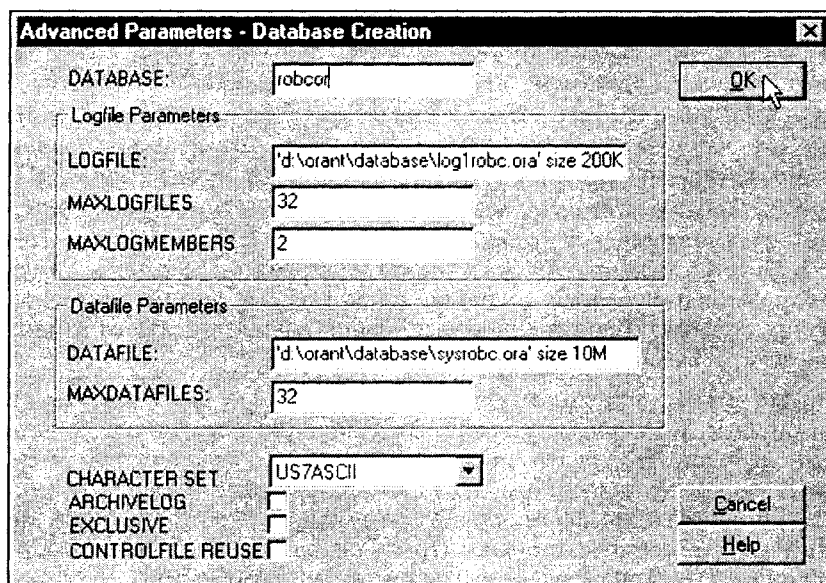


Рис. 16.20. Окно Advanced Parameters

## Резюме

Управление данными — важный аспект деятельности любого предприятия. Данные должны рассматриваться как корпоративное достояние, имеющее свою стоимость. Значение набора данных оценивается полезной информацией, которую можно по-

лучить на его основе. Хорошее управление данными, скорее всего, позволит должным образом структурировать информацию, что является основой для принятия верных решений.

СУБД — наиболее широко используемое электронное средство корпоративного управления данными. Роль СУБД состоит в поддержке организационных решений на всех уровнях предприятия. Поэтому СУБД должна обеспечивать поддержку для стратегических, тактических и оперативных решений. Данные компании, управляемые СУБД, хранятся в корпоративной БД или БД предприятия.

Внедрение СУБД на предприятии — очень деликатная работа. Кроме организационных и технических проблем внедрения СУБД необходимо тщательно исследовать влияние СУБД на структуру управления и среду общения предприятия.

В ответ на изменения в организации данных были разработаны функции администрирования данных, которые стимулировались техническим совершенствованием компьютерных систем. Функции администрирования данных развивались на основе эволюции от обработки данных в подразделении к подразделению централизованной электронной обработки данных (EDP, Electronic Data Processing), к более формальному определению данных как "корпоративного достояния" отдела информационных систем (IS). Обычные системы файлов работали с помощью приложений, что приводило к появлению "островков информации". По мере того как приложения получали совместный доступ к архивам данных, становилась очевидной потребность в централизованном управлении информацией, что привело к разработке функций администрирования.

Администратор базы данных (DBA) отвечает за управление корпоративной базой данных. Внутренняя организация функций администрирования БД варьируется от компании к компании. Хотя определенных стандартов не существует, общепринятой практикой считается деление функций DBA в соответствии с этапами жизненного цикла БД. Некоторые компании считают, что необходимо обеспечить управление не только компьютеризованными данными, но и любыми видами информационных ресурсов предприятия. Такими широкими полномочиями наделяется администратор данных (data administrator, DA).

Функции DA и DBA перекрываются. Вообще говоря, DA больше ориентирован на организационные функции, в то время как DBA — на технические функции. Если сравнивать функции DBA и DA, то можно сказать, что функции DA не зависят от СУБД и имеют более широкий и продолжительный спектр деятельности. Однако если в схему организации должность DA не включена, то его функции выполняет DBA. Поскольку в круг обязанностей DBA входят и технические, и организационные функции, он должен обладать весьма широкими познаниями.

Организационная роль DBA предполагает его ответственность за планирование, организацию, тестирование, наблюдение и доступ к сервисам, относящимся к функциональным обязанностям DBA. Организационные функции DBA включают в себя, по крайней мере, следующие обязанности:

- ☐ поддержка конечных пользователей;
- ☐ определение и реализация политик, процедур и стандартов для функционирования БД;



- ☐ обеспечение безопасности, конфиденциальности и целостности данных;
- ☐ обеспечение резервного копирования и восстановления;
- ☐ наблюдение за распределением и использованием данных БД.

Технические функции DBA тесно связаны с его организационной ролью. Технические функции DBA требуют, по крайней мере, выполнения следующих действий:

- ☐ оценка, выбор и установка СУБД и утилит;
- ☐ проектирование и реализация баз данных и приложений;
- ☐ тестирование и оценка баз данных и приложений;
- ☐ функционирование СУБД, утилит и приложений;
- ☐ обучение и поддержка пользователей;
- ☐ обслуживание СУБД, утилит и приложений.

Разработка стратегии администрирования данных тесно связана с направлением деятельности компании и ее целями. Поэтому разработка стратегического плана должна соответствовать администрированию данных и требует детального анализа целей, положения дел и коммерческих потребностей компании. Чтобы руководить разработкой такого всеобъемлющего плана, необходима интегрирующая методология. Наиболее общепринятой интегрирующей методологией считается информационный инжиниринг (инфотехника).

Информационный инжиниринг позволяет транслировать стратегические планы в набор данных и приложений, отражающих основные положения планов. Реализация общей стратегии предприятия — очень важный процесс, на который оказывают влияние организационные, технологические и интеллектуальные проблемы.

Чтобы транслировать стратегические планы в оперативные, DBA располагает арсеналом инструментальных средств администрирования. Эти средства включают в себя словарь данных и CASE-инструментарий.

В этой главе представлена краткая иллюстрация выполнения некоторых функций администрирования данных с помощью средств Oracle Workgroup Server for NT. Здесь охватывается несколько задач, выполняемых типичным DBA в среде Oracle с помощью Oracle Database Administrator Tools. Так как каждый поставщик СУБД предоставляет собственный набор средств выполнения административных задач, необходимо свериться с "Руководством по администрированию" вашей СУБД о том, как выполнять специальные задачи администрирования БД.

## Основные термины

CASE-инструментарий разработки интерфейса — front-end CASE tool

Автоматизированные средства разработки программ — computer-aided software engineering (CASE)

Авторизованное управление — authorization management

Администратор БД — database administrator (DBA)

- Администратор данных — data administrator (DA)
- Активный словарь данных — active data dictionary
- Архитектура информационных систем — information system architecture
- База данных предприятия — enterprise database
- Безопасность — security
- Дамп базы данных — database dump
- Инкрементальное резервное копирование — incremental backup
- Инспектор безопасности базы данных — database security officer (SO)
- Информационный инжиниринг — information engineering (IE)
- Контрольный журнал — audit log
- Конфиденциальность — privacy
- Менеджер информационных ресурсов — information resource manager
- Объект базы данных (Oracle) — database object
- Отдел информационных систем — information system department (IS)
- Отдел обработки данных — data processing department (DP)
- Отдел электронной обработки данных — electronic data processing (EDP) department
- Параллельное резервное копирование — concurrent backup
- Пассивный словарь данных — passive data dictionary
- План доступа — access plan
- Политики — policies
- Полное резервное копирование — full backup
- Пользователь (Oracle) — user
- Пространство таблиц (Oracle) — tablespace
- Профиль (Oracle) — profile
- Процедуры — procedures
- Роль (Oracle) — role
- Серверный CASE-инструментарий — back-end CASE tool
- Системный администратор — system administrator (SYSADM)
- Словарь информационных ресурсов — information resource dictionary
- Стандарт — standard
- Управление чрезвычайными ситуациями — disaster management
- Файл данных (Oracle) — datafile
- Функции администрирования БД — database administration function
- Экземпляр базы данных (Oracle) — database instance

## Вопросы

1. Приведите примеры необработанной и структурированной информации.
2. Поясните взаимосвязь между конечными пользователями, информацией, структурированной информацией и принятием решений. Начертите схему и поясните взаимодействие ее элементов.
3. Предположим, что вы штатный сотрудник отдела DBA. Какие данные вы должны получить на уровне высшего руководства для обеспечения их функциями администрирования данных?
4. Как и почему системы управления базами данных стали стандартными средствами управления данными на предприятии? Обсудите некоторые преимущества баз данных по сравнению с системами файлов.
5. Одной фразой опишите роль баз данных на предприятии. Поясните свой ответ.
6. Дайте определение безопасности и конфиденциальности. Как эти две концепции связаны друг с другом?
7. Опишите и сравните информационные потребности на стратегическом, тактическом и оперативном уровнях предприятия. Приведите примеры, поясняющие ваш ответ.
8. Что нужно принимать во внимание при обдумывании внедрения СУБД на предприятии?
9. Кто является администратором БД? Опишите основные обязанности DBA.
10. Какое место функциональные обязанности DBA занимают в схеме предприятия? Как оно влияет на функции DBA?
11. Как и почему новые технологические разработки в компьютерах и базах данных меняют роль DBA?
12. Объясните внутреннюю структуру отдела DBA на основе подхода DBLC.
13. Поясните и сравните должности DA и DBA.
14. Поясните, в чем состоит роль DBA как арбитра между двумя важнейшими ценностями предприятия. Начертите схему, поясняющую ваши рассуждения.
15. Опишите и охарактеризуйте навыки, необходимые DBA.
16. В чем состоят организационные функции DBA? Опишите организационные действия и сервисы, обеспечиваемые DBA.
17. Какие действия выполняет DBA для поддержки конечных пользователей?
18. Поясните организационную роль DBA в определении и проведении в жизнь политик, процедур и стандартов.
19. Защита данных, конфиденциальность и целостность — важные функции базы данных в управлении авторизацией. Какие действия требуются от DBA для проведения этих функций в жизнь?
20. Обсудите необходимость и свойства процедур резервного копирования и восстановления БД. Затем по возможности подробно опишите действия по планированию резервного копирования и восстановления.

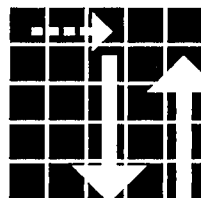
21. Предположим, ваша компания назначила вас ответственным за выбор корпоративной СУБД. Разработайте контрольный список технических и других аспектов, влияющих на этот выбор.
22. Опишите действия, которые обычно связаны с проектированием и реализацией сервисов, относящихся к техническим функциональным обязанностям DBA. Какие технические навыки требуются от персонала DBA?
23. Почему тестирование и оценку базы данных и приложений не могут выполнять те же, кто отвечает за их проектирование и разработку? Каких минимальных стандартов необходимо придерживаться во время тестирования и оценки?
24. Выявите узкие места производительности СУБД. Предложите решения по повышению эффективности СУБД.
25. Каковы типичные действия, необходимые при обслуживании СУБД, утилит и приложений? Можно ли рассматривать настройку производительности приложения как действия по обслуживанию СУБД? Поясните ваш ответ.
26. Определите концепцию словаря данных. Затем обсудите различные типы словарей данных. Если вы управляете всем набором данных предприятия, какими свойствами должен обладать словарь данных?
27. Используя SQL-операторы, приведите некоторые примеры применения словаря данных для наблюдения за безопасностью БД.

### Примечание

Если вы используете IBM OS/2 EE DBM v1.3, имена главных таблиц выглядят так: SYSTABLES, SYSCOLUMNS и SYSTABAUTH.

28. Какими общими свойствами обладают CASE-инструментарий и СУБД? Как эти свойства используются для расширения функций администрирования данных?
29. Кратко опишите концепции информационного инжиниринга (IE) и архитектуры информационных систем (ISA). Какое влияние оказывают эти концепции на стратегию администрирования данных?
30. Определите и поясните некоторые важнейшие факторы успеха разработки и реализации успешной стратегии администрирования данных.
31. Какие средства используются в Oracle для создания пользователей?
32. Что такое пространство таблиц (tablespace) в Oracle?
33. Что такое роль (role) в Oracle?
34. Что такое файл данных (datafile) в Oracle?
35. Что такое профиль (profile) базы данных в Oracle?
36. Что такое схема базы данных в Oracle?
37. Какая роль (role) требуется для создания триггеров и процедур в Oracle?

## Приложение 1



# Клиент/серверная сетевая инфраструктура

## Обзор

Поскольку основное внимание в клиент/серверных вычислениях уделяется совместному использованию ресурсов, строгое соблюдение сетевых стандартов имеет очень важное значение. К счастью, IEEE (Institute of Electrical and Electronic Engineers, Институт инженеров по электротехнике и электронике) разработал стандарты для обеспечения единообразия подходов в сетевых системах. Стандарты IEEE определяют технические детали, задающие топологию сети и передачу данных между совместно используемыми ресурсами. Кроме того, стандарты IEEE устанавливают правила, которым подчиняются сетевые кабели, длина кабеля между компьютерами, устройства, использующиеся в сети, и т. д.

Клиент/серверная сетевая инфраструктура включает в себя компоненты аппаратного и программного обеспечения. Основные компоненты программного обеспечения клиент/серверной сети были рассмотрены в *гл. 12*. Основные компоненты аппаратного обеспечения представляют собой кабели и коммуникационные устройства. Далее каждую сеть можно классифицировать по ее масштабу (тип сети) и по способу подключения компьютеров к другим сетевым устройствам (топология). В этом *приложении* исследуется каждый из этих компонентов.

## Сетевые кабели

Обычно кабели используются для физического соединения компьютеров и передачи данных между ними. Существуют три основных типа кабелей: витая пара, коаксиальный кабель и оптоволоконный кабель. Имеется также альтернативная возможность беспроводного подключения.

- ☐ *Витая пара* (twisted pair) — это самый распространенный вид кабеля, который легко проложить и просто обслуживать, в то же время он имеет низкую стоимость. Кабель витой пары напоминает обычный телефонный кабель и состоит из пары проводов, перевитых друг с другом внутри оболочки. Витая пара подразделяется на *экранированную витую пару* (shielded twisted pair, STP) и *неэкранированную витую пару* (unshielded twisted pair, UTP). Требования к качеству витой пары зависят от ее применения. Качество определяется по шкале качества и пригодности от категории 1 (самое низкое качество) до категории 5 (самое высокое каче-

ство). Эта шкала характеризует устойчивость кабеля к электромагнитным полям, его электрическое сопротивление, скорость передачи данных и т. д. Для клиент/серверных систем рекомендуется использовать STP или UTP категории 5.

- ❑ **Коаксиальный кабель** (иногда называют просто коаксиал). В нем используются медные провода, заключенные в два изолирующих или экранирующих слоя. Этот кабель поставляется в различных исполнениях и очень похож на антенный телевизионный кабель. Наиболее часто в локальных сетях используются тонкий и толстый коаксиал. Тонкий коаксиал дешевле и его проще прокладывать, чем толстый, но толстый коаксиал позволяет увеличить расстояние между компьютерами.
- ❑ **Оптоволоконный кабель** самый дорогой, но он обеспечивает самое высокое качество передачи данных и позволяет значительно увеличить расстояние между компьютерами. Этот кабель не подвержен влиянию электромагнитных полей, поскольку в нем используется лазерная технология передачи сигнала по стеклянному кабелю. Оптоволоконный кабель рекомендуется для соединения важных точек сети, например, для связи между двумя серверами баз данных.
- ❑ **Беспроводная связь.** Устройства беспроводной связи, например, спутник и радио, приобретают все большую популярность для подключения удаленных сайтов и как альтернатива кабельным офисным сетям. Эти устройства обладают большим потенциалом и в будущем заменят традиционные кабельные подключения.

## Топология сети

Термин *топология сети* (*network topology*) относится к способу передачи данных по сети, очень тесно связанному со способом физического соединения компьютеров. Известны три основные сетевые топологии.

- ❑ **Шинная топология** (*bus topology*). Требуется, чтобы все компьютеры подключались к главному сетевому кабелю. В этом случае сообщения, передаваемые по сети, обрабатываются всеми компьютерами на шине до тех пор, пока сообщение не достигнет конечной цели. Например, если А передает сообщение D, то это сообщение, перед тем как достичь D, пройдет через В и С, как это показано на рис. П1.1.

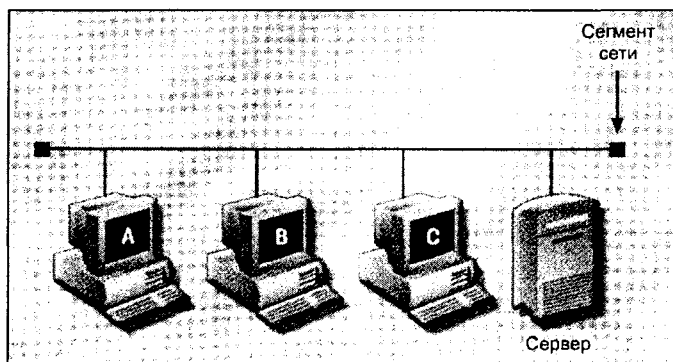


Рис. П1.1. Шинная топология

Шинная топология обычно реализуется с помощью коаксиального кабеля и широко применяется для соединения компьютеров в средних и небольших сетях. Принципиальный недостаток этой технологии состоит в том, что если узел В или С неисправен, то не будет работать и весь сегмент сети (*сегмент сети* это один фрагмент кабеля, соединяющий несколько компьютеров).

- Топология "кольцо" (*ring topology*). При этой топологии компьютеры соединяются один с другим по кабелю, проложенному в виде замкнутого кольца. Сообщения передаются от компьютера к компьютеру до тех пор, пока не достигнут пункта назначения. Компания IBM использует топологию "кольцо" для своей сети Token Ring. В реализации Token Ring в качестве концентратора используется устройство, называемое *multiple access unit (MAU, модуль множественного доступа)*, через которое сетевые компьютеры соединяются физически. Топология "кольцо", представленная на рис. П1.2, более гибка, чем шинная топология, поскольку позволяет добавлять компьютеры к "кольцу" или отключать их от кольца без влияния на остальные компьютеры.

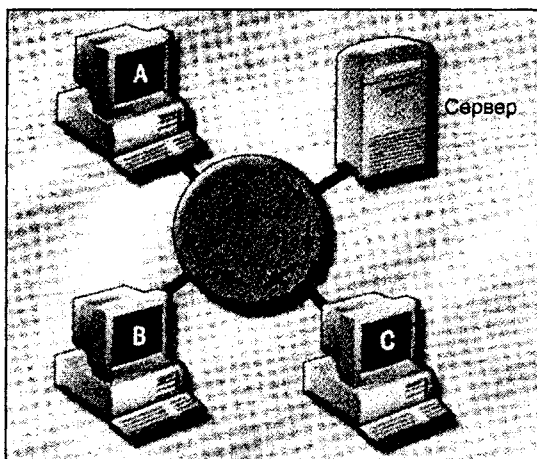


Рис. П1.2. Топология "кольцо"

В топологии "кольцо" компьютеры логически обмениваются сообщениями, передавая их по кольцу. Если сообщение послано от компьютера в сети Token Ring, оно должно использовать маркер (*token*). *Маркер* перемещается по кольцу от компьютера к компьютеру как палочка в эстафете. Передачу может осуществлять только компьютер, у которого имеется маркер.

- Топология "звезда". Позволяет всем компьютерам подключаться к центральному компьютеру или *сетевому концентратору (network hub)*, как показано на рис. П1.3. Как и в топологии "кольцо", в топологии "звезда" можно добавлять или удалять рабочие станции в сети, не оказывая влияния на остальные компьютеры. В отличие от топологии "кольцо", в топологии "звезда" компьютеры не соединены друг

с другом. Вместо этого они подключаются к центральному компьютеру или к сетевому концентратору. Поэтому все сетевые сообщения передаются через центральный компьютер или сетевой концентратор.

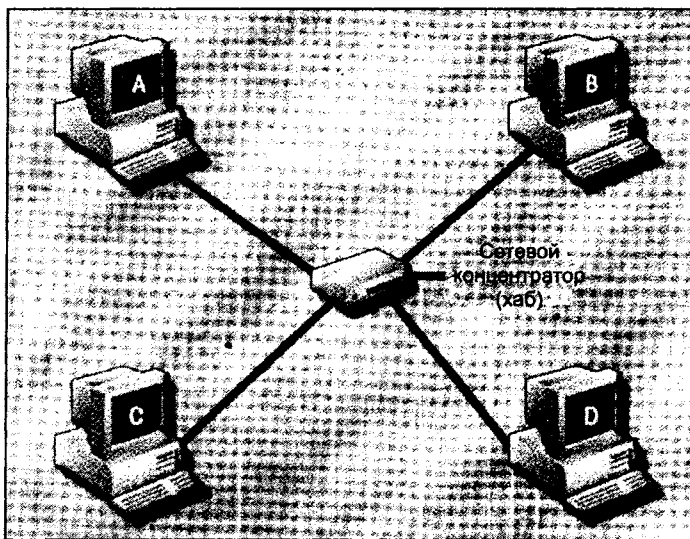


Рис. П1.3. Топология "звезда"

Сетевая топология не зависит от типа кабеля. Иначе говоря, в любой топологии может использоваться коаксиальный кабель, витая пара или оптоволоконный кабель.

## Типы сетей

Сети обычно классифицируются по географии охвата: локальные сети, университетские, городские и глобальные.

- *Локальная сеть (local area network, LAN)* обычно соединяет компьютеры офиса, отдела, этажа или здания. LAN — наиболее часто встречающийся тип сети, особенно часто используемый при подключении рабочих групп. Существуют два основных типа LAN: Ethernet и Token Ring. Каждый тип подчиняется своему набору стандартов IEEE. *Сетевой стандарт Ethernet (IEEE 802.3)* основан на шинной топологии или топологии "звезда", где может использоваться коаксиальный кабель, витая пара или оптоволоконный кабель. Большинство сетей Ethernet передают данные со скоростью 10 Мбит/с (10 миллионов бит в секунду) или 100 Мбит/с. *Сетевой стандарт Token Ring (IEEE 802.5)* основан на топологии "кольцо", где может использоваться экранированная витая пара, неэкранированная витая пара или оптоволоконный кабель. Сети Token Ring могут передавать информацию со скоростью 4 Мбит/с или 16 Мбит/с.



- ❑ *Университетская сеть (campus-wide network, CWN)*. Эти сети типичны для университетов или колледжей, где имеющиеся локальные сети (LAN) соединяются с помощью главной кабельной системы, называемой *магистральной сетью (network backbone)*.
- ❑ *Городская сеть (metropolitan area network, MAN)*. Такой тип сети используется для соединения компьютеров в масштабах города или пригородов. MAN разрабатывается для охвата более широкой территории, чем CWN. Она может даже включать в себя несколько CWN в рамках города или пригородов.
- ❑ *Глобальная сеть (wide area network, WAN)*. Глобальная сеть используется для подключения компьютеров пользователей, живущих в разных странах. Сети MAN и WAN для подключения к удаленным сайтам обычно пользуются услугами телефонных и специализированных коммуникационных компаний.
- ❑ *Беспроводная локальная сеть (wireless LAN, WLAN)*. Такая сеть используется для соединения компьютеров на основе беспроводной технологии. Сетевым стандартом для WLAN считается IEEE 802.11b.

## Сетевые коммуникационные устройства

Стандарты IEEE также определяют использование специальных сетевых коммуникационных устройств, включая сетевые интерфейсные карты, концентраторы, репитеры, мосты и маршрутизаторы. Эти устройства позволяют подключать даже несовместимые сети. Они также позволяют смешивать различные кабельные устройства внутри одной сети. Поскольку сети являются очень важным компонентом клиент/серверной архитектуры, необходимо иметь представление об основных сетевых коммуникационных устройствах.

- ❑ *Сетевая интерфейсная карта (network interface card, NIC)* представляет собой электронную плату, позволяющую подключать компьютер к сети. Сетевые карты взаимодействуют с физическими кабелями или беспроводными устройствами передачи и получения сигналов.
- ❑ *Мост (bridge)* — это устройство, соединяющее схожие сети. Мост, который позволяет компьютерам одной сети взаимодействовать с компьютерами другой сети, оперируя на уровне данных модели OSI, дает возможность управлять двумя или более сетями как одной логической сетью.
- ❑ *Репитер (repeater)* используется в сетях Ethernet для добавления сегментов сети и для расширения масштабов сети. Это устройство, которое регенерирует сигнал и передает его на все сегменты; работает на физическом уровне модели OSI.
- ❑ *Концентратор (hub)* — специальный тип репитера, позволяющий добавлять компьютеры в сеть конфигурации "звезда".
- ❑ *Маршрутизатор (router)* — интеллектуальное устройство, используемое для соединения разнородных сетей. Маршрутизаторы оперируют на сетевом уровне и позволяют использовать различные сетевые протоколы, топологии и типы кабелей. Часто используется для разделения сети на небольшие подсети. Маршрутизатор можно запрограммировать для поддержки специфических сетевых протоколов.

- *Наконимель (concentrator)* — устройство, напоминающее сетевой кабельный шкаф. Объединяет в себе множество функций, таких как мост, маршрутизатор, репитер и сегментация сети. Накопители поддерживают различные сетевые топологии, кабели и протоколы. Некоторые из них предоставляют возможности управления сетью.

Сетевые устройства используются для расширения возможностей сети и масштабов охвата, а также для взаимодействия существующих сетей со схожими или абсолютно другими сетями. Имейте в виду, что приведенный перечень далеко не полон, он представляет только некоторые примеры наиболее часто используемых устройств. Новые сетевые коммуникационные устройства, комбинирующие и расширяющие функции существующих, появляются, как грибы после дождя. Находиться на перед-ней линии новых сетевых технологий — это повседневный и бесконечный труд в современном сетевом мире.

# Глоссарий

**3GL, 4GL** — см. *язык программирования третьего поколения, язык программирования четвертого поколения.*

**ActiveX** — альтернатива Java, разработанная корпорацией Microsoft. Спецификация для разработки программ, выполняемых внутри обозревателя Microsoft Internet Explorer. Ориентирована в основном на приложения Windows и не переносима на другие операционные системы. Позволяет добавлять на Web-страницы дополнительные элементы управления, например, раскрывающиеся списки и календарь.

**ALTER** — SQL-команда для внесения изменений в структуру таблицы. Добавляет столбец или заменяет его характеристики. Дополняется ключевым словом ADD или MODIFY.

**American National Standards Institute (ANSI) (Национальный институт стандартизации США)** — группа, принявшая рекомендации рабочей группы по базам данных (DBTG) и дополнившая в 1975 году стандарты баз данных.

**AND** — логический оператор SQL, соединяющий две команды для генерации списка, содержащего объединенные требования.

**API** — см. *интерфейс прикладного программирования.*

**AVG** — функция SQL, которая выдает среднее значение по выбранному столбцу.

**BETWEEN** — SQL-оператор, предназначенный для задания диапазона ограничений.

**CASE** — см. *автоматизированные средства разработки программ.*

**CODASYL, Conference on Data Systems Languages (Постоянно действующая конференция по языкам обработки данных)** — организация, изначально сформированная для стандартизации языка COBOL. Входящая в нее группа DBTG помогла разработать стандарты баз данных в начале 1970-х гг.

**COMMIT** — SQL-команда, которая сохраняет изменения, добавления и удаления содержимого таблицы.

**COUNT** — агрегатная функция SQL, которая выдает число строк данного столбца, содержащих непустые значения, используется вместе с оператором DISTINCT.

**CREATE DOMAIN** — SQL-команда для формирования значений некоторого атрибута.

**CREATE INDEX** — SQL-команда, создающая индексы на основе какого-либо выбранного атрибута.

**CREATE TABLE** — SQL-команда, предназначенная для реализации структур таблицы с помощью характеристик и атрибутов.

**CREATE VIEW** — SQL-команда, создающая логическую, "виртуальную" таблицу, которая может рассматриваться как обычная таблица. Предохраняет пользователя от непреднамеренного добавления или удаления данных из фактической таблицы.

**Database Task Group, DBTG (Рабочая группа по базам данных)** — комитет CODASYL, помогавший разрабатывать стандарты баз данных в первой половине 1970-х гг. См. также *CODASYL.*

**DBA** — см. администратор базы данных.

**DELETE** — SQL-команда, позволяющая удалять из таблицы выбранные данные.

**DISTINCT** — SQL-оператор, предназначенный для вывода списка только значений, отличающихся друг от друга.

**DLL** — см. динамически подключаемая библиотека.

**DM** — см. менеджер данных.

**DML** — см. язык манипулирования данными.

**DROP** — SQL-команда, предназначенная для удаления таблиц.

**DTD** — см. определение типа документа.

**E-mail** — см. электронная почта.

**equiJOIN (эквисоединение)** — оператор соединения, объединяющий таблицы на основании условия эквивалентности (equality condition), по которому сравниваются выбранные столбцы таблиц.

**ER-диаграмма, диаграмма "сущность-связь" (entity relationship diagram)** — диаграмма, отражающая структуру модели взаимосвязи сущностей, атрибутов и связей. Также отражает связность и мощность.

**ER-модель, модель "сущность-связь" (entity relationship model)** — модель данных, разработанная П. Ченом (P. Chen) в 1975 году. Описывает типы связей (1:1, 1:M, M:N) между сущностями на концептуальном уровне при помощи ER-диаграмм.

**EXISTS** — специальный SQL-оператор, предназначенный для определения того, является ли значение атрибута непустым (отличным от null)).

**HTML, язык разметки гипертекста (hypertext markup language)** — стандартный язык форматирования документа для Web-страниц.

**HTTP, протокол передачи гипертекстовых файлов (hypertext transfer protocol)** — стандартный протокол, используемый Web-обозревателем и Web-сервером для связи.

**IEEE, Институт инженеров по электротехнике и электронике США (Institute of Electrical and Electronic Engineers)** — организация, разрабатывающая стандарты для унификации сетей. Стандарты IEEE определяют топологию сетей и передачу данных через носители с разрешенным доступом к ним.

**IN** — особый SQL-оператор, используемый для того, чтобы проверить, соответствует ли значение атрибута значению, находящемуся в подмножестве перечисленных значений.

**INSERT** — SQL-команда, делает возможной вставку данных в таблицу, одна строка за одну операцию. Используется для обеспечения ввода исходных данных в новые таблицы или для добавления данных в уже существующую таблицу.

**IS NULL** — особый оператор стандартного языка SQL, который совместно с оператором WHERE предназначен для проверки пустых (null) или неопределенных значений атрибутов.

**ISO (International Organization for Standardization)** — Международная организация по стандартизации.

**Java** — объектно-ориентированный язык программирования, разработанный корпорацией Sun Microsystems, который выполняется верхним уровнем программного

обеспечения Web-обозревателя. Приложения Java компилируются и хранятся на Web-сервере. Главное преимущество Java состоит в том, что однажды созданное приложение может запускаться на многих платформах.

**JavaScript** — язык подготовки сценариев (позволяет запускать серию команд или макрос), разработанный корпорацией Netscape и позволяющий Web-авторам создавать интерактивные Web-сайты. JavaScript-код встраивается в тело Web-страниц. JavaScript загружается вместе с Web-страницей и активизируется, когда возникает определенное событие — например, при щелчке кнопкой мыши на объекте.

**LAN** — см. *локальная сеть*.

**left outer JOIN (левостороннее внешнее соединение)** — в паре соединенных таблиц левостороннее внешнее соединение выводит все строки, например, таблицы ПОКУПАТЕЛИ, включая те, что не имеют совпадающих значений в таблице АГЕНТ. См. также *outer JOIN*.

**LIKE** — особый оператор стандартного языка SQL, который вместе с оператором WHERE используется для проверки на схожесть символьных строк.

**MAX** — агрегатная SQL-функция, которая выдает максимальные значения атрибутов, встречающиеся в данном столбце.

**MIN** — агрегатная SQL-функция, которая выдает минимальные значения атрибутов, встречающиеся в данном столбце.

**natural JOIN (естественное соединение)** — реляционная функция, соединяющая таблицы на основе строк с общим атрибутом (или атрибутами).

**NOT** — логический SQL-оператор (отрицание).

**OR** — логический SQL-оператор (ИЛИ), используемый для установки логических ограничений на запросы.

**ORDER BY** — управляющая SQL-команда, полезная для упорядочивания последовательности (например, по возрастанию или по убыванию).

**outer JOIN (внешнее соединение)** — операция реляционной алгебры, которая выводит таблицу, где остаются значения, совпадающие со значениями в другой таблице, а все несовпадающие значения должны принимать пустые значения (null).

**right outer JOIN (правостороннее внешнее соединение)** — использование реляционной JOIN-функции для вывода всех строк таблиц. См. также *left outer JOIN*.

**ROLLBACK** — SQL-команда, восстанавливающая содержимое таблицы базы данных в первоначальное состояние.

**SELECT** — SQL-команда, которая выводит значения всех атрибутов, найденных в таблице.

**slice and dice ("вдоль и поперек")** — на многомерном жаргоне — способность делать срезы куба данных для предоставления углубленного анализа.

**SQL** — см. *язык структурированных запросов*.

**SUM** — агрегатная SQL-функция, выдающая сумму всех значений выбранных атрибутов в данном столбце.

**TCP/IP, Transmission Control Protocol/Internet Protocol (Протокол управления передачей/Протокол Интернет)** — официальный протокол Интернета, всемирной сети неоднородных компьютерных систем.

**theta JOIN (тета-соединение)** — оператор соединения, который соединяет таблицы, используя особые условия сравнения столбцов таблиц.

**UPDATE** — SQL-команда, используемая совместно с арифметическими операторами и позволяющая делать изменения в данных.

**Web-кэширование (Web caching)** — техника повышения эффективности работы в Интернете, при которой создаются временные каталоги для хранения Web-страниц.

**Web-обозреватель, Web-браузер (Web browser)** — приложение конечного пользователя, используемое для навигации по Интернету. Запускается на клиентском компьютере и запрашивает сервисы у Web-сервера.

**Web-сайт (Web site)** — место размещения Web-сервера и коллекции Web-страниц, хранимых на локальном жестком диске серверного компьютера.

**Web-сервер (Web server)** — специализированное приложение, единственная функция которого — принимать клиентские запросы, обрабатывать их и посылать запрошенный Web-ресурс клиентскому Web-обозревателю.

**Web-страница (Web page)** — текстовый документ World Wide Web, содержащий текст и специальные команды на языке HTML.

**World Wide Web Consortium, W3C (Консорциум производителей программного обеспечения для WWW, поддерживающих его стандарты)** — международная группа, работающая с Интернетом. Ее цель — развитие открытых стандартов для Сети, в том числе, первое определение стандартов XML 1.0 1998 года.

**XML (Extensible Markup Language)** — язык разметки документов, берущий свое начало в SGML (метаязык, используемый для представления и управления элементами данных). В отличие от других языков разметки (см. HTML), XML позволяет управлять элементами данных.

**XSD, язык описания схем XML (eXtensible markup language Schema Definition)** — описание данных, которое в отличие от DTD-документа (использующего специфический синтаксис) использует синтаксис, имеющий сходство с XML-документом.

**XSL, расширяемый язык стилей (eXtensible Style Language)** — спецификация, используемая для определения правил, по которым XML-данные форматируются и отображаются, например, разделение стиля и заголовка.

**XSLT (eXtensible Style Language Transformations)** — описание основного механизма, используемого для извлечения и обработки данных из одного XML-документа и трансформации их в другой.

**Абстрактные типы данных, АТД** — типы данных, описывающие набор сходных объектов с доступными и включенными (инкапсулированными) представлениями данных и методами. Абстрактные типы данных преимущественно используются для описания сложных объектов. См. также класс.

**Автоматизированные средства разработки программ, CASE (computer-aided software engineering)** — средства, используемые для автоматизации части или всего жизненного цикла разработки систем. Технология разработки программного обеспечения в рамках некоторой выбранной методологии с использованием специальных программных пакетов (CASE-инструментария).

**Автоматическая оптимизация запроса** — метод, с помощью которого СУБД рассчитывает оптимальный способ выполнения запроса.

**Администратор базы данных (database administrator, DBA)** — лицо (или группа лиц), ответственное за планирование, организацию, контроль и слежение за централизованной и открытой для доступа базы данных компании. Администратор базы данных — это главный администратор информационного отдела.

**Администратор данных (data administrator, DA)** — лицо (или лица), ответственное за управление всеми источниками данных, как компьютеризированными, так и рукописными. Администратор данных имеет более широкие полномочия и больше ответственности, чем администратор базы данных (DBA).

**Аналитическая обработка в реальном времени (online Analytical Processing, OLAP)** — средства системы поддержки решений (DSS), использующие техники многомерного анализа данных. OLAP создает улучшенную среду анализа данных, обеспечивающую принятие решений, коммерческое моделирование и операции научных исследований.

**Аномалия данных (data anomaly)** — противоречивость данных, возникающая, когда в базе данных были сделаны некорректные изменения. Например: работник переезжает, но новый адрес указан только в одном файле, а не во всех связанных файлах базы данных.

**Аппаратная зависимость (hardware-dependent)** — свойство модели, делающее ее реализацию строго зависимой от выбранных базовых аппаратных средств.

**Аппаратная независимость (hardware independence)** — свойство модели, позволяющее ей не зависеть при реализации от базовых аппаратных средств.

**Архитектура "клиент/сервер" (client/server architecture)** — клиент/серверная инфраструктура, составленная из оборудования и программного обеспечения для того, чтобы сформировать систему из клиентов, серверов и промежуточного программного обеспечения (ППО).

**Атомарность транзакций** — все части транзакции рассматриваются как единый элемент операции, в которой все действия должны быть выполнены полностью, при обеспечении непротиворечивости базы данных.

**Атрибут** — характеристика сущности или объекта. У атрибута есть имя и тип данных. *См. также сущность.*

**База данных, БД (database)** — компьютерная структура, в которой хранится набор связанных данных. В БД содержатся данные двух типов: данные конечного пользователя (исходные факты) и метаданные. Метаданные представляют собой данные о данных, т. е. характеристики и связи данных.

**База данных предприятия (enterprise database)** — представление данных целой организации, которое предоставляет поддержку для текущих и ожидаемых в будущем потребностей.

**База данных рабочей группы (workgroup database)** — многопользовательская база данных, которая поддерживает сравнительно небольшое количество пользователей (обычно меньше 50) или база данных отдельного учреждения или организации.

**База данных системы принятия решений (decision support database)** — база данных, основная цель которой — структурирование информации, необходимой для принятия тактических или стратегических решений на средних и высших уровнях управления.

**Рабочая база данных (production database)** — база данных, предназначенная в большей степени для "быстрого реагирования", качество работы которой в значительной

степени определяется скоростью ответа на запрос (скорость отклика). Рабочую базу данных также называют организационной или *транзакционной базой данных* (подчеркивая ее направленность на выполнение транзакций).

**Транзакционная база данных** — см. *рабочая база данных*.

**Базовые типы данных (base data types)** — этим термином определяются типы данных, наиболее часто используемые в традиционных языках программирования. К базовым типам данных относятся *real* (действительный), *integer* (целый) и *string* (строковый).

**Безопасность (security)** — 1) защита данных от случайного или преднамеренного использования неавторизованными пользователями; 2) сервисы, разработанные для гарантии защиты и конфиденциальности данных с помощью шифрования, электронных сертификатов (SSL, S-HTTP), брандмауэров и прокси-серверов.

**Беспроводная локальная сеть (wireless LAN, WLAN)** — локальная сеть, компоненты которой соединяются не по проводам, а при помощи беспроводных средств (радиоволны, инфракрасные лучи).

**Бизнес-бизнес (business to business, B2B)** — вид интернет-ресурса. Ресурс такого типа ориентирован на осуществление операций и поддержку отношений между компаниями.

**Бизнес-клиент (business to consumer, B2C)** — вид интернет-ресурса. Ресурс такого типа ориентирован на осуществление операций и поддержку отношений между компаниями и клиентами.

**Бизнес-правила** — текстовое описание установок, правил и технологических приемов внутри организации. Например: *Пилот не может в течение суток работать больше десяти часов. Профессор может читать до четырех курсов лекций в течение одного семестра*.

**Бинарная связь** — термин, выражающий характер связи логических объектов и используемый для описания связей между двумя сущностями. Пример: ПРОФЕССОР читает КУРС.

**БКНФ** — см. *нормальная форма Бойса—Кодда*.

**Блокировка (lock)** — механизм, обеспечивающий монопольное использование ячейки данных в определенной транзакции и предотвращающий использование той же самой ячейки данных другой транзакцией. Транзакция требует заблаговременной блокировки доступа к ячейке данных, подобная блокировка снимается после выполнения операции, чтобы предоставить другой транзакции возможность заблокировать ячейку данных для работы с ней.

**Блокировка на уровне поля (field-level lock)** — позволяет параллельным транзакциям иметь доступ к одной и той же строке при условии, что они будут использовать разные поля (атрибуты) строки. Предоставляет наиболее гибкий доступ к данным для нескольких пользователей, при этом требуются компьютеры повышенной мощности.

**Блокировка на уровне страниц (page-level lock)** — при этом типе блокировки система управления базами данных заблокирует всю страницу на диске или раздел (section) диска.

**Блокировка на уровне строки (row-level lock)** — сравнительно менее ограничивающая блокировка базы данных, при которой СУБД позволяет одновременным транзакци-



ям иметь доступ к различным строкам одной и той же таблицы, даже если строки расположены на одной странице.

**Блокировка на уровне таблицы (table-level lock)** — сравнительно мало ограничивающая блокировка, при которой заблокирована целая таблица, не предоставляющая доступ транзакции T1 ни к одной из строк, до тех пор пока транзакция T2 не закончит работу с таблицей.

**Булева алгебра** — раздел математики, который работает с логическими операторами И, ИЛИ и НЕ.

**Буфер** — резервный блок первичной памяти, где временно хранятся данные, считанные из вторичной памяти. Позволяет снизить число операций ввода/вывода между первичной и вторичной памятью за счет использования первичной памяти, более быстрой по сравнению с вторичной памятью.

**Модель OSI** — международная программа стандартизации обмена данными между компьютерными системами различных производителей на основе семиуровневой модели протоколов передачи данных в открытых системах (прикладной уровень, уровень представления, уровень сеанса, транспортный уровень, сетевой уровень, уровень канала передачи данных и физический уровень), предложенная *ISO*.

**Витая пара (twisted-pair)** — сетевой кабель, образованный парой проводов, закрученных внутри защитной изоляции; обычный кабель во многих сетях, используемый из-за простоты его прокладки и низкой цены. Напоминает обычный телефонный кабель.

**Витрина данных (data mart)** — небольшое, однообъектное подмножество информационного хранилища, предоставляющее небольшой группе людей средства поддержки принятия решений.

**Вложенный запрос (nested query)** — запрос, находящийся внутри запроса. Обычно состоит из одного внешнего цикла и одного или более внутренних, находящихся внутри внешнего.

**Внешний ключ (foreign key)** — *см. ключ*.

**Внешняя модель (external model)** — представление прикладного программиста о структуре данных. Она работает с подмножествами данных глобальной схемы базы данных с фокусировкой на коммерческие понятия.

**Внутренняя модель (internal model)** — приспособливает концептуальную модель к особенностям СУБД при реализации.

**Восстановление базы данных (database recovery)** — восстановление базы данных до ее предыдущего непротиворечивого состояния. Техники восстановления основываются на атомарном свойстве транзакции.

**Восходящее проектирование (bottom-up design)** — стратегия проектирования, которая начинается с выявления индивидуальных компонентов разработки с их последующей агрегацией. В разработке баз данных это процесс, который начинается с определения атрибутов и их группировки в сущности. *См. нисходящее проектирование*.

**Вторая нормальная форма, 2НФ (second normal form, 2NF)** — второй этап процесса нормализации, при котором в таблице, приведенной к *1НФ*, устраняются частичные зависимости (исключаются атрибуты, частично зависящие от первичного ключа).

**Вторичный ключ (secondary key)** — ключ, используемый только для поиска данных. Например, покупатель не всегда знает свой абонентский номер (первичный ключ), но комбинация имени, фамилии, отчества и телефонного номера обычно идентифицирует требуемую строку столбца.

**Гиперкубы (hypercubes)** — значение данных в  $n$ -мерном кубе данных ( $n > 3$ ). Помогает конечным пользователям МСУБД представить данные в многомерной форме. См. Также *куб данных*.

**Гиперссылка (hyperlink)** — ссылка между Web-страницами в гипертекстовом или каком-либо другом типе электронного документа.

**Глобальная сеть (wide area network, WAN)** — тип сети для подключения компьютерных пользователей во всемирном масштабе; в основном работает при помощи телефонных компаний.

**Горизонтальная фрагментация (horizontal fragmentation)** — процесс, относящийся к разработке распределенных баз данных, который разбивает таблицу на подмножества, состоящие из уникальных строк.

**Городская, или региональная сеть (metropolitan area network, MAN)** — тип сети, используемый для соединения компьютеров по всему городу и пригородам.

**Денормализация (denormalization)** — операция, понижающая уровень нормализации путем объединения таблиц. Чаще всего это делается для повышения производительности БД, за что приходится расплачиваться избыточностью данных и возможным появлением аномалий. Следует использовать с осторожностью. См. также *нормализация*.

**Динамическая Web-страница (dynamic Web pages)** — страница с содержимым, обновляемым во времени, например, система онлайн-заказа товаров.

**Динамическая оптимизация запросов (dynamic query optimization)** — метод оптимизации запросов, с помощью которого способ обращения к данным в базе данных определяется во время выполнения запроса или программы.

**Динамически подключаемая библиотека (Dynamic Link Library, DLL)** — открытый для доступа код, который рассматривает интерфейс прикладного программирования (API) как часть программы Web-сервера, причем так, что он может при необходимости инициализироваться динамически.

**Домен (domain)** — компонент реляционной базы данных, предназначенный для формирования и описания набора возможных значений атрибута.

**Единичное наследование (single inheritance)** — в объектно-ориентированной модели данных свойство объекта, позволяющее ему иметь только одного родителя из супер-класса, от которого он наследует свои структуры данных и методы. См. также *наследование*.

**Жизненный цикл базы данных (Database Life Cycle, DBLC)** — история развития базы данных в пределах информационной системы. Делится на шесть этапов: начальный этап, проектирование, реализация и загрузка, проверка и оценка, функционирование и эксплуатация, а также модернизация.

**Журнал транзакций (transaction log)** — используется СУБД для сохранения следов всех операций транзакций, обновляющих базу данных. Информацию, хранимую в этом журнале, СУБД использует для восстановления базы данных при сбое.

**Зависимость по данным (data dependence)** — состояние данных, в котором представление и действия над данными зависят от способа физического хранения данных.

**Запись (record)** — набор связанных (логически соединенных) полей.

**Запрос (query)** — вопросы или задачи, адресуемые базе данных конечным пользователем.

**Запрос к базе данных (database request)** — эквивалент одиночного SQL-оператора в прикладной программе или транзакции.

**Идентификатор объекта (object ID, OID)** — генерируемый системой идентификатор объекта, не зависящий от положения объекта и его физического адреса в памяти.

**Иерархическая модель базы данных (hierarchical database model)** — данная модель основана на структуре "перевернутого" дерева, в которой каждая запись называется сегментом. Самая верхняя запись — корневой сегмент (*корень*). Каждый сегмент связан с сегментами, расположенными сразу под ним, связью 1:M.

**Иерархический путь (hierarchical path)** — упорядоченная последовательность сегментов, которые должны быть доступны для СУБД для отыскания данного сегмента.

**Иерархия классов (class hierarchy)** — структура классов в иерархическом дереве, где каждый "родительский" класс это *суперкласс*, а каждый класс-"потомок" — *подкласс*.

**Иерархия обобщенных представлений (generalization hierarchy)** — используется в модели отношений логических объектов для отражения связей между множеством объектов-супертипов и одним или более множеством объектов-подтипов. В этой иерархии объект-супертип представляется предком каждого объекта-подтипа.

**Избыточные данные (redundant data)** — размноженные данные, хранимые более чем в одном месте.

**Извлечение данных (data extraction)** — процесс извлечения и проверки правильности данных, взятых из рабочей базы данных и внешних источников данных. Также называется *фильтрацией данных*.

**Индекс (index)** — средство, необходимое базе данных для того же, для чего нужен предметный указатель в книге. Индексный файл состоит из ссылочных значений — ключей индекса и множества указателей.

**Инкапсуляция (encapsulation)** — возможность скрыть внутреннее представление данных и методов объекта метода от внешних объектов. Термин объектно-ориентированного программирования.

**Инкрементальное резервное копирование (incremental backup)** — копирование всех новых данных, которые были добавлены в базу данных, начиная с предыдущей даты резервного копирования. Гарантирует полное восстановление всех данных в случае физического повреждения или полного краха базы данных. Эта функция может быть запущена вручную или в автоматическом режиме через определенные промежутки времени с помощью средств базы данных.

**Инспектор безопасности БД (Database Security Officer, DSO)** — лицо, ответственное за безопасность, целостность, резервирование и восстановление базы данных.

**Интернет** — глобальная сеть компьютеров, соединенных с помощью стандартного сетевого протокола TCP/IP (*Transmission Control Protocol/Internet Protocol*). Интернет можно себе представить как магистраль передачи данных. Термины *Internet* и *World Wide Web* часто заменяют друг друга, но они *не* синонимы.

**Интерфейс прикладного программирования (API)** — программный продукт, с помощью которого программисты взаимодействуют с промежуточным программным обеспечением. Позволяет использовать обобщенный код SQL, тем самым обеспечивает клиентским процессам независимость от сервера базы данных.

**Интерфейс уровня вызовов (Call Level Interface, CLI)** — стандарт, разработанный организацией SQL Access Group для доступа к базам данных.

**Интерфейсное приложение (front-end application)** — любой процесс, который запрашивает сервисы у обслуживающих процессов. *См. также клиент.*

**Инtranet (intranet)** — компьютерная сеть, предназначенная для использования внутри компании, является ее собственностью и ею же управляется. К таким сетям имеют доступ только компьютеры, находящиеся внутри компьютерной сети этой компании. Целью подобных систем является улучшение операций компании посредством улучшенной системы управления доступа к данным и коммуникациям.

**Информационное хранилище (data warehouse)** — структура базы данных информационной организации, предпочитаемая системами, обеспечивающими принятие решений. Это интегрированная, субъектно-ориентированная, зависящая от времени, долговременная база данных, предоставляющая поддержку принятия решений. *См. также система принятия решений.*

**Информационные системы (Information Systems, IS)** — системы, предоставляющие возможность накопления, хранения и выборки данных; облегчают структурирование данных и управление ими. Информационная система включает в себя аппаратные средства, программное обеспечение (СУБД и приложения), базу (базы) данных, персонал и методики.

**Информационный инжиниринг, инфотехника (information engineering)** — методика преобразования стратегических целей компании в данные и приложения, помогающие достигнуть целей компании.

**Кардинальное число, мощность (cardinality)** — выражает значение связности компонентов. Задаёт число допустимых вхождений сущностей, связанных с единым местонахождением объединенной сущности.

**Каталог распределенных данных (Distributed Data Catalog, DDC)** — словарь данных, который содержит описание (имя фрагмента, место) распределенной базы данных.

**Класс (class)** — совокупность сходных объектов с общей структурой (атрибутами) и поведением (методами). Класс инкапсулирует представление данных объекта и реализацию метода. Классы организованы в иерархию классов. *См. также абстрактные типы данных.*

**Кластерные таблицы (cluster tables)** — таблицы, хранящие строки разных таблиц вместе в упорядоченных дисковых областях для ускорения доступа к данным.

**Клиент (client)** — любой процесс, требующий от обслуживающего процесса особых сервисов. *См. клиент/сервер.*

**Клиент/сервер (client/server)** — вычислительная модель, основанная на распределении функций по двум типам независимых процессов: клиенты и серверы. Клиенты делают запросы, а серверы отвечают на запросы клиентов. Клиенты и серверы могут существовать как на одном компьютере, так и на разных компьютерах, соединенных сетью. Ключевая особенность — разделение задач по обработке данных.

**Ключ (key)** — идентификатор сущностей, основанный на концепции функциональных зависимостей; может быть классифицирован следующим образом: *Суперключ*: атрибут (или комбинация атрибутов), уникально определяющий каждую сущность в таблице. *Потенциальный ключ*: минимальный суперключ, т. е. не содержащий подмножества атрибутов, которые сами по себе суперключи. *Первичный ключ*: потенциальный ключ, выбранный в качестве уникального идентификатора сущности. *Вторичный ключ*: ключ, используемый только для извлечения данных. *Внешний ключ*: атрибут (или комбинация атрибутов) в одной таблице, значения которого должны соответствовать первичному ключу в другой таблице или быть пустыми (null).

**Ключевые атрибуты (key attributes)** — формируют первичный ключ сущности. *См. также первичный атрибут.*

**Коаксиальный кабель (coaxial cable)** — медные кабели, окруженные двумя слоями изоляции или экранирования. Иногда называется просто коаксиал. Очень похож на телевизионный кабель.

**Конечный пользователь (end user)** — лицо, использующее прикладные программы для запуска ежедневных операций организации (продавец, диспетчер и т. д.) Конечные пользователи высокого уровня используют информацию из баз данных для принятия стратегических решений.

**Контроль версий (versioning)** — свойство ООСУБД, позволяющее базе данных хранить следы всех сделанных изменений.

**Контроль параллельного выполнения операций (concurrency control)** — характеристика СУБД, позволяющая координировать одновременное выполнение транзакций в системе базы данных при сохранении целостности.

**Контрольная точка (checkpoint)** — операция, во время которой СУБД записывает на диск все содержимое своих обновленных буферов.

**Контрольный журнал (audit log)** — опция системы управления базой данных, автоматически записывающая короткое описание операций базы данных, которые производились всеми пользователями.

**Конфиденциальность (privacy)** — контроль использования данных, связанный с правами персон и предприятия, для определения того, "кто, что, когда, где и как".

**Концептуальная модель (conceptual model)** — абстрактное представление данных с точки зрения администраторов и разработчиков баз данных. Описывает основные объекты данных, избегая деталей.

**Концептуальное проектирование (conceptual design)** — процесс, использующий технику моделирования данных для создания модели структуры базы данных, представляющей объекты реального мира наиболее правдоподобным способом. Не зависит как от программного обеспечения, так и от оборудования.

**Координатор (coordinator)** — узел процессора транзакций (TP), координирующий исполнение двухфазной команды COMMIT в системе управления распределенными базами данных. *См. также процессор данных, процессор транзакций.*

**Корень (root)** — верхний (начальный) компонент иерархического дерева базы данных.

**Кортеж (tuple)** — строка таблицы базы данных.

**Куб данных (data cube)** — данные, взятые из рабочей базы данных или информационного хранилища, помогающего конечным пользователям МСУБД представить

хранимые данные в виде трехмерного куба. Местоположение каждого значения данных в кубе данных основывается на осях X, Y и Z куба. Данные являются статистическими (созданы до того, как они используются), поэтому их невозможно получить с помощью *нерегламентированного запроса*.

**Кэш куба (cube cache)** — кэш-память, в которой хранятся кубы данных. Использование кэша куба позволяет ускорить доступ к данным.

**Логический формат данных (data logical format)** — данные с точки зрения человека, когда не принимаются во внимание истинные физические характеристики данных, ("с точки зрения" компьютера). *См. также физический формат данных.*

**Логическое проектирование (logical design)** — этап разработки, на котором концептуальный проект сопоставляется с требованиями выбранной СУБД. Зависит от выбранного программного обеспечения СУБД. Раньше использовалось для преобразования концептуального проекта во внутреннюю модель выбранной СУБД (DB2, SQL Server, Oracle, IMS, Informix, Access и Ingress).

**Локальная сеть (local area network, LAN)** — компьютерная сеть, занимающая небольшую площадь, например, одно здание. Позволяет предоставлять доступ конечным пользователям персональных компьютеров к файлам через центральный компьютер, работающий как сетевой файл-сервер. С помощью локальной сети многочисленные пользователи могут получать доступ через сеть к устройствам, например, к принтерам.

**Магистраль сети (network backbone)** — главная кабельная система сети для одной или более локальных сетей.

**Маркер (token)** — в сетевой топологии "кольцо", метка, переходящая от одного компьютера к другому, это похоже на передачу эстафетной палочки. В настоящий момент времени данные может передавать только компьютер с маркером.

**Маршрутизатор (router)** — 1) интеллектуальное устройство, применяемое для соединения сетей, использующих разные архитектуры и протоколы; 2) аппаратное оборудование или программное обеспечение, соединяющее много разнотипных сетей.

**Менеджер блокировок (lock manager)** — компонент СУБД, ответственный за установку и снятие блокировок.

**Менеджер данных (data manager, DM)** — специалист по обработке данных, занятый наблюдением за работой отдела. Сюда входят: управление техническими и кадровыми ресурсами, надзор за старшими программистами, локализация неисправностей в программах.

**Метаданные (metadata)** — данные о данных, т. е. данные, информирующие о характеристиках и связях данных. *См. также словарь данных.*

**Метка (штамп) времени (time stamping)** — подход к планированию одновременных транзакций путем назначения глобальной уникальной метки времени каждой транзакции.

**Метрики (metrics)** — факты, полученные во время функционирования системы, в противоположность постоянно хранимым фактам.

**Многомерная аналитическая обработка в реальном времени (Multidimensional Online Analytical Processing, MOLAP)** — расширяет функциональность аналитической обработки в реальном времени до многомерных систем управления базами данных.

**Многомерные системы управления базами данных, МСУБД (multidimensional database management systems, MDBMS)** — базы данных, которые используют патентованные техники хранения данных в  $n$ -мерных массивах, подобных матрице.

**Многопользовательская СУБД (multiuser DBMS)** — система управления базами данных с несколькими пользователями.

**Множественное наследование (multiple inheritance)** — форма наследования классами функций и свойств, при которой производный класс может иметь любое число базовых (родительских).

**Множество (set)** — в сетевой модели — описание связи 1:М между владельцем структурного типа и членом структурного типа.

**Модель "птичья лапка" (Crow's Foot model)** — версия ER-диаграммы, использующая для представления стороны связи "многие" символ в виде трехпальцевой птичьей лапки.

**Модель (model)** — описание или аналогия, используемая для представления чего-либо, что не может быть рассмотрено непосредственно; упрощенные абстракции событий или условий реального мира.

**Модель базы данных (database model)** — совокупность логических конструкций, используемых для описания и представления как структуры данных, так и связей данных, а также операций с данными, содержащихся в базе данных. *См. также модель данных.*

**Модель данных (data model)** — представление, обычно графическое, сложной структуры данных реального мира. Модели данных обычно используются на этапе разработки жизненного цикла баз данных. *См. также модель базы данных.*

**Модель Чена (Chen model)** — диаграмма связей сущностей, описанная в канонических записях Питера Чена (Peter Chen); характеризуется использованием ромбов для изображения связей и прямоугольников для изображения сущностей.

**Модуль (module)** — 1) сегмент разработки, который может быть выполнен как самостоятельный элемент, иногда связанный, для того чтобы породить систему; 2) компонент информационной системы, управляющий особой функцией, например, инвентаризацией, заказами, расчетной ведомостью и т. д.

**Мост (bridge)** — аппаратное устройство для соединения сходных сетей. Позволяет компьютерам одной сети взаимодействовать с компьютерами другой сети.

**Накопитель (concentrator)** — устройство, соединяющее множество проводов, для того чтобы позволить одновременно большому количеству пользователей иметь доступ к сети; сетевой кабельный шкаф.

**Наследование (inheritance)** — в объектно-ориентированной модели свойство объекта наследовать структуру данных и методы надклассов в классовой иерархии. *См. также иерархия классов.*

**Независимость от данных (data independence)** — состояние, когда доступ к данным не зависит от изменения характеристик физического хранения данных.

**Независимость от программных средств (software independence)** — свойство любой модели или приложения, когда оно не зависит от программного обеспечения, использованного для его реализации.

**Неключевой атрибут** — *см. непервичный атрибут.*

**Непервичный атрибут (nonprime attribute)** — неключевой атрибут, т. е. атрибут, который не является даже частью ключа.

**Нерегламентированные запросы** — запросы к базе данных, создаваемые под влиянием текущей ситуации.

**Неэкранированная витая пара (unshielded twisted-pair, UTP)** — витая пара без защитного покрытия, предназначенного для работы в тяжелых условиях, что делает ее более восприимчивой к электромагнитным пробоям.

**Нисходящее проектирование (top-down design)** — стратегия разработки, которая начинается с выявления основных макроструктур системы и затем переходит к определению самых мелких элементов в пределах этих структур. В разработке баз данных это процесс, который начинается с определения сущностей, а затем определяются атрибуты.

**Первая нормальная форма, 1НФ (first normal form, 1NF)** — первый уровень процесса нормализации, при котором в таблице определены ключевые атрибуты, отсутствуют повторяющиеся группы, а все атрибуты зависят от первичного ключа.

**Нормализация (normalization)** — процесс, который присваивает сущностям атрибуты так, что избыточность данных уменьшается или исчезает.

**Нормальная форма Бойса—Кодда** — особый вид третьей нормальной формы (3НФ), в которой каждый определитель является потенциальным ключом. Таблица, приведенная к БКНФ, является приведенной и к 3НФ.

**Оборудование (hardware)** — компьютер и соответствующие комплектующие.

**Обработка транзакций (transaction processing)** — в электронной коммерции — серия действий, изменений и/или функций среди тысяч подключенных покупателей.

**Общий шлюзовой интерфейс (Common Gateway Interface, CGI)** — стандарт взаимодействия серверных процессов с внешними приложениями.

**Объект (object)** — абстрактное представление сущности реального мира, у которого есть уникальные особенности, встроенные свойства и возможность взаимодействовать с другими объектами и с собой.

**Объект базы данных (database object)** — любой объект базы данных, созданный конечным пользователем, например, таблицы, представления, индексы, хранимые процедуры и триггеры. Должны управляться DBA.

**Объектно-реляционная система управления базами данных, ОРСУБД (object/relational database management system, O/RDBMS)** — модель данных, основанная на расширенной реляционной модели данных, которая, по существу, включает в себя многие лучшие черты объектно-ориентированных моделей при более простом структурном окружении реляционной базы данных.

**Объектно-ориентированная модель данных, ООМД (Object-Oriented Data Model, OODM)** — модель данных на основе объектного подхода. Предоставляет поддержку определенных пользователем *типов данных, наследования, полиморфизма, инкапсуляции* и т. д.

**Объектно-ориентированная система управления базой данных, ООСУБД (Object-Oriented Database Management System, OODBMS)** — программное обеспечение, используемое для лучшего управления данными в объектно-ориентированной модели базы данных.



**Объектно-ориентированное программирование, ООП (Object-Oriented Programming, OOP)** — альтернатива методам традиционного программирования, основанная на объектно-ориентированных понятиях. Экономит время программирования и количество строк кода, увеличивает производительность труда программиста.

**Объектно-ориентированный язык программирования (Object-Oriented Programming Language, OOPL)** — язык программирования, основанный на объектно-ориентированных понятиях.

**Объектно-ориентированный язык запросов (Object Query Language, OQL)** — язык запросов к базе данных, используемый объектно-ориентированной системой управления базами данных.

**Одновременное резервное копирование (concurrent backup)** — резервное копирование, производимое во время работы с базой данных одного или нескольких пользователей.

**Однозначный атрибут (single-valued attribute)** — атрибут, который может иметь только одно значение.

**Однопользовательская СУБД (single-user DBMS)** — классификация системы управления базами данных с единственным пользователем.

**Омоним (homonym)** — использование одного имени для разных атрибутов. Как правило, этого следует избегать. Некоторое реляционное программное обеспечение автоматически проверяет систему на омонимы и либо предупреждает пользователя об их существовании, либо автоматически делает нужные корректировки. *См. также синоним.*

**Определение типа документа (Document Type Definition, DTD)** — файл с расширением dtd, который описывает элементы XML; в действительности, DTD-файл предоставляет набор логических моделей и определяет синтаксические правила или допустимые теги для каждого типа XML-документа.

**Оптимистический подход (optimistic approach)** — в управлении параллельным выполнением операций этот подход основан на допущении, что большинство операций базы данных не конфликтует и транзакции выполняются без ограничений до тех пор, пока являются фиксированными.

**Оптоволоконный кабель (fiber-optic cable)** — позволяет достигать наивысшей скорости передачи информации и дает возможность тысячам пользователей получать одновременный доступ к сети по одной кабельной линии. Остается наиболее дорогим методом передачи информации из-за сложного лазерного оборудования, используемого при передаче сигнала.

**Острова информации (islands of information)** — независимые массивы данных (data pool), созданные и управляемые разными организационными частями. Типичны для устаревших систем файлов.

**Открытый интерфейс доступа к базам данных (Open database Connectivity, ODBC)** — промежуточное программное обеспечение баз данных, разработанное корпорацией Microsoft для предоставления API к приложениям Windows.

**Отношения (relations)** — таблицы, связанные друг с другом совместным использованием общих характеристик сущностей.

**Первичный атрибут (prime attribute)** — ключевой атрибут, т. е. атрибут, который является, как минимум, частью ключа.

**Первичный ключ (primary key)** — см. *ключ*.

**Переменные экземпляра (instance variables)** — см. *атрибут*.

**План доступа** — хранимая процедура, сгенерированная во время компиляции приложения, созданного и управляемого СУБД. План доступа определяет то, как запрос приложения будет обращаться к базе данных во время выполнения.

**Поддержка принятия решений (decision support)** — методика (или ряд методик), разработанная для структурирования информации и ее использования в качестве основы для принятия решений.

**Подкласс/суперкласс (subclass/superclass)** — см. *иерархия классов*.

**Подключаемый модуль (plug-in)** — внешнее клиентское приложение, которое при необходимости автоматически выполняется обозревателем.

**Подсхема (subschema)** — подмножество схемы. Подсхема определяет части базы данных так, как они "видны" прикладным программам, которые их используют. См. также *схема*.

**Позднее связывание (late binding)** — свойство, при котором тип данных атрибута неизвестен до времени выполнения или рабочего цикла.

**Поле (field)** — символ или группа символов (алфавитно-числовых), определяющих характеристику персоны, места или вещи. Например, социальное обеспечение, адрес, телефон, состояние банковского счета и т. д. — все это поля.

**Полиморфизм (polymorphism)** — характеристика объектно-ориентированной модели данных, с помощью которой различные объекты отвечают одним и тем же сообщениям разными способами.

**Политики (policies)** — основные правила, используемые для управления операциями компании, обеспечивая достижение целей компании.

**Полная функциональная зависимость (full functional dependence)** — положение, при котором атрибут функционально зависит от составного ключа, но не от любого подмножества этого ключа.

**Потенциальный ключ (candidate key)** — см. *ключ*.

**Потомок (child)** — сегмент, расположенный в иерархическом дереве ниже другого сегмента; каждый нижний сегмент является потомком сегмента, находящегося над ним и непосредственно с ним связанного. См. также *предок*.

**Предок (parent)** — в иерархическом дереве — сегмент, расположенный над другим сегментом; каждый сегмент является предком (родителем) другого, расположенного под ним.

**Проверка (verification)** — процесс, посредством которого концептуальная модель проверяется на достоверность всех транзакций базы данных (INSERT, UPDATE, DELETE и OUTPUT).

**Программа пакетного обновления данных** — операция, соединяющая транзакции в единый "командный файл" (batch-файл), чтобы обновить поля таблицы за одну операцию.

**Программное обеспечение (software)** — операционная система, утилиты (обслуживающие программы), файлы, программы управления файлами и прикладные программы (приложения), которые генерируют отчеты из данных, хранимых в файлах.

**Проектирование базы данных (database design)** — процесс, предоставляющий описание структуры базы данных. Это вторая фаза жизненного цикла базы данных, во время которой выявляются компоненты базы данных.

**Прозрачность выполнения (performance transparency)** — свойство СУРБД, позволяющее системе работать так, как будто бы она была централизованной СУБД (без увеличения времени реакции).

**Прозрачность распределения (distribution transparency)** — свойство системы управления распределенной базой данных, позволяющее пользователю рассматривать базу данных как единую систему.

**Прозрачность транзакций (transaction transparency)** — свойство СУРБД, гарантирующее, что транзакции будут поддерживать целостность и непротиворечивость распределенной базы данных. Гарантирует, что транзакции будут завершены только в том случае, если все узлы базы данных завершат свои части транзакции.

**Промежуточное программное обеспечение, ППО (middleware)** — компьютерное программное обеспечение, позволяющее клиентам и серверам работать в архитектуре "клиент/сервер". Промежуточное программное обеспечение отделяет клиентские процессы от сетевых протоколов и деталей серверных протоколов обработки.

**Пространство объекта (object space)** — эквивалент схемы базы данных с точки зрения разработчика объектно-ориентированной базы данных.

**Противоречивость данных (data inconsistency)** — состояние, при котором разные версии одних и тех же данных выдают разные (противоречивые) результаты.

**Протокол (protocol)** — особый набор правил, нужный для выполнения определенной функции. В объектно-ориентированной модели данных протокол соответствует коллекции сообщений, которым отвечает объект.

**Протокол передачи файлов (File Transfer Protocol, FTP)** — используется для предоставления возможностей передачи файлов между компьютерами в сетях Интернет/интранет, использующих общественные или известные имена для классификации и группировки сообщений.

**Процедурный SQL (procedural SQL, PL/SQL)** — позволяет использовать процедурный код и SQL-операторы, хранимые в БД.

**Процессор данных (data processor, DP)** — компонент программного обеспечения, устанавливаемый на каждом компьютере, который хранит и извлекает данные с помощью системы управления распределенной базой данных. Процессор данных отвечает за управление локальными данными компьютера и координирует доступ к подобным данным. *См. также процессор транзакций.*

**Процессор транзакций (transaction processor, TP)** — компонент программного обеспечения на каждом компьютере, который запрашивает данные через СУРБД. Процессор транзакций отвечает за выполнение и координацию всех баз данных, созданных локальным приложением, и имеет доступ к данным на любом процессоре данных. *См. также процессор данных.*

**Псевдоним** — альтернативное имя, используемое, например, чтобы обозначить таблицу в SQL-запросе.

**Пустое значение (null)** — отсутствие значения атрибута. Обратите внимание, что null это не пробел.

**Распределенная обработка данных (distributed processing)** — предоставление доступа (разделения) к логической обработке данных через два или более узлов сети.

**Распределенный запрос (distributed request)** — запрос к базе данных, позволяющий одному SQL-оператору иметь доступ к данным на нескольких удаленных процессорах данных в распределенной базе данных.

**Рекурсивная связь (recursive relationship)** — сущность, которая выражает связь по отношению к себе самой.

**Реляционная алгебра (relational algebra)** — набор математических принципов, формирующих основу управления содержимым реляционных таблиц; включает в себя восемь основных функций: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT и DIVIDE.

**Реляционная база данных (relational database)** — структура данных, хранящая информацию о сущностях, атрибутах этих сущностей, и связях между этими сущностями. Воспринимается пользователем как набор таблиц.

**Реляционная схема (relational schema)** — описание организации базы данных с точки зрения администратора базы данных, где показаны соединяющие поля и типы связей.

**Репитер (repeater)** — устройство, используемое в сетях Ethernet для добавления к сети сегментов сети и расширения пределов досягаемости сигнала.

**Репликация (replication)** — процесс создания и управления дублированными версиями базы данных. Используется для размещения копий в различных местах и для уменьшения времени доступа.

**Репликация данных (data replication)** — хранение дублированных фрагментов базы данных на нескольких узлах системы управления распределенными базами данных. Дублирование подобных фрагментов прозрачно для конечного пользователя.

**Роль (role)** — процесс агрегирования данных на более высоком уровне для анализа, выделения и представления информации.

**Связность (connectivity)** — описывает классификацию связей между сущностями. Сюда включаются связи 1:1, 1:M и M:N.

**Связь (relationship)** — взаимодействие между сущностями. В ER-диаграмме объектов отображается в виде ромба. Ее уровень определяется по количеству связанных сущностей: унарная, бинарная, тернарная или выше.

**Сегмент сети (network segment)** — одна секция кабеля, подсоединенная к нескольким компьютерам.

**Семантическая модель данных (Semantic Data Model, SDM)** — первая из серии моделей данных, ближе соответствующая реальному миру; моделирует как данные, так и связи между ними в единой структуре, называемой объектом. SDM, опубликованная в 1981 году, была разработана М. Хаммером (M. Hammer) и Д. Мак-Леодом (D. Mcleod).

**Сервер (server)** — любой процесс, предоставляющий запрашиваемые сервисы клиентам. *См. клиент/сервер.*

**Сервер Web-приложений (Web application server)** — приложение промежуточного программного обеспечения, расширяющее функциональность Web-серверов путем связывания их широким кругом сервисов, таких как базы данных, системы каталогов, поисковые машины и т. д.

**Серверное приложение** — процесс, предоставляющий сервис клиентам. *См. также сервер.*

**Серверное расширение (server-side extension)** — программа, взаимодействующая непосредственно с *Web-сервером* для управления запросами особых типов. Такие программы значительно расширяют функциональные возможности *Web-сервера* и, таким образом, интранета.

**Серверные инструментальные средства CASE** — компьютерные полуавтоматические программные средства, предоставляющие поддержку на этапах кодирования и выполнения синхронного управления передачей данных (SDLC). Например, программные средства интерфейсной части (front-end) обеспечивают поддержку на этапах планирования, анализа и разработки.

**Сервис (service)** — в Windows NT — специальная программа, запускаемая как часть операционной системы. Гарантирует работоспособность требуемых системе и пользователю ресурсов.

**Сетевые протоколы (network protocols)** — набор правил (на физическом уровне), определяющих правила передачи, обработки и интерпретации сообщений между компьютерами.

**Сеть (World Wide Web, WWW)** — всемирная коллекция отформатированных особым образом и соединенных документов, известных как *Web-страницы*.

**Сетка класса (class lattice)** — иерархия классов может называться сеткой классов, если у содержащихся в ней классов есть многочисленные родительские классы.

**Сильный ("толстый") клиент (fat client)** — клиент, несущий относительно большую по сравнению с сервером пропорцию нагрузки по обработке данных. Сильные клиенты всегда работают со слабыми серверами и наоборот, слабые клиенты всегда работают с сильными серверами.

**Сильный ("толстый") сервер (fat server)** — сервер, который несет относительно большую по сравнению с клиентом пропорцию нагрузки по обработке данных. Сильные серверы всегда работают со слабыми клиентами и наоборот, слабые серверы всегда работают с сильными клиентами.

**Синоним (synonym)** — использование различных имен для определения одного и того же объекта, такого как сущность, атрибут или связь. Как правило, их необходимо избегать. *См. также омоним.*

**Система без сохранения состояния (stateless system)** — характеризуется тем, что в любое данное время Web-сервер не знает статус любого из клиентов, работающих с ним. WWW не резервирует память для поддержки открытого "состояния" связи между клиентом и сервером.

**Система принятия решений (Decision Support System, DSS)** — компьютеризованные средства, специально приспособленные для отыскания и отображения данных, отвечающих коммерческим проблемам и запросам. Информация предназначена для содействия руководителям в решении таких проблем, как финансирование, страхование, здравоохранение, банковское дело, торговля и производство. Основная функция *С. П. П.* состоит в извлечении и фильтрации внешних и оперативных данных в склад данных для дальнейшего использования в запросах конечного пользователя.

**Система базы данных (database system)** — совокупность компонентов, которые определяют и регулируют сбор, хранение, управление и использование данных в среде базы данных.

**Система управления базой данных, СУБД (Database Management System, DBMS)** — программное обеспечение — посредник между пользователем и базой данных. *СУБД* преобразовывает запросы пользователя в компьютерный код, который необходим для их выполнения. *СУБД* помогает пользователю управлять данными, хранимыми в базе данных.

**Система управления распределенной базой данных, СУРБД (distributed database management system)** — система управления, которая поддерживает базу данных, распределенную по нескольким узлам; управляет хранением и обработкой логически связанных данных через взаимосвязанные компьютерные системы, в которых данные и их обработка также распределены по нескольким узлам.

**Система управления реляционной базой данных, РСУБД (relational database management system, RDBMS)** — коллекция программ, управляющих комплексной информацией реляционной базы данных. Программное обеспечение *PCYБД* преобразовывает логические запросы пользователя в команды, которые физически находят и извлекают запрошенные данные. Хорошая *PCYБД* также создает и поддерживает *словарь данных (системный каталог)* для помощи в обеспечении безопасности данных, целостности данных, параллельного доступа, легкого доступа и системного администрирования данных в базе данных через язык запросов (SQL) и приложения.

**Системный администратор (systems administrator, SYSADM)** — лицо, ответственное за координирование действий по обработке информации. Системный администратор также управляет использованием программного обеспечения базы данных в техническом отделе; запрашивает и оценивает разработки базы данных, координирует развитие приложений, основанных на источнике данных, назначает право управления базой данных определенным пользователям. Системный администратор также координирует всех администраторов баз данных при операциях со многими базами данных.

**Системный каталог (system catalog)** — детализированная система словарей данных, которая описывает все объекты в базе данных.

**Склад данных (data store)** — компонент системы, обеспечивающий принятие решений, который действует как база данных для хранения коммерческой информации и данных модели предприятия. Данные в архиве хранятся для того, чтобы конечный пользователь модели предприятия мог получать доступ к ним с помощью инструментов создания запросов.

**Слабая (неидентифицируемая) связь (weak (non-identifying) relationship)** — когда одна сущность жизненно не зависит от другой сущности; с позиции разработки базы данных это имеет место в том случае, если первичный ключ связанной сущности не содержит компонент первичного ключа родительской сущности.

**Слабая сущность (weak entity)** — сущность, которая не может существовать сама по себе. Например, ИЖДИВЕНЦУ требуется существование РАБОТНИКА. Слабая сущность наследует как минимум часть ключа от других сущностей.

**Слабый ("тонкий") клиент (thin client)** — клиент, несущий меньшую относительно сервера нагрузку по обработке данных. Слабые клиенты всегда работают с сильными серверами и наоборот.

**Слабый ("тонкий") сервер (thin server)** — сервер, несущий меньшую относительно клиента нагрузку по обработке данных. Слабые серверы всегда работают с сильными клиентами и наоборот.

**Словарь данных (data dictionary)** — компонент СУБД, в котором хранятся метаданные — данные о данных. Таким образом, словарь данных хранит как определения данных, так и их характеристики и связи. Словарь данных может также включать данные, которые находятся за пределами СУБД.

**Словарь распределенных данных (distributed data dictionary, DDD)** — см. *каталог распределенных данных*.

**Сложный объект (complex object)** — объект, составленный несколькими разными объектами со сложными связями. См. также *абстрактные типы данных*.

**Согласованное состояние базы данных (consistent database state)** — состояние базы данных, в котором соблюдаются все ограничения целостности данных.

**Составная сущность (composite entity)** — сущность, разработанная для того, чтобы трансформировать связь М:Н ("многие-ко-многим") в две связи 1:М ("один-ко-многим"). Первичный ключ составной сущности объединяет, как минимум, первичные ключи сущностей, с которыми он связан.

**Составной атрибут (composite attribute)** — атрибут, который может быть впоследствии разделен на дополнительные осмысленные атрибуты. Например: телефонный номер А (615-898-2368) может быть разделен на трехзначный междугородный телефонный код (615), код зоны (898) и четырехзначный код (2368).

**Составной ключ (composite key)** — ключ, состоящий из нескольких атрибутов, который можно разбить на меньшие составные части.

**Составной объект (composite object)** — объект, содержащий как минимум один многозначный атрибут и не содержащий атрибутов, соответствующих другим объектам.

**Состояние объекта (object state)** — набор значений, которые атрибуты объекта имеют в данное время.

**Средства добычи данных (data-mining tools)** — средства, анализирующие данные, обнаруживающие проблемы или возможности, скрытые в связях данных, формирующие компьютерные модели, основанные на добытых ими сведениях, а затем использующие модели для предсказания поведения бизнеса, требуя минимального вмешательства конечного пользователя. Средства добычи данных являются основой для получения знаний.

**Статическая Web-страница (static Web pages)** — Web-страница, которая не слишком изменяется или вообще не меняется во времени.

**Структурная зависимость (structural dependence)** — характеристика данных, которая при изменении в схеме базы данных приводит к нарушению доступа к данным, требуя таким образом изменений во всех программах доступа.

**Структурная независимость (structural independence)** — характеристика данных, которая при изменении в схеме базы данных, не приводит к нарушению доступа к данным.

**Суперключ (superkey)** — см. *ключ*.

**Сущность (entity)** — нечто, о чем вы желаете сохранить информацию. Обычно это человек, место, вещь, понятие или событие. См. также *атрибут*.

**Схема "звезда" (star schema)** — техника моделирования данных, используемая для отображения данных многовариантной поддержки принятия решений в реляционной базе данных с целью предоставления улучшенного анализа данных.

**Схема (schema)** — описание организации базы данных с точки зрения администратора базы данных. Включает в себя имя базы данных, характеристики и такие компоненты, как типы записей, сегменты, атрибуты и т. д.

**Схема XML (eXtensible Markup Language schema, XML schema)** — улучшенный язык определения данных, используемый для описания структуры (элементов, типов данных, типов связей, диапазонов, значений по умолчанию) XML-документов с данными.

**Сценарий (script)** — небольшая программа, содержащая команды, написанные на некотором интерпретируемом языке программирования.

**Сцепленность (cohesivity)** — сила связи между компонентами модуля. Сцепленность модуля должна быть высокой.

**Таблица (table)** — виртуальная матрица, включающая в себя пересекающиеся строки (сущности) и столбцы (атрибуты), которая представляет собой множество сущностей в реляционной модели. Также называется отношением.

**Таблица фактов (fact table)** — расположена в центре схемы "звезда" и содержит факты, соединенные и систематизированные согласно их обычным измерениям.

**Тег (tag)** — управляющая команда, вставляемая в документ для определения способа его форматирования. Теги используются в серверных языках разметки для представления данных.

**Тернарная связь (ternary relationship)** — связь логических объектов, описывающая взаимоотношения трех сущностей. Например, СПОНСОР жертвует деньги ФОНДУ, из которого деньги передаются ПОЛУЧАТЕЛЮ. (Заметьте, что между СПОНСОРОМ и ПОЛУЧАТЕЛЕМ прямой зависимости нет.)

**Топология "звезда" (star topology)** — сетевая топология, в которой компьютеры соединены один с другим в виде звезды через центральный компьютер или сетевой концентратор. В такой топологии, как и в топологии "кольцо", добавление или потеря компьютера не имеет негативного влияния на другие компьютеры.

**Топология "кольцо" (ring topology)** — топология сети, при которой компьютеры соединены один с другим посредством кабельной схемы, которая, согласно названию, напоминает кольцо. Такая топология более гибка, чем шинная топология, потому что добавление или потеря компьютера не имеет негативного влияния на другие компьютеры сети.

**Транзакционная СУБД (transactional DBMS)** — система управления базой данных, которая поддерживает базу данных, первоначально разработанную для поддержки транзакций "немедленного реагирования" или ограниченных по времени транзакций.

**Транзакция (transaction)** — последовательность операций базы данных (один или более запросов к базе данных), которые обращаются к базе данных. Транзакция — это *логическая единица работы*, т. е. она должна быть *полностью* выполнена или отменена без промежуточных положений. Все транзакции должны иметь следующие свойства: 1) *атомарность* требует, чтобы все операции (части) транзакции были завершены, иначе транзакция отменяется; 2) *устойчивость* гарантирует, что если тран-



закция завершена, то база данных достигает непротиворечивого состояния; это состояние не может быть утрачено даже в случае сбоя системы; 3) *сериализуемость* гарантирует, что выбранный порядок операций транзакций создаст завершающее положение базы данных, которое было бы достигнуто, если бы транзакции были выполнены последовательно; 4) *изолированность* гарантирует, что данные, используемые во время выполнения транзакции, не будут использованы второй транзакцией до того, как первая не будет закончена.

**Транзитивная зависимость (transitive dependency)** — состояние, при котором атрибут зависит от другого атрибута, не являющегося частью первичного ключа.

**Третья нормальная форма, 3НФ (third normal form, 3NF)** — третий этап процесса нормализации, когда в таблице, приведенной к 2НФ, устраняются транзитивные зависимости (когда один неключевой атрибут функционально зависит от другого неключевого атрибута).

**Триггер (trigger)** — процедурный SQL-код, который автоматически запускается системой управления реляционной базой данных в случае обращения к данным.

**Тупик (deadlock)** — состояние, возникающее, когда две или более транзакций ожидают снятия их блокировок с заблокированной до этого ячейки данных.

**Углубленное исследование (drill-down)** — декомпозиция объекта на более мелкие составляющие (т. е. на данные более низкого уровня агрегации). Используется, в основном, в системах, обеспечивающих принятие решений, для сосредоточения на особых географических районах, типах коммерции и т. д.

**Удаленная транзакция (remote transaction)** — свойство СУРБД, позволяющее транзакции (сформированной несколькими запросами) получать доступ к данным в едином удаленном процессоре данных. *См. также удаленный запрос.*

**Удаленный запрос (remote request)** — свойство СУРБД, позволяющее одной SQL-операции иметь доступ к данным в едином удаленном процессоре данных. *См. также удаленная транзакция.*

**Указатель (pointer)** — справочное средство, которое "указывает" на местоположение данных внутри компьютерной среды хранения данных.

**Унарная связь (unary relationship)** — логическое понятие, используемое для описания связи *внутри* сущности.

**Университетская сеть (Campus-Wide Network, CWN)** — типичная сеть колледжа или университета, в которой здания, содержащие локальные сети (LAN), обычно соединены основной, магистральной сетью.

**Унифицированный указатель информационного ресурса (Uniform Resource Locator, URL)** — стандартизованная строка символов, указывающая местонахождение документа в Интернете.

**Управление авторизацией** — все действия, направленные на усиление безопасности, конфиденциальности и неприкосновенности данных в базе данных.

**Управление в чрезвычайных ситуациях (disaster management)** — совокупность действий администратора, направленных на обеспечение безопасности работоспособности данных после физического повреждения или полного сбоя базы данных.

**Управление данными (data management)** — процесс, который ставит своей целью сбор, хранение и поиск данных. Обычно функции управления данными включают в себя суммирование, удаление, модификацию и листинг.

**Участники (participants)** — термин отношений логических объектов (ER-термин), используемый для обозначения сущностей, участвующих в связи. Например, ПРОФЕССОР читает КУРС. Связь "читает" основана на участниках ПРОФЕССОР и КУРС.

**Файл (file)** — именованный набор связанных записей.

**Факты (facts)** — числовые значения, представляющие собой определенные коммерческие показатели или показатели деятельности. Обычно измеряются в единицах, стоимостях, ценах и показателях доходов при описании коммерческих операций.

**Физический формат данных (data physical format)** — структурные (физические) характеристики данных "с точки зрения" компьютера. *См. также логический формат данных.*

**Физическое проектирование (physical design)** — этап разработки базы данных, на котором учитываются характеристики хранения и доступа к данным базы данных. Как только эти характеристики увязываются с функциями устройств, аппаратными средствами, методами доступа к данным на уровне системы и выбранной СУБД, проектирование на физическом уровне становится и аппаратно зависимым, и зависимым от программного обеспечения.

**Фильтрация данных (data filtering)** — *см. извлечение данных.*

**Флаг (flag)** — специальный код, применяемый разработчиками для предотвращения пустых значений путем привлечения внимания к отсутствию значения в таблице.

**Фрагментация данных (data fragmentation)** — процесс, разбивающий одну таблицу на два или более фрагментов. Фрагменты базы данных рассматриваются как единая логическая база данных. *См. также распределенная база данных.*

**Функциональная зависимость (functional dependence)** — в пределах одной связи R атрибут В функционально зависит от атрибута А тогда и только тогда, когда значение атрибута А определяет только одно значение атрибута В. Связь "В зависит от А" эквивалентна связи "А определяет В" и записывается как  $A \rightarrow B$ .

**Хранимая процедура (stored procedure)** — 1) именованная группа процедурных и SQL-операторов; 2) хранимый на сервере SQL-код или программа на другом характерном для СУБД процедурном языке.

**Хранимая функция (stored function)** — именованная группа процедурных и SQL-операторов, возвращающих значение, обозначаемое в программном коде оператором RETURN.

**Целостность данных (data integrity)** — состояние, при котором выбранные данные всегда выдают одни и те же результаты. Целостность данных обязательна для баз данных.

**Целостность на уровне ссылок (referential integrity)** — состояние, при котором внешний ключ зависимой таблицы должен либо иметь неопределенное значение, либо совпадать со значением в связанной таблице. Даже если у атрибута не будет отвечающего ему атрибута, то все равно будет невозможно получить неверную запись.

**Целостность на уровне сущностей (entity integrity)** — отсутствие неопределенных (null) значений в первичном ключе. Гарантия того, что у каждой сущности будет уникальное значение.

**Централизованная разработка** — процесс, в котором единая концепция разработки создается в соответствии с требованиями к базе данных организации. Обычно применяется в том случае, когда компоненты данных состоят из сравнительно малого числа объектов и процедур.

**Централизованная стратегия распределения данных** — стратегия распределения данных, основанная на хранении всей базы данных в одном месте.

**Централизованная СУБД** — система управления базой данных, которая поддерживает работу базы данных, расположенной в одном месте.

**Частичная зависимость (partial dependency)** — состояние, при котором атрибут зависит только от части (подмножества) первичного ключа.

**Четвертая нормальная форма, 4НФ (fourth normal form, 4NF)** — таблица приведена к 4НФ, если она приведена к 3НФ и не содержит многозначных атрибутов.

**Шинная топология** — сетевая топология, требующая, чтобы все компьютеры были подключены к магистральному кабелю сети. Обычное использование коаксиального кабеля несет в себе опасность: сбой даже одного компьютера может повлечь выход из строя целого сегмента сети.

**Шлюз (gateway)** — тип промежуточного программного обеспечения, предназначенного для преобразования запросов клиента в соответствующие протоколы, требуемые для доступа к особым сервисам. Такое программное обеспечение устраняет требование использовать одни и те же протоколы как к серверам, так и к клиентам.

**Экземпляр класса (class instance)** — каждый индивидуальный объект, содержащийся в классе. Каждый экземпляр класса должен иметь общую со всеми другими экземплярами класса структуру и реагировать на то же сообщение, что и другие экземпляры класса, если все они находятся в одном классе. Также называется *экземпляром объекта*.

**Экземпляр объекта (object instance)** — каждый отдельно взятый объект, принадлежащий данному классу.

**Экземпляр сущности (entity instance)** — термин, используемый в моделировании отношений логических объектов (ER-моделировании) для ссылки на определенную строку таблицы.

**Экранированная витая пара (shielded twisted-pair, STP)** — пара кабелей, закрученных один вокруг другого, у которых есть внешняя защитная изоляция для предотвращения электромагнитной интерференции.

**Экстранет (Extranet)** — особая сеть предприятия, доступная определенным лицам (и вне данного предприятия), имеющим отношение к формированию корпоративной цепочки нарастания стоимости. Э. позволяет автоматизировать документооборот путем предоставления клиентам возможности размещать свои заказы, обрабатывать собственные платежи или выполнять иные задачи, обычно исполняемые сотрудником организации внутри нее.

**Электронная коммерция, е-коммерция (e-commerce)** — использование электронных технологий на компьютерной основе для продвижения на рынке новых продуктов,

услуг или решений и/или для поддержки или увеличения существующих коммерческих операций; расширяет возможности бизнеса с помощью предоставления доступа к информации широким массам пользователей.

**Электронная почта, E-mail (electronic mail)** — графические и текстовые сообщения, посылаемые другим конечным пользователям выбранного компьютера, подключенного к той же сети. Широко используется из-за низкой стоимости и возможности обмениваться информацией с одним или многими адресатами.

**Электронный бумажник (electronic wallet)** — эквивалент реального бумажника, но предназначенный для использования на электронном рынке. В нем хранятся кредитные карточки пользователя, электронная наличность и другая персональная информация, такая как адреса доставки, телефонные номера и адреса электронной почты.

**Язык запросов (query language)** — непроцедурный язык, позволяющий пользователю определить, *что* должно быть сделано без определения того, *как* это должно быть сделано. Пример языка запросов — язык SQL.

**Язык манипулирования данными (Data Manipulation Language, DML)** — язык (набор команд), позволяющий конечному пользователю оперировать данными базы данных.

**Язык определения данных (Data Definition Language, DDL)** — язык, позволяющий администратору базы данных определить компоненты структуры, схемы и подсхемы.

**Язык программирования третьего поколения (third-generation language, 3GL)** — язык, требующий от программиста определить то, *что* должно быть сделано, и то, *как* это должно быть сделано. Примеры таких языков: Cobol, Basic и Fortran.

**Язык программирования четвертого поколения (fourth-generation language, 4GL)** — непроцедурный язык, такой как SQL, который требует от пользователя определить только то, *что* должно быть сделано, а не *как* это надо сделать.

**Язык структурированных запросов (Structured Query Language, SQL)** — мощный и гибкий язык программного обеспечения реляционных баз данных, состоящий из команд, которые помогают пользователям создавать структуры баз данных и таблиц, производить различные манипуляции с данными, управлять данными и запрашивать у базы данных полезную информацию.

# Предметный указатель

## A

access plan 951  
ActiveX 873  
ANSI 271  
API 720, 869  
ASCII 295  
audit log 944

## B, C

back-end application 703  
CASE 401, 959  
CGI 868  
Common Gateway Interface 868  
communications middleware 703  
concentrator 990  
concurrent backup 946  
CWN 989

## D

DA 934  
data mining 800  
data store 749  
datafile 969  
DDL 43  
DML 43  
DSO 945  
DTD 850

## E, F

ER-модель 62  
Extended Relational Model 684  
fat client 697  
fat server 697  
front-end application 703  
full backup 946

## H, I

hub 989  
IDAPI 706  
IDEX 192  
incremental backup 946  
Inversion Entry 170  
IPC 713  
IPX 706  
IRM 934

## J, L, M, N

Java 873  
JavaScript 873  
LAN 988  
MAN 989  
MIS, отдел информационных систем 701  
MOLAP 780  
network backbone 989

## O, P, R

ODBC 706  
OID 641  
OLAP 765  
plug-in 873  
privacy 928  
production databases 24  
profile 973  
ROLAP 776  
role 973  
router 989

## S

SDLC 397  
SET 832  
SGML 848

## SQL, команды:

ALTER 304  
AND 299  
AVG 321  
BETWEEN 301  
COMMIT 289  
CREATE DOMAIN 283  
CREATE INDEX 326  
CREATE TABLE 280  
CREATE VIEW 325  
DELETE 291  
DISTINCT 316  
DROP 312  
EXISTS 304  
GROUP BY 321  
HAVING 323  
IN 303  
INSERT 287  
IS NULL 301  
LIKE 302  
MAX 319  
MIN 319  
NOT 300  
OR 299  
ORDER BY 313  
ROLLBACK 291  
SELECT 289  
SUM 320

UNIQUE 326

UPDATE 290, 306

SQL\*Net 721

stateless system 891

SYSADM 934

**T**

tablespace 969

TCP/IP 706

thin client 697

thin server 697

token 987

transactional databases 24

**U, V, W, X**

user 973

value chain 825

VBScript 874

VLDB 759

WAN 989

XML 847

дочерний элемент 849

корневой элемент 848

XSD 852

XSL 853

**A**

Абстрактные типы данных 654

Автоматизированное проектирование  
систем 401Автоматическая оптимизация  
запроса 617

Администратор:

    базы данных 39, 932  
    данных 934

Активный словарь данных 956

Аномалии:

    включения 36  
    модификации 36  
    удаления 36

Архитектура:

информационной системы 962

"клиент/сервер" 603, 703

Ассоциативный объект 655

Ассоциация 153

Атомарность 561

транзакций 580

Атрибут 92, 641

сущности 63

**Б**

База данных 21

большого размера 759

поддержки решений 41

предприятия 41, 929  
рабочей группы 41  
Базовые типы данных 642  
Банк данных 42  
Бездисковые рабочие станции 430  
Безопасность 928  
Бизнес-правила 169, 411  
Бинарная связь 179  
Блокировка 570  
    на уровне базы данных 571  
    на уровне строки 573  
Булева алгебра 300  
Буферы базы данных 582

## В

Вертикальная фрагментация 619  
"витая пара" 985  
Витрина данных 764  
Вложенный запрос 320  
Внешнее соединение 110  
Внешний ключ 101  
Внешний цикл 320  
Внешняя модель 157  
Внутренний цикл 320  
Внутренняя модель 156  
Восстановление базы данных 580  
Восходящее проектирование 433  
Временной диапазон 751  
Встроенный SQL 337  
Вторая нормальная форма, 2НФ 236  
Вторичный ключ 101  
Выборочное копирование 581

## Г

Гетерогенная система распределенных баз данных 604  
Гиперкуб 780  
Глобальная сеть 989  
Гомогенная система распределенных баз данных 603  
Горизонтальная фрагментация 619  
Городская сеть 989  
Группа хранения 425

## Д

Дамп базы данных 946  
Двоичная блокировка 574  
Двухзвенная клиент/серверная система 697  
Декартово произведение 105  
Денормализация 227  
Детерминант 243  
Децентрализованное проектирование 434  
Диаграмма:  
    "сущность-связь" 62  
    зависимостей 233  
Динамически подключаемая библиотека, DLL 869  
Динамический:  
    алгоритм 617  
    режим создания статистики 618  
Динамичная Web-страница 824  
Дисковая страница 573  
Дисковый блок 573  
Добыча данных 800  
Договорные типы данных 642  
Долговечность 561  
Домен 162, 642  
    атрибута 93

## Е, Ж

Единичное наследование 650  
е-коммерция 698, 815  
Естественное соединение 107  
Жизненный цикл:  
    базы данных 402  
    разработки систем 397  
Журнал транзакций 563

## З

Зависимая таблица 355  
Зависимость:  
    от существования 169  
    по данным 33  
Запрос 22  
    к базе данных 558  
Защита с помощью паролей 429

## И

- Идентификатор:
  - времени 761
  - объекта 641
- Идентифицируемая связь 171
- Иерархическая:
  - последовательность 49
  - структура 48
- Иерархический маршрут 49
- Иерархия:
  - классов 68, 648
  - обобщенных представлений 188
- Извлечение и фильтрация данных 749
- Изолированность 561
- Индексный ключ 129
- Инкапсуляция 645
- Инкрементальное копирование 946
- Инспектор безопасности базы данных 945
- Интегрированный словарь данных 956
- Интерфейс прикладного программирования 720, 869
- Интерфейсное приложение 703
- Инtranет 695, 825
- Информационная:
  - система 395
  - техника 962
- Информационное хранилище 24
- Информационный инжиниринг 962
- Инфотехника 962
- Исследование "вдоль и поперек" 787
- Итеративный процесс 195

## К

- Каскадное упорядочивание 314
- Каталог распределенных данных 608
- Класс 67
  - пересечения 667
- Кластерная таблица 541
- Клиент 696
- Ключ 96
- Ключевой атрибут 98, 235
- Компьютеризированное проектирование систем 191
- Контролируемая избыточность 100
- Контроль версий 671

- Контрольная точка базы данных 582
- Контрольный журнал 429, 944
- Конфиденциальность 928
- Концентратор 989
- Концептуальная модель 45, 153
- Координатор 615
- Корень 48
- Кортеж 92
- Куб данных 780
- Кэш куба 781

## Л

- Логический формат данных 33
- Логическое проектирование 25, 423
- Локальная сеть 988

## М

- Магистральная сеть 989
- Маркер 987
- Маршрутизатор 989
- Межобъектная связь 662
- Менеджер:
  - блокировок 571
  - данных 599
  - информационных ресурсов 934
  - транзакций 599
- Метаданные 21
- Метасимвол 289
- Метка времени 579
- Метод 643
  - класса 68, 648
- Метрика 784
- Многозначный атрибут 164
  - объекта 643
- Многомерная система:
  - оперативного анализа данных 780
  - управления базами данных 780
- Многопользовательская СУБД 41
- Множественное наследование 650
- Модель 45
  - "птичья лапка" 192
  - "сущность-связь" 62
  - Rein85 192
  - базы данных 45
  - реализации 45
  - Чена 191



Монотонность 579  
Мост 989  
Мощность связи 168

## Н

Набор сущностей 63, 91  
Навигационная система 51  
Накопитель 990  
Наследование 68, 649  
Настольная база данных 41  
Независимость от существования 169  
Неидентифицируемая связь 170  
Немедленное обновление 583  
Необязательное участие 173  
Необязательность 173  
Непересекающиеся подтипы 190  
Нерегламентированный запрос 22  
Неупорядоченные данные 20  
Нисходящее проектирование 432  
Нормализация 227  
Нормальная форма Бойса—Кодда, БКНФ 243

## О

Область таблиц 425  
Обновляемое представление 334  
Общий шлюзовой интерфейс 868  
Объект 640  
    базы данных 971  
    модели SDM 66  
Объект-набор 643  
Объектная модель данных 656  
Объектно-реляционная модель 684  
Объектно-ориентированная модель 66  
    Данных 656  
Объектно-ориентированная СУБД, ООСУБД 675  
Объектно-ориентированное  
    программирование 639  
Объектно-ориентированный язык:  
    запросов 674  
    программирования 639  
Объектно-реляционная СУБД 71  
Обязательное участие 174  
Однозначный атрибут 164  
    объекта 643

Однопользовательская СУБД 41  
Операционная (рабочая) база данных 24  
Определяемость 97  
Опрос состояния 645  
Организация хранилища данных 764  
Отдел информационных систем,  
    MIS 931  
    обработки данных 931  
    электронной обработки данных 931  
Отложенная запись 583  
Отложенное обновление 583

## П

Пакетная процедура обновления 333  
Параллельное резервное копиро-  
    вание 946  
Пассивный словарь данных 956  
Первая нормальная форма, 1НФ 235  
Первичный:  
    атрибут 235  
    ключ 96  
Переменная экземпляра 641  
Пересекающиеся подтипы 190  
Периодичность 796  
План доступа 951  
Планировщик 569  
Повторяющиеся группы 232  
Поддержка решений 748  
Подкласс 649  
Подключаемый модуль 873  
Позднее связывание 669  
Полиморфизм 652  
Политики 941  
Полная резервная копия 581, 946  
Полная функциональная зависимость 98  
Полностью реплицированная база  
    данных 624  
Пользователь 973  
Последовательная обработка записей 674  
Потенциальный ключ 99  
ППО Web-БД 867  
Права доступа 429  
Правило взаимной  
    непротиворечивости 624  
Представление 325

**Прозрачность:**

- гетерогенности 606
- ошибок 605
- производительности 605
- распределения 605
- реплики 617
- транзакций 605, 609
- фрагментации 606

**Прозрачные свойства СУРБД 605****Производный атрибут 166****Промежуточная сущность 121, 185****Промежуточное программное обеспечение баз данных для Web 867****Промежуточное программное обеспечение передачи данных 703****Простой:**

- атрибут 164
- объект 655

**Пространство:**

- объектов 658
- таблиц 969

**Противоречивость данных 35****Протокол 647****DO-UNDO-REDO 615****TCP/IP 718****двухфазного подтверждения транзакции 613****связи между процессами 713****СУРБД 599****упреждающей записи 582, 615****Профиль 973****Процедурный sql 337****Процедуры 941****Процессор:**

- приложений 599
- транзакций 599

**Прямой порядок обхода 49****Псевдоним 298, 329****Р****Рабочая СУБД 41****Разбиение таблицы 796****Размещение данных 618****Разработка:**

- базы данных 397
- систем 395

**Разряженность 782****Раннее связывание 669****Распределение данных 625****Распределенная:**

- база данных 594
- обработка 594
- СУБД 41
- транзакция 610

**Распределенный запрос 611****Редиректор 603****Резервный журнал транзакций 582****Рекурсивная связь 182****Реляционная:**

- алгебра 104
- модель базы данных 91
- система оперативного анализа данных 776

**Репитер 989****Реплика 617****Репликация 796**

- данных 618, 623

**Реплицированное размещение данных 625****Родительская таблица 355****Роль 973****С****Самостоятельный словарь данных 956****Связность 168****Связываемость модулей 421****Связывание данных 856****Связь 153, 167****Сегмент сети 987****Секретность 829, 928****Секционированное размещение данных 625****Сервер 696**

- Web-приложений 874
- базы данных 594

**Серверное:**

- приложение 703
- расширение 867

**Сервис Windows NT 967****Сериализуемость 561****Сетевой стандарт:**

- Ethernet 988
- Token Ring 988

Сетевой транслятор 720  
Сетевые интерфейсные карты 989  
Сеть классов 648  
Сильная связь 171  
Сильный (толстый) клиент 697  
Сильный (толстый) сервер 697  
Синоним 114  
Система:  
    автоматизированной разработки программ 959  
    оперативного анализа данных 765  
    поддержки решений 41  
    управления базой данных 22  
    управления распределенной базой данных 590  
    управления реляционной базой данных 57  
    файлов 19  
Системный:  
    администратор 39, 934  
    анализ 395  
    каталог 112  
Сквозная запись 583  
Склад данных 749  
Слабая:  
    связь 170  
    сущность 177  
Слабый (тонкий):  
    клиент 697  
    сервер 697  
Словарь:  
    данных 42, 112  
    информационных ресурсов 957  
    распределенных данных 608  
Сложный объект 654, 655  
Смешанная фрагментация 619  
Смешанный объект 655  
Совместимость по объединению 105  
Совместное использование адресуемых объектов 660  
Сообщение 644  
Составная сущность 121, 185  
Составной:  
    атрибут 164  
    ключ 98, 162  
    объект 655

Состояние объекта 643  
Стандарт 724, 941  
Статистические алгоритмы оптимизации запроса 618  
Статический алгоритм 617  
Статичная web-страница 824  
Степень:  
    детализации блокировок 571  
    связи 179  
Столбцы соединения 107  
Страница 573  
Структурированные данные 20  
Структурная:  
    зависимость 33  
    независимость 59  
СУБД 22  
Субординатор 615  
Суперкласс 649  
Суперключ 98  
СУРБД 590  
Сущность 24  
Схема 53  
    Oracle 972  
    "звезда" 777  
    базы данных 275  
    глобального распределения 608  
    объектов 658  
Сценарий ColdFusion 876  
Сцепленность модуля 421

## Т

Таблица 57, 92  
    измерений 784  
    фактов 784  
Тернарная связь 179  
Тета-соединение 110  
Точка:  
    блокирования 576  
    завершения 583  
Транзакционная:  
    база данных 24  
    СУБД 41  
Транзакция 558  
Транзитивная зависимость 234  
Транслятор баз данных 720

Третья нормальная форма, 3НФ 238  
Трехзвенная клиент/серверная  
система 697  
Триггер 338  
Тупик 578

## У

Удаленная транзакция 609  
Удаленный запрос 609  
Унарная связь 179  
Университетская сеть 989  
Уникальность 579  
фрагмента 607  
Управление данными 21  
параллельным выполнением 419  
транзакций 564  
чрезвычайными ситуациями 946  
Уровень коммуникаций 703  
Устойчивое состояние базы данных 559  
Участник 167

## Ф

Фаза:  
записи 580  
подтверждения 580  
роста 576  
сжатия 577  
чтения 580  
Файл данных 969  
Факт 784  
Физическая:  
безопасность 429  
модель 160  
Физический формат данных 33  
Физическое проектирование 424  
Фильтрация данных 757  
Флаг 103  
Фрагмент базы данных 594  
Фрагментация данных 618  
Фрейм 716  
Функции администрирования баз  
данных 932  
Функциональная зависимость 97

## Х

Хранилище данных 42, 760  
Хранимая:  
процедура 345, 733  
функция 348

## Ц

Целостность:  
данных 35  
на уровне ссылки 101  
на уровне сущности 99  
Централизованная СУБД 41  
Централизованное:  
проектирование 434  
размещение данных 625

## Ч, Ш

Частичная зависимость 234  
Четвертая нормальная форма, 4НФ 252  
Шифрование данных 430  
Шлюз 723

## Э

Эквисоединение 110  
Экземпляр:  
базы данных 968  
класса 646  
объекта 646  
сущности 161  
Экстранет 695, 825  
Электронная коммерция 815  
Электронные деньги 831  
Электронный бумажник 833

## Я

Язык:  
запросов 43  
манипулирования данными 43  
определения данных 43  
третьего поколения (3GL) 31  
четвертого поколения (4GL) 60