В. К. ВОЛК

БАЗЫ ДАННЫХ
ПРОЕКТИРОВАНИЕ, ПРОГРАММИРОВАНИЕ, УПРАВЛЕНИЕ И АДМИНИСТРИРОВАНИЕ

Учебник

САНКТ-ПЕТЕРБУРГ
МОСКВА
КРАСНОДАР
2020

УДК 004.65
ББК 32.973-018.2я73

В 67 **Волк В. К.** Базы данных. Проектирование, программирование, управление и администрирование : учебник / В. К. Волк. — Санкт-Петербург : Лань, 2020. — 244 с. : ил. — (Учебники для вузов. Специальная литература). — Текст : непосредственный.

ISBN 978-5-8114-4189-1

Учебник посвящен базам данных — одному из направлений IT-индустрии, в рамках которого традиционно рассматриваются технологии надежного хранения больших объемов информации, ее эффективного поиска и извлечения по запросам потребителей. Материал, представленный в учебнике, охватывает основные фазы жизненного цикла базы данных: последовательно обсуждаются основные концепции и проблематика баз данных, рассматриваются процессы их проектирования, программирования и управления, а также технологии их администрирования с целью достижения высокой производительности доступа к данным и обеспечения требуемого уровня информационной безопасности. База данных рассматривается как информационная модель предметной области, а ее разработка — как многоэтапный процесс последовательного преобразования концептуальной модели в логическую (реляционную) модель данных на стадии проектирования, последующей программной реализации логической модели средствами языка SQL и настройки параметров физической модели в процессе эксплуатации базы данных.

Основное внимание в учебнике уделено прикладным аспектам технологий баз данных, изложение материала сопровождается многочисленными примерами и листингами программного SQL-кода. Учебник содержит практикум по проектированию, программированию и администрированию баз данных, включающий примеры и практические задания для самостоятельного выполнения.

Предназначен для студентов IT-специальностей и может быть использован преподавателями при подготовке лекционных курсов, проведении практических и лабораторных занятий, курсовом проектировании.

УДК 004.65
ББК 32.973-018.2я73

Обложка
П. И. ПОЛЯКОВА

ПРЕДИСЛОВИЕ

Высокая ценность информации признавалась во все времена, но только во второй половине XX в., когда появилась возможность эффективного управления действительно большими объемами информации, она стала важнейшим стратегическим ресурсом, обслуживание которого потребовало создания специализированных программно-технических средств — автоматизированных информационных систем (АИС).

АИС обеспечивают надежное хранение и оперативное обновление информации, а также ее поиск, извлечение и аналитическую обработку по запросам потребителей. При всем разнообразии архитектур, решаемых задач и условий использования АИС в структуре их программного обеспечения (ПО) явно выделяют два относительно автономных компонента: подсистему хранения данных и подсистему обработки информации. Основу подсистемы хранения составляют *база данных* (БД) — программно реализованная информационная модель предметной области АИС, управляемая специализированной программной системой (СУБД), и *подсистема обработки информации*, включающая множество прикладных программ, получающих доступ к СУБД для чтения и/или модификации информации, хранимой в базе данных.

АИС относятся к категории социотехнических систем, важнейшим компонентом которых, наряду с аппаратным и программным обеспечением, являются пользователи — разработчики, администраторы и конечные пользователи, участвующие в создании или эксплуатации АИС на различных стадиях ее жизненного цикла.

Материал, излагаемый в учебнике, охватывает основные стадии жизненного цикла базы данных и в основном соответствует содержанию учебной дисциплины «Базы данных», изучаемой студентами IT-специальностей. Учебное пособие состоит из пяти относительно автономных частей, каждая из которых ориентирована на определенную категорию пользователей баз данных.

1–3-я части учебника (главы с 1-й по 7-ю) ориентированы на проектировщиков и программистов баз данных, обычно выделяемых в отдельную категорию разработчиков программных систем. Здесь рассматриваются:

– проблемы и пути достижения автономности системы хранения данных в составе программного обеспечения АИС, вводятся понятия *базы данных* и *системы управления базами данных*;

– информационные модели, используемые на различных стадиях жизненного цикла баз данных, дается обзор основных свойств иерархической, сетевой, реляционной и объектно-ориентированной логических моделей данных;

– реляционная модель данных Кодда: ее структурный, целостностный и манипуляционный компоненты, включая реляционную алгебру и реляционное исчисление кортежей (в объеме, минимально необходимом для понимания методов нормализации реляционных баз данных и освоения базовых конструкций языка SQL);

– процедура проектирования базы данных, включающая разработку концептуальной ER-модели предметной области АИС, преобразование ER-модели

в схему реляционной БД и ее последующую нормализацию для улучшения эксплуатационных характеристик;

– программная реализация реляционных БД, рассматриваемая на примере одного из диалектов языка SQL, соответствующего стандарту SQL-89, обсуждаются также элементы процедурного расширения языка, соответствующие стандарту SQL-92.

5-я и 7-я главы учебника — это проектный практикум и практикум по SQL-программированию. Эти главы могут быть использованы как при проведении практических занятий со студентами, так и для самостоятельного освоения соответствующих разделов технологий баз данных.

4-я часть учебника (главы с 8-й по 14-ю) посвящена вопросам управления данными и ориентирована на администраторов баз данных — IT-специалистов, обеспечивающих эффективное функционирование БД в процессе их эксплуатации. В этой части рассматриваются:

– задачи администрирования БД, базовые функции реляционных СУБД и более детально — проблемы многопользовательского доступа к БД и пути их решения средствами управления транзакциями и блокировками;

– низкоуровневая («физическая») модель данных, поддерживаемая на файловом уровне одним из коммерческих серверов баз данных, и инструментальные средства управления этой моделью, предоставляемые администраторам баз данных;

– методы высокопроизводительного доступа к данным, обеспечиваемые индексными структурами, алгоритмы поиска информации, а также инструментальные средства анализа и управления индексами;

– процедура трансляции исходного SQL-кода и оптимизации запросов, инструментальные средства визуализации и анализа процедурных планов их исполнения.

5-я часть учебника (главы с 15-й по 17-ю) ориентирована на специалистов по защите информации, часто выделяемых в отдельную категорию администраторов баз данных. Здесь рассматриваются:

– концепции, задачи и методы защиты информации, хранимой в базах данных;

– типовая архитектура подсистемы информационной безопасности СУБД;

– методы и инструментальные средства разграничения доступа к объектам БД со стороны пользователей.

14-я и 17-я главы учебника — практикум по администрированию баз данных и управлению информационной безопасностью, содержащий практические задания аналитического и технологического характера, выполнение которых позволит студентам более детально изучить низкоуровневую структуру базы данных, реализованные в СУБД алгоритмы управления данными и методы защиты от несанкционированного доступа, а также потребует от них освоения разнообразных технологических инструментов, используемых администраторами баз данных в своей профессиональной деятельности.

Следует отметить прикладной характер учебника, основное внимание в нем уделено технологическим аспектам проектирования, программирования и администрирования баз данных.

Теоретическим вопросам построения баз данных посвящены многочисленные публикации [4, 7, 16, 21, 24–33], среди которых можно выделить курс лекций С. Кузнецова [7], в котором, в частности, более детально рассмотрена реляционная модель данных: реляционная алгебра Кодда, алгебра «А» Дэйта и Дарвена, реляционное исчисление кортежей и доменов, теория нормальных форм отношений. Имеются также учебные издания, авторы которых рассматривают технологии проектирования баз данных и их программной реализации, среди которых можно отметить книги Г. Рикарди [9] и Л. Бейли [1].

В то же время (и это вполне объяснимо) вопросы построения СУБД, поддерживаемые ими низкоуровневые структуры данных, методы и алгоритмы управления, а также технологии администрирования баз данных представлены в основном научными и обзорными статьями в специализированных журналах, технической документацией, публикуемой разработчиками СУБД на своих официальных интернет-ресурсах, а также статьями в многочисленных профессиональных блогах, поддерживаемых администраторами баз данных. За редким исключением (например, книги С. Бобровского [2] и Е. Мамаева [8]) материалы такого рода трудно отнести к категории учебных изданий, их можно рекомендовать студентам, изучающим базы данных, лишь в качестве дополнительных информационных источников.

Учебник не является практическим руководством по администрированию баз данных и, разумеется, не охватывает весь объем трудовых функций, определенных соответствующими профессиональными стандартами. При его подготовке основное внимание было уделено вопросам организации физической модели данных, оптимизации выполнения SQL-запросов и разграничения доступа к данным, то есть решению наименее рутинных задач администрирования, требующих профессиональной компетентности в вопросах функционирования серверов баз данных, организации хранения данных на файловом уровне и методах поиска информации, основанных на использовании индексных структур данных.

При подготовке учебника автор учитывал личный опыт преподавания технологий баз данных студентам старших курсов IT-специальностей Курганского государственного университета. Использование учебника предполагает наличие у студентов базовой подготовки в области программной инженерии: программирование, структуры и типовые алгоритмы обработки данных, модели жизненного цикла ПО, основы языка UML.

Автор выражает благодарность всем студентам, использовавшим предварительный вариант учебника и сделавшим ряд полезных замечаний по его содержанию. Особая благодарность студентам П. Казакову и Д. Стенникову, проделавшим большую работу по анализу структуры системных объектов баз данных, и М. Останину, исследовавшему индексные структуры данных и процедурные планы выполнения SQL-запросов.

ЧАСТЬ 1.
ОСНОВНЫЕ КОНЦЕПЦИИ

ГЛАВА 1. АВТОНОМНОСТЬ БАЗ ДАННЫХ

Автономность подсистемы хранения данных, в основу которой положен принцип независимости данных от обрабатывающих их программ, является существенной особенностью АИС, отличающей их от других программных систем. Отсутствие взаимозависимости программ и данных позволяет, в частности, использовать БД для решения многих прикладных задач, постепенно наращивая функциональность АИС, а также обеспечивает возможность модификации структуры самой БД в соответствии с изменениями моделируемого сегмента предметной области.

На пути к достижению автономности баз данных в составе АИС эти системы прошли ряд этапов в своем эволюционном развитии. В первых компьютерных системах и программный код, и обрабатываемые им данные размещались в едином запоминающем устройстве (вспомним один из базовых принципов фон Неймана, «сегмент кода» и «сегмент данных» в ассемблерном программном коде, пример которого приведен на листинге 1.1), и это накладывало серьезные ограничения на объем обрабатываемой информации.

```
format MZ;                              Исполняемый файл DOS EXE (MZ EXE)
entry code_seg:start;                   Точка входа в программу
stack 1000h;                            Размер стека
segment data_seg;                       Сегмент данных
hello_str db 'Hello!',0dh,0ah,'$';      Выделение байт под строку
segment code_seg;                       Сегмент кода
start:;                                 Точка входа в программу
mov ax,data_seg;                        Инициализация регистра DS
mov ds,ax
mov ah,09h
mov dx,hello_str
int 21h;                                Вывод строки
mov ax,4C00h
int 21h;                                Завершение программы
```

Листинг 1.1
Пример ассемблерного кода программы

Оснащение компьютеров быстродействующими дисковыми накопителями большой емкости позволило снять это ограничение, так как стало возможным хранение больших информационных массивов отдельно от программного кода — в файлах на внешних запоминающих устройствах. При этом, однако, описание структур таких файлов по-прежнему оставалось включенным в программный код, что не избавляло систему от (как минимум) одного существен-

ного недостатка, препятствующего реализации концепции автономности БД в структуре АИС.

Для многих программных приложений объединение описаний структур данных и алгоритмов их обработки вполне естественно и не является их недостатком, оно поддерживается (а в ряде случаев строго регламентируется) языками программирования высокого уровня. Классическая схема исходного кода прикладной программы (листинг 1.2), написанной на высокоуровневом языке, предполагает наличие декларативной части, включающей описание типов и структур используемых данных, и собственно алгоритмической части, содержащей языковое описание алгоритмов их обработки.

```
#include <iostream>
using namespace std;
class Point {
   int x, y;
public:
   void display();
int& givex() {return x;}
int& givey() {return y;}
Point (int xi = 0, int yi = 0) {x = xi; y = yi;} };
void Point::display() {
   cout << «x = » << x; cout << «, y = » << y << «\n»;}
void main() {
setlocale(LC_ALL,«Rus»);
Point A;     Point B(5,3);
cout << «Точка A: »;  A.display();
cout << «Точка B: »;  B.display();
system(«pause»);}
```

Листинг 1.2
Пример кода программы на языке высокого уровня

Объединение в едином программном коде *описания структур данных*, называемого *метаданными*, и описания алгоритмов их обработки неизбежно приводит к взаимозависимости программ и данных. Изменение метаданных (например, в связи с необходимостью корректировки модели предметной области) или алгоритмов их обработки (например, для реализации нового пользовательского запроса к БД) может потребовать модификации как декларативной, так и алгоритмической частей программного кода.

Следующим этапом эволюции был отказ от хранения метаданных в программном коде и размещение их во внешнем файле аналогично тому, как это ранее было сделано для основных данных. Теперь при изменении структуры данных не требовалось вносить изменения в программный код, достаточно было модифицировать файл с метаданными, доступный прикладной программе и необходимый ей для интерпретации и последующей обработки основных данных. Правда, такой прием давал лишь частичное решение проблемы взаимозависимости данных и программ, так как описание структуры

самих метаданных все равно приходилось хранить в коде прикладной программы.

Было найдено весьма радикальное решение и этой проблемы — прикладные программы были лишены возможности прямого доступа к БД и избавлены от необходимости интерпретации структуры как основных данных, так и метаданных системы. Решение этой задачи, а также всего комплекса задач управления данными возлагалось на специализированную служебную программу, имеющую непосредственный доступ к метаданным БД и обеспечивающую информационную поддержку прикладных программ.

Дальнейшее развитие этой концепции привело к созданию *систем управления базами данных* (СУБД), называемых также *серверами баз данных*. СУБД является необходимым компонентом подсистемы хранения данных АИС и образует промежуточный слой между файловой системой и прикладными программами, входящими в состав системы обработки данных АИС (рис. 1.1).

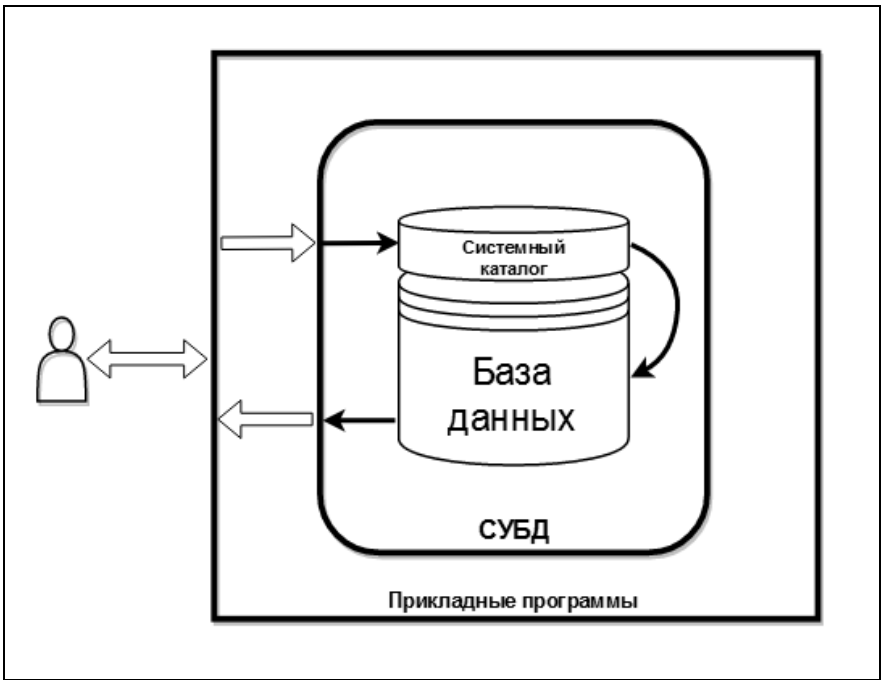


Рис. 1.1
Схема взаимодействия
прикладных программ с базой данных

Прикладные программы используют специальное представление о структуре БД (так называемую *логическую модель данных*) и отправляют в адрес СУБД запросы, сформулированные на языке этой модели. СУБД, получив такой запрос, обращается к системному каталогу (специальному служебному разделу БД, в котором хранятся метаданные — описание структуры БД), транслирует запрос в файловое представление (*физическую модель данных*), затем

исполняет запрос путем прямого доступа к файлам БД и преобразует его результаты в логическое представление, понятное прикладной программе. Таким образом, прикладная программа получает от СУБД запрашиваемую информацию в формате логической модели данных, ничего при этом не зная о формате файлового представления этой информации в БД.

Совместное файловое хранение метаданных с основными данными и использование универсальных программных средств управления данными — это принципиальные свойства БД, обеспечивающие ее автономность в составе АИС и независимость от пользовательских программ обработки информации. Именно это отличает базу данных от любой другой системы хранения информации на устройствах внешней памяти.

ГЛАВА 2. МОДЕЛИ ДАННЫХ

2.1. Проектирование как процесс преобразования моделей

Представление о базе данных как об информационной модели предметной области позволяет рассматривать процесс ее создания как процедуру последовательной детализации этой модели — от общего (пользовательского) представления об информационных сервисах, предоставляемых АИС, до схемы размещения соответствующих структур данных на внешних запоминающих устройствах.

С другой стороны, представление о базе данных как о сложном программно-техническом объекте позволяет применять к процессу ее проектирования весь комплекс методов, инструментов и технологических приемов программной инженерии, основанных на базовых принципах, обеспечивающих эффективное решение проблемы «борьбы со сложностью» проектируемых объектов и повышение производительности процесса разработки.

Принцип *декомпозиции* предписывает рассматривать сложную систему как множество взаимосвязанных компонентов, проектирование которых может выполняться параллельно разными командами разработчиков с последующим объединением полученных ими результатов в единый проект. Примером применения этого принципа может служить декомпозиция АИС на две относительно автономные подсистемы — базу данных и подсистему обработки информации, технологии создания которых существенно отличаются и требуют привлечения разработчиков различных специальностей.

В соответствии с принципами *иерархичности* и *многоэтапности* процесс проектирования базы данных включает ряд последовательных стадий, на каждой из которых разрабатываются соответствующие представления (модели) проектируемого объекта, описывающие его с разной степенью детализации. Проектирование БД — это процесс последовательной детализации пользовательского представления о предметной области АИС и его преобразования сначала в *концептуальную модель*, описывающую объектную среду моделируемой предметной области, затем в *логическую модель*, определяющую представление прикладных программ о структуре базы данных, и наконец в *физическую модель* данных, детально определяющую структуры данных и схему их размещения на устройствах внешней памяти.

Основным содержанием начальной стадии проекта АИС является проведение детального анализа бизнес-процессов предметной области с целью выявления и согласования с заказчиком требований к проектируемому объекту и оценки их реализуемости. На этой стадии для описания проектируемой базы данных используются так называемые *внешние модели* — представления об основных функциях и информационных сервисах, предоставляемых АИС по запросам пользователей.

Технологии анализа требований к проектируемым программным системам — это отдельное направление программной инженерии и в данном учебнике не рассматриваются. Существует множество методов анализа бизнес-

процессов и поддерживающих их CASE-средств, среди которых можно отметить, например, UseCase-модель языка UML, позволяющую классифицировать пользователей проектируемой системы, определить состав ее сервисов и описать процесс их взаимодействия в процессе функционирования АИС. UseCase-диаграмма и сценарии вариантов использования проектируемой АИС содержат информацию, необходимую проектировщику БД для разработки концептуальной модели.

2.2. Концептуальная модель предметной области АИС

Информационное моделирование бизнес-процессов было введено в практику проектирования БД работами Брауна (A. P. G. Brown) [17], Чена (P. Chen) [14, 21] и ряда других авторов [33] в 1960–1970-х гг. В этих работах была предложена концептуальная модель предметной области, представляющая результат ее объектной декомпозиции. В основе концептуальной модели лежали понятия сущности (entity) и связи (relationship), рассматриваемые как абстракции реальных моделируемых объектов и семантических отношений между ними. Такая модель получила название *ER-модели*, или модели «*сущность — связь*».

Каждая сущность в такой модели представляет множество однотипных экземпляров некоторого объекта предметной области и наделяется «атрибутами», описывающими свойства объектов, существенные в рамках решаемой задачи. Множество значений описательных атрибутов экземпляров сущностей является основным результатом выполнения пользовательских запросов к базе данных, при этом некоторые атрибуты могут выступать и в роли идентификаторов, с помощью которых производится выборка соответствующих экземпляров.

Связи между сущностями представляют отношения между реальными объектами и обеспечивают возможность навигационного поиска экземпляров одних сущностей по их связям с другими экземплярами.

Для визуального представления ER-модели было разработано несколько систем графической нотации, на их основе созданы CASE-средства, многие из которых и сегодня эффективно используются на начальных стадиях проектов баз данных. Считается, что *ER-диаграмма*, известная также как *диаграмма Чена*, положила начало разработке средств графического моделирования, используемых при проектировании программных систем. Среди множества диаграмм современного языка графического моделирования UML одно из центральных мест занимает *диаграмма классов*, унаследовавшая основные свойства ER-диаграммы.

2.3. Дореляционные логические модели данных

Логическая модель данных формируется на базе концептуальной модели и представляет собой ее детализацию и последующее описание на некотором высокоуровневом языке, поддерживаемом СУБД. Качество логической модели во многом определяет эффективность алгоритмов поиска данных, реализуемых СУБД.

2.3.1. Иерархическая модель

Разработчики первых СУБД использовали *иерархические* модели, которые базировались на древовидных структурах данных, хорошо известных в программировании. Иерархическая база данных представляется множеством деревьев: в вершинах дерева помещаются *записи*, состоящие из поименованных *полей* и представляющие экземпляры некоторого объекта предметной области. Записи связаны строго иерархическими отношениями — у записи-«потомка» не должно быть более одной записи-«предка».

Одной из первых СУБД, использующих иерархическую модель данных, была разработка компании IBM 1968 г. *Information Management System* (*IMS*) [32], первоначально предназначавшаяся для управления спецификацией изделий аэрокосмической отрасли США. Следует отметить, что такая модель идеально подходила для моделируемой предметной области, по своей природе имеющей иерархическую структуру: спецификация технического объекта описывается в терминах «изделие» — «агрегат» — «узел» — «деталь», связанных иерархическим отношением композиции.

Основным структурным элементом иерархической модели, поддерживаемой IMS, является так называемый сегмент, представляющий множество однотипных объектов предметной области. Каждый сегмент может включать атомарные информационные блоки, называемые «областями» и представляющие атрибуты экземпляров этого объекта. Сегменты модели могут быть иерархически связаны друг с другом, что отражает определенные семантические отношения между объектами предметной области.

Например, в базе данных интернет-провайдера (рис. 1.2) корневой сегмент «*Клиенты*», включающий области «Имя», «Адрес» и «Телефон», связан с дочерними сегментами «*Услуги*» и «*Заявки*», экземпляры которых хранят информацию соответственно об услугах, оказываемых провайдером своим клиентам, и о поступивших от них заявках.

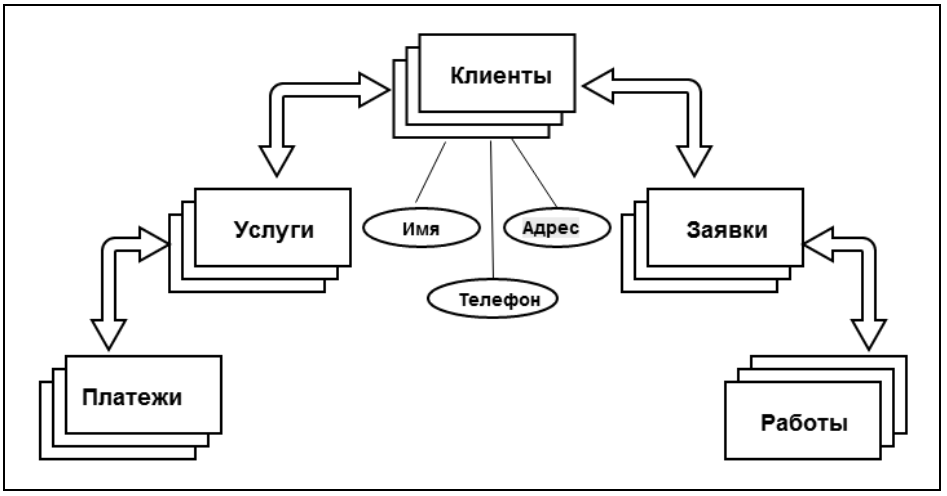


Рис. 1.2
Фрагмент иерархической модели данных

В свою очередь, сегмент «*Услуги*» может быть связан с дочерним сегментом «*Платежи*», каждый экземпляр которого содержит информацию о платежах, полученных провайдером за соответствующую услугу, оказанную клиенту, а сегмент «*Заявки*» связан с дочерним сегментом «*Работы*», представляющим работы, выполненные провайдером по заявкам клиентов.

Главным преимуществом иерархической модели является высокая эффективность алгоритмов поиска (стоимость процедуры «спуска по дереву» пропорциональна глубине дерева), а основными ее недостатками считаются высокая стоимость операций модификации (алгоритм расщепления и слияния вершин дерева при вставке и удалении записей) и необходимость дублирования данных при их хранении в БД.

Причиной дублирования данных является требование строгой иерархичности модели в ситуациях, когда в предметной области это требование объективно не соблюдается. Например, если в иерархической БД интернет-провайдера (рис. 1.2) необходимо дополнительно учитывать распределение работ по выполнению заявок между сотрудниками, потребуется создать дубликаты вершин в различных деревьях базы данных (рис. 1.3*а*).

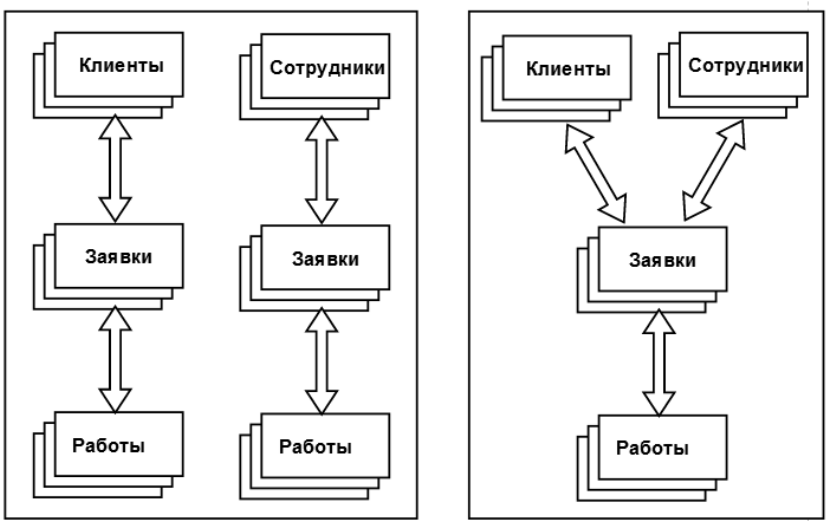


Рис. 1.3
Дублирование данных в иерархической модели:
а — иерархическая модель; *б* — отказ от строгой иерархии.

Более радикальное решение в такой ситуации (рис. 1.3*б*) — отказ от иерархической модели данных в пользу сетевой модели, позволяющей связывать дочерние сегменты с несколькими родительскими.

2.3.2. Сетевая модель CODASYL

В 1969 г. Конференцией по языкам систем данных (Conference on Data Systems Languages) была разработана *сетевая модель данных*, получившая название «*модель CODASYL*» [18]. Спецификация CODASYL содержит деталь-

ную проработку сетевой модели данных, которая, как известно, способна представлять весьма сложные отношения между элементами данных, но также хорошо известны и проблемы программной реализации методов хранения и навигационного поиска на такой модели данных (достаточно вспомнить алгоритмы поиска путей на графах, реализованных на списковых структурах).

Спецификация CODASYL объединяет в себе элементы концептуальной, логической и физической моделей данных: она включает язык определения логической схемы БД, язык манипулирования данными и язык управления внешними носителями данных; для связывания логической схемы БД с файловой структурой данных используется понятие «области базы данных», а для определения пользовательских представлений БД — понятие «подсхемы». В этой спецификации также были определены функции администрирования, включающие операции резервного копирования и восстановления БД, управления производительностью, сбора статистики, аудита, авторизации пользователей и др.

Основу логической модели данных CODASYL (рис. 1.4) составляют два ее именованных компонента: «запись», поля которой представляют множество свойств моделируемого объекта предметной области, и «набор записей» — двухуровневый граф, моделирующий некоторое иерархическое отношение (агрегации или обобщения) между реальными объектами.

Поле записи также является именованным компонентом модели и может быть представлено как атомарным элементом, так и агрегатом, состоящим из атомарных элементов и/или других агрегатов.

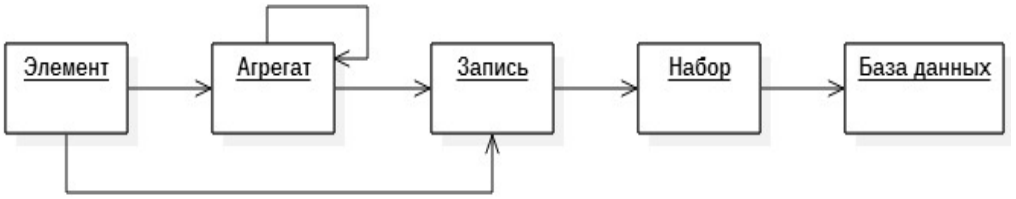


Рис. 1.4
Компоненты сетевой модели данных CODASYL

Таким образом, агрегат представляет собой некоторую иерархическую структуру внутри записи и может рассматриваться и как единое целое, и как множество агрегированных элементов. Пример структуры одной из записей базы данных интернет-провайдера представлен на рисунке 1.5.

Запись «*Клиент*» включает пять полей, два из которых (*Лицевой счет* и *Паспорт*) представлены атомарными элементами, а три остальных — агрегатами типа «повторяющаяся группа». Агрегат *Имя* состоит из трех атомарных элементов, агрегат *Адрес* — из одного атомарного элемента и одного агрегата типа повторяющаяся группа, а агрегат *Телефон* — из двух атомарных элементов и одного агрегата типа вектор.

Такая структура записи позволяет простым запросом к БД получить полную информацию о реквизитах клиента, и при этом обеспечивается возможность доступа к отдельным элементам агрегированных полей записи.

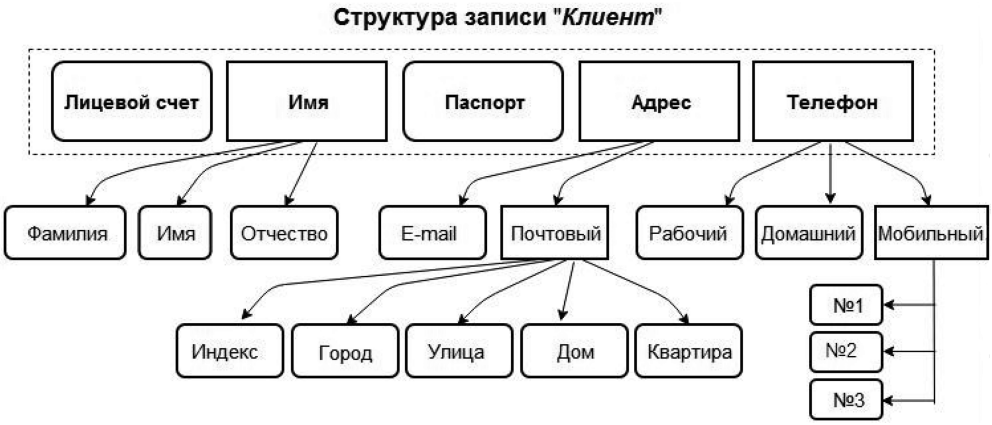


Рис. 1.5
Структура записи сетевой модели CODASYL

Например, можно получить список клиентов, проживающих в определенном городе или на определенной улице города, или реализовать функцию автодозвона до клиентов, имеющих финансовую задолженность на лицевом счете, последовательно выбирая из базы данных номера их рабочего, домашнего и всех мобильных телефонов.

Можно также организовать массовую почтовую рассылку писем всем таким клиентам с автоматическим заполнением соответствующих адресных полей на почтовом конверте, а в тексте письма — с уважительным обращением к каждому клиенту по имени и отчеству (перед стандартным напоминанием о необходимости срочного погашения накопленной задолженности и угрозой судебного преследования в противном случае).

Заметим, что поле *Паспорт* в этой записи представлено атомарным элементом, хотя реальные паспортные данные клиента — это более сложная структура, которую вполне можно было бы описать агрегатом вида «*серия — номер — дата выдачи — место выдачи*». Возможно, решение об атомарности этого поля было принято разработчиком модели по результатам анализа пользовательских запросов к проектируемой базе данных, который показал, что паспортные данные клиента необходимы только для автоматизированного формирования заключительного раздела клиентского договора.

В модели данных CODASYL «записи» могут объединяться в «наборы» — двухуровневые иерархические структуры, в каждой из которых одна запись является «владельцем», а другая — «членом» набора. Запись может быть членом и одновременно владельцем нескольких разных наборов (что, собственно, и определяет сетевой характер этой модели), однако в наборе должна соблюдаться иерархия экземпляров записей: экземпляр записи-владельца может быть связан с несколькими экземплярами записей-членов, но экземпляр записи-члена не может быть связан более чем с одним экземпляром-владельцем.

На базе спецификации CODASYL Чарльзом Бахманом (C. W. Bachman) была разработана система графического отображения сетевой модели данных [16], получившая название «диаграмма Бахмана» и дающая программисту визу-

альное представление о структуре записей, их параметрах, а также об отношениях между объектами и путях навигационного поиска данных.

На рисунке 1.6 представлен фрагмент упрощенной диаграммы Бахмана, описывающей сетевую модель базы данных интернет-провайдера, иерархическая модель которой была рассмотрена выше (рис. 1.2 и 1.3*а*). Наборы записей на диаграмме обозначены линиями со стрелками, направленными от записей-владельцев к записям-членам наборов.

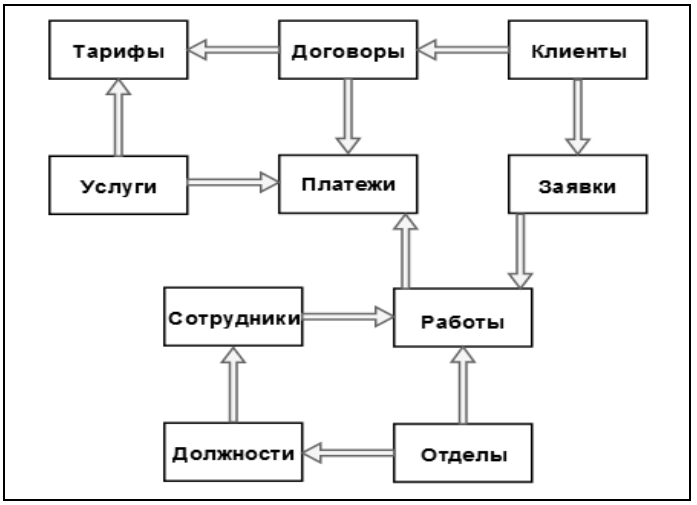


Рис. 1.6
Графическое представление фрагмента модели CODASYL

Набор «*Услуги→Тарифы*» представляет собой модель прайс-листа интернет-провайдера: с каждым экземпляром записи-владельца «*Услуги*» связано множество экземпляров записи-члена «*Тарифы*». Клиентская база провайдера представлена набором «*Клиенты→Договоры*», а содержательная часть договоров с клиентами — набором «*Договоры→Тарифы*». Экземпляр записи «*Тарифы*», являясь членом двух наборов, представляет один из тарифов одной из услуг, предоставляемых клиенту по одному из заключенных им договоров.

Набор «*Клиенты→Заявки*» — это модель журнала заявок от клиентов, принятых сотрудником call-центра, а набор «*Заявки→Работы*» определяет состав работ, требующих выполнения для исполнения заявок. Два набора — «*Отделы→Должности*» и «*Должности→Сотрудники*» представляют штатное расписание отделов, а набор «*Сотрудники→Работы*» — распределение работ, связанных с выполнением заявок, между сотрудниками отделов.

Запись «*Платежи*» является членом трех различных наборов: экземпляр этой записи представляет один платеж либо за оказанную клиенту услугу по одному из договоров, либо за выполненную работу по одной из заявок, поступивших от клиента.

Модель CODASYL предоставляет разработчику два метода хранения и извлечения записей: метод прямого доступа, использующий алгоритм хеширования, и метод связанных списков, использующий систему указателей, устанав-

ливающих отношения между записями, в том числе между членами и владельцами наборов. Предусмотрены многочисленные возможности управления форматом записей и средства ускорения доступа к данным, например размещение на одном физическом устройстве (файле) записей-членов и записей-владельцев одного набора.

Преимуществами сетевой модели CODASYL являются способность представлять сложно структурированные свойства информационных объектов и сложные отношения между ними, что позволяет адекватно моделировать свойства практически любой предметной области, а также высокая производительность за счет гибкого управления физической моделью данных.

Недостатки спецификации CODASYL — это обратная сторона ее преимуществ, их обычно связывают с двумя основными факторами.

Во-первых, это сложность самой сетевой модели данных и, как следствие, высокая трудоемкость освоения технологии ее разработки. Разработка, анализ и модификация структуры сетевой базы данных (даже при использовании визуальных средств моделирования и наличии специально подготовленных специалистов) могли занимать весьма длительный период.

Во-вторых, достижение высокой производительности выполнения операций извлечения данных потребовало детального описания физической модели данных, что сделало операции загрузки БД и изменения ее структуры крайне дорогостоящими. В условиях, когда ключ записи напрямую связан с ее физическим адресом и одновременно используется в качестве указателя для реализации наборов данных в виде связанных списков и деревьев, для создания новых связей требуется перестройка БД на физическом уровне.

В 1970 г. на базе сетевой модели была разработана СУБД *IDMS* (*Integrated Database Management System*) [19], послужившая платформой для создания многих корпоративных систем обработки данных. Несмотря на отмеченные выше недостатки сетевой модели CODASYL, построенные на ее основе базы данных в то время намного превосходили по производительности и эффективности хранения данных параллельно развивавшиеся реляционные системы.

2.4. Реляционная модель данных

Теоретические разработки в области реляционных моделей данных были выполнены в 1960–1970-х гг. Коддом (E. F. Codd) [22–24] и позднее Дейтом (C. J. Date) и Дарвеном (H. Darwen) [4, 25–29]. Имеются многочисленные публикации, посвященные теории и практике применения реляционной модели [7, 8, 13, 14]. Отметим основные идеи, положенные в основу реляционного подхода и позволившие реляционным системам вначале составить серьезную конкуренцию иерархическим и сетевым базам данных, а затем практически вытеснить их с рынка СУБД, используемых в АИС массового применения.

В отличие от модели CODASYL, базирующейся на чисто программистских решениях, реляционная модель имеет строгую математическую основу — математическую логику и теорию множеств. Очевидно, сказалась фундамен-

тальная подготовка доктора Кодда, предположившего, что базу данных можно представлять в виде набора отношений (relationships), к которым прямо применимы языки и понятия математической логики и над которыми допустимо выполнение операций реляционной алгебры (также разработанной Коддом и включающей специальное расширение теоретико-множественных операций).

Реляционная база данных — это множество взаимосвязанных именованных отношений. Отношение — это информационная модель реального объекта («сущности») предметной области, формально представленная множеством однотипных кортежей. Кортеж отношения представляет экземпляр моделируемого объекта, свойства которого определяются значениями соответствующих атрибутов («полей») кортежа.

Связи между кортежами отношений (при их наличии) реализуются через простой механизм «внешних ключей», являющихся, по существу, ссылками на атрибуты связываемых кортежей нескольких отношений.

В реализации *кортеж отношения* является аналогом *записи* модели CODASYL, а *атрибут* кортежа — аналогом *поля* записи с той лишь разницей, что реляционная модель не допускает никакой внутренней структуры атрибутов отношений — все они должны быть атомарными. Такое упрощение позволяет ассоциировать отношения реляционной БД с прямоугольными плоскими таблицами, а кортежи отношений — со строками таких таблиц, что упрощает представление о структуре базы данных и делает их доступными даже конечным пользователям, не являющимся специалистами в области IT[1].

Несмотря на то что реляционная модель данных объективно проигрывает иерархической и сетевой моделям по информативности и скорости доступа к данным, ее основное преимущество — в простоте представления структуры БД и, как следствие, в высокой технологичности разработки БД, а также в эффективности выполнения модифицирующих операций[2].

Первой реляционной СУБД была *System R* [30], созданная в середине 1970-х гг. корпорацией IBM в результате выполнения исследовательского проекта. Многие архитектурные решения и алгоритмы, реализованные в *System R*, были использованы при разработке последующих коммерческих реляционных СУБД. *System R* была также первой СУБД, поддерживающей язык SQL.

Увеличение аппаратной мощности компьютеров, переход на мини-ЭВМ, внедрение клиент-серверной архитектуры вычислительных систем привели к тому, что к середине 1980-х гг. высокая технологичность реляционных СУБД сделала их более популярными, а их традиционная критика за низкую производительность перестала быть актуальной.

[1] Подтверждением тезиса об «общедоступности» реляционных баз данных может служить факт включения реляционных СУБД в популярные пакеты офисных приложений наряду с органайзерами, текстовыми редакторами и электронными таблицами.

[2] Чтобы убедиться в правомерности такого утверждения, полезно сравнить по производительности типовые алгоритмы вставки/удаления узла графа или вершины дерева с алгоритмом вставки/удаления строки в неотсортированной таблице.

В результате реляционные системы постепенно вытеснили с рынка неудобные в разработке и негибкие при эксплуатации иерархические и сетевые CODASYL-системы, получившие общее название «дореляционных» СУБД.

2.4.1. Компоненты реляционной модели данных

Логическая модель данных считается заданной, если определены три ее базовые составляющие: *структурная*, *целостностная* и *манипуляционная*.

Структурная составляющая модели определяет множество допустимых *структур данных* логического уровня, обеспечивающих представление сущностей и связей ER-модели. Структуры данных должны поддерживаться на языковом уровне, на них должны «работать» методы манипуляционной составляющей модели.

Целостностная составляющая включает множество *ограничений целостности данных*, поддерживаемых серверами БД и обеспечивающих возможность автоматического контроля непротиворечивости и согласованности объектов БД при их модификации соответствующими запросами.

Манипуляционная составляющая модели — это набор *допустимых операций*, выполняемых над допустимыми структурами данных и обеспечивающих реализацию пользовательских запросов к базе данных.

2.4.2. Допустимые структуры данных

Единственной структурой данных, допустимой в реляционной (R-) модели и используемой для представления сущностей, атрибутов и связей ER-модели, является *отношение* (*relationship*). По аналогии с языками программирования отношение можно рассматривать как структурированный тип данных, используемый для представления *неупорядоченного множества кортежей*, каждый из которых состоит из множества *атрибутов* соответствующих *скалярных типов*.

Используют также и альтернативные наименования реляционных структур: отношения называют *таблицами* или *множествами записей*, кортежи — *строками* или *записями*, а атрибуты кортежей — *столбцами таблиц* или *полями записей*.

Описание переменной типа отношение называется *схемой отношения* (соответственно — *заголовком таблицы* или *описанием типа записи*).

Схема отношения описывает внутреннюю структуру всех входящих в него кортежей и включает список имен атрибутов этого отношения с указанием для каждого атрибута типа данных, домена допустимых значений и других ограничений целостности.

Концепции типов данных и доменов допустимых значений атрибутов отношений будут рассмотрены ниже при обсуждении ограничений целостности реляционной модели данных. При обращении к атрибутам допускается использовать их двухуровневые имена (по аналогии с именами элементов структурированных типов данных в языках программирования), когда в качестве префикса имени атрибута используется имя его отношения, например атрибут ***atr*** кортежей отношений ***R1*** и ***R2*** может быть обозначен соответственно как ***R1.atr*** и ***R2.atr***.

Значение переменной типа отношение, называемое *телом отношения,* — это множество кортежей, структура которых соответствует схеме отношения. Каждый кортеж отношения представляет некоторый экземпляр соответствующей сущности, а значения атрибутов кортежа представляют свойства этого экземпляра.

Язык SQL содержит операторы *Create Table* и *Alter Table*, используемые для определения и модификации схем отношений. Для модификации тела отношения используются SQL-операторы *Insert*, *Delete* и *Update*, определяющие состав кортежей тела отношения и значения их атрибутов. В определенном смысле эти операторы можно считать операторами присваивания значений переменным типа отношение.

Отношение характеризуется *арностью* и *мощностью*: арность — это количество атрибутов в каждом из кортежей (соответственно столбцов в таблице или полей в записи), а мощность — это количество кортежей в теле отношения (строк в таблице или записей во множестве).

2.4.3. Ограничения целостности данных

Обеспечение целостности базы данных в процессе ее эксплуатации — одна из важнейших функций СУБД, при этом под целостностью понимается сохранение согласованного и непротиворечивого состояния информации, хранимой в базе данных.

Реляционная модель данных накладывает ограничения целостности двух видов: ограничения *структурной целостности* отдельных отношений базы данных и ограничения *ссылочной целостности*, обеспечивающие согласованность связей между ними.

Структурная целостность отношений включает:

- базовые ограничения целостности реляционной модели данных:
  - *атомарность атрибутов*;
  - *уникальность кортежей* отношения;
- ограничения *типов данных* атрибутов отношений;
- ограничения *доменов* допустимых значений атрибутов;
- *проверяемые* ограничения, накладываемые на значения атрибутов.

Атомарность атрибутов означает, что ни один из атрибутов отношения не может иметь никакой внутренней структуры, видимой пользователям и поддерживаемой СУБД[3]. В реализации требование атомарности атрибутов предписывает использовать для атрибутов отношений исключительно скалярные типы данных, что позволяет считать отношение *плоской таблицей*.

Требование *уникальности кортежей* запрещает наличие в теле отношения кортежей-дубликатов. Такой «запрет» вытекает из определения отношения как *множества* кортежей — в классической теории множеств все элементы множества должны быть различными.

[3] Для сравнения — записи сетевой модели CODASYL (рис. 1.5) могут иметь агрегированные поля, что, несомненно, является достоинством этой модели данных.

В реализации соблюдение этого ограничения требует наличия в схеме отношения *первичного ключа* — такого минимального подмножества атрибутов, составное значение которых не должно повторяться в разных кортежах и может использоваться в качестве уникального идентификатора кортежа. СУБД, получив информацию о статусе «первичного ключа» некоторого атрибута отношения, не допустит повторения его значений в различных кортежах, что, собственно, и будет гарантировать отсутствие кортежей-дубликатов в теле отношения.

Если в схеме отношения объективно присутствуют несколько таких уникальных идентификаторов (элементарных или составных), первичным ключом объявляется самый экономичный из них, а все остальные получают статус «возможного первичного ключа», сохраняя при этом свойство уникальности.

Требование «экономичности» первичных ключей вытекает из способа реализации связей (раздел 4.1) между отношениями реляционной базы данных, согласно которому схема одного из связываемых отношений дополняется атрибутом — так называемым *внешним ключом*, в качестве которого используется копия первичного ключа другого отношения. Имеются и другие основания требовать экономичности первичных ключей — одно из них связано с использованием индексных структур данных, в которых ключи многократно дублируются на различных уровнях индексных «деревьев».

Если ни один из возможных первичных ключей не удовлетворяет требованию экономичности, разработчик может дополнить схему отношения «искусственным» атрибутом «короткого» типа данных, присвоить этому атрибуту свойство уникальности и объявить его первичным ключом. Как правило, СУБД поддерживают «автоинкрементные» целочисленные типы данных, специально предназначенные для искусственных первичных ключей отношений, и автоматически присваивают таким ключам очередные (или случайные) уникальные значения при вставке кортежей.

Включение в схемы отношений искусственных первичных ключей, не ассоциированных ни с одним из свойств сущностей предметной области, и использование автоинкрементных типов данных для таких атрибутов считается хорошим стилем, так как избавляет разработчика базы данных от необходимости присваивать значения первичным ключам и контролировать их уникальность.

Ограничение типов данных атрибутов отношения уже затрагивалось ранее при обсуждении их атомарности. Реляционные СУБД поддерживают множество скалярных типов данных — строковых, числовых, темпоральных и многих других. Ограничение типа данных трактуется здесь точно так же, как и в языках программирования — тип данных атрибута определяет низкоуровневый формат его хранения, ограничивает множество потенциально возможных значений атрибута и набор допустимых операций по его обработке, а также блокирует возможность сравнения значений атрибутов разных типов.

Ограничение домена позволяет более тонко разграничить значения атрибутов одного типа: атрибуты отношений будут считаться сравнимыми, только если они принадлежат одному домену. При этом *домен* может рассматриваться как некоторый подтип базового типа данных. Например, пусть в схеме отноше-

ния *Товарный_Склад* определены атрибуты числового типа *цена*, *количество* и *срок_хранения*, принадлежащие соответственно к доменам *Цены_товаров*, *Складской_запас* и *Сроки_реализации*.

Принадлежность этих атрибутов числовому типу данных формально позволит выполнять над ними различные математические операции: например, можно определить суммарную стоимость складского запаса определенного товара умножением его *цены* на *количество* или увеличить нормативный срок реализации товара на 30 дней сложением атрибута *срок_хранения* с константой *30*. Если не учитывать ограничения домена, формально возможными окажутся и любые логические операции сравнения значений этих «однотипных» атрибутов, в том числе и операции, не имеющие смысла. Ограничения домена позволят СУБД блокировать выполнение таких операций, например попытка сравнения значений атрибутов *количество* и *срок_хранения* будет заблокирована, так как эти атрибуты принадлежат разным доменам.

Так называемые *проверяемые ограничения* (*check constraints*) представляют собой логические выражения, связываемые с некоторым атрибутом отношения. СУБД будет автоматически контролировать истинность значения такого выражения при каждой модификации значения этого атрибута.

Ссылочная целостность — это целостность *схемы базы данных*, представленной множеством *схем отношений*, кортежи которых могут быть связаны ссылками на значения их атрибутов — так называемых *внешних ключей* (раздел 4.1).

Для обеспечения ссылочной целостности базы данных СУБД должна контролировать соответствие типов данных и доменов, заданных для первичных и внешних ключей в схемах связываемых отношений, а также контролировать соответствие значений этих ключей при выполнении любых операций модификации связанных кортежей отношений.

Концепции структурной и целостностной составляющих реляционной модели данных иллюстрируются листингом 1.3, на котором приведена SQL-реализация фрагмента схемы реляционной БД, описывающего контингент студентов.

Операторами CREATE TABLE создаются схемы трех отношений (таблиц), представляющих три сущности предметной области: «Факультеты», «Студенческие группы» и «Студенты». В каждой из схем отношений определены первичные ключи (ограничение PRIMARY KEY) автоинкрементного (IDENTITY) типа.

Ограничение UNIQUE задано для атрибутов, являющихся *возможными ключами* отношений. СУБД будет блокировать появление дубликатов значений этих атрибутов при вставке или модификации значений кортежей отношений.

Ограничение NOT NULL не допускает *неопределенных* значений атрибутов — если для атрибута не задано *значение по умолчанию* (DEFAULT), то СУБД выполнит откат операции вставки или модификации кортежей.

Для атрибутов ***Groups.Year*** (год обучения студенческой группы) и ***Students.Rating*** (персональный рейтинг студента) заданы *проверяемые ограничения*

(CHECK CONSTRAINT), блокирующие возможность ошибочного ввода значений, выходящих за пределы заданных диапазонов.

В схему отношения ***Students*** включен атрибут ***Group*** числового типа, для которого задано *ограничение внешнего ключа* FOREIGN KEY, обеспечивающее ссылочную целостность базы данных. Атрибут ***Students.Group*** ссылается (REFERENCES) на первичный ключ ***ID_Group*** *родительского отношения* **Groups** и совместим с ним по типу данных. Параметры этого ссылочного ограничения задают поведение СУБД при модификации кортежей родительского отношения.

```
CREATE TABLE Departmets(
ID_Dep INT IDENTITY PRIMARY KEY,
DepShortName CHAR(4) NOT NULL UNIQUE,
DepName VARCHAR(128) NOT NULL UNIQUE,
DepAdress VARCHAR(128) NOT NULL DEFAULT «NoAdress»);
CREATE TABLE Groups(
ID_Group IDENTITY PRIMARY KEY,
GroupName CHAR(8) NOT NULL UNIQUE,
Year BYTE NOT NULL DEFAULT 1
        CONSTRAINT LearningYears CHECK(BETWEEN 1 AND 6),
Department INT FOREIGN KEY REFERENCES Departmets(ID_Dep)
        ON DELETE NO ACTION
        ON UPDATE CASCADE),
Monitor INT NOT NULL FOREIGN KEY
        REFERENCES Students(ID_Stud)
ON DELETE SET NULL
ON UPDATE CASCADE);
CREATE TABLE Students(
ID_Stud  IDENTITY PRIMARY KEY,
StudName VARCHAR(32) NOT NULL DEFAULT «InvisibleStudient»,
StudAdress VARCHAR(128) NOT NULL DEFAULT «HomelessStudient»);
Rating BYTE NOT NULL DEFAULT 0
CONSTRAINT PersonalRatings CHECK(BETWEEN 0 AND 100),
Scholarship INT NOT NULL DEFAULT 0,
Bonus INT NOT NULL DEFAULT 0,
Group INT NOT NULL FOREIGN KEY REFERENCES Groups(ID_Group)
ON DELETE NO ACTION
ON UPDATE CASCADE);
ALTER TABLE Students
        ADD CONSTRAINT MonitorBonus
CHECK (
IF Students.ID_Stud IN (SELECT Groups.Monitor FROM Groups)
Then Students.Bonus = 0.01 * Students.Scholarship * Students.Rating);
```

Листинг 1.3
Примеры использования ограничений целостности

При изменении (ON UPDATE) значений первичного ключа ***ID_Group*** в каком-либо кортеже родительского отношения **Groups** соответственно обновляются (CASCADE) и значения внешнего ключа ***Group*** в кортежах дочернего (ссылающегося) отношения, связанных с измененным кортежем родительского отношения. Параметр NO ACTION этого ограничения означает, что при попытке удаления (ON DELETE) кортежа родительского отношения, в котором значение первичного ключа совпадает со значением внешнего ключа в дочернем отношении, СУБД выполнит откат операции удаления (семантически это блокирует удаление студенческой группы, пока в ней числятся студенты).

Внешний ключ ***Monitor*** отношения **Groups** ссылается на первичный ключ ***ID_Stud*** родительского отношения **Students** — эта ссылка определяет студентов, являющихся старостами соответствующих групп. При удалении (ON DELETE) из отношения **Students** кортежа, представляющего студента — старосту одной из групп, внешний ключ ***Monitor*** в соответствующем кортеже отношения **Groups** получит неопределенное значение (SET NULL), что соответствует реальной ситуации: при отчислении старосты студенческая группа на некоторое время может остаться без руководителя.

Внешний ключ ***Department*** ссылается на первичный ключ ***ID_Dep*** отношения **Departmets**, что семантически отражает принадлежность студенческих групп факультетам университета и невозможность удаления факультета, пока на нем обучается хотя бы одна группа студентов.

Оператором ALTER TABLE вносится изменение в схему отношения ***Students*** — добавляется *проверяемое ограничение* ***MonitorBonus*** на значение атрибута ***Bonus***: при выполнении операций модификации кортежей отношения ***Students*** СУБД будет автоматически проверять значение этого атрибута на соответствие заданному ограничению (1% от размера стипендии ***Scholarship*** за каждый балл персонального рейтинга ***Rating*** каждому студенту, являющемуся старостой какой-либо группы).

2.4.4. Методы обработки данных

Манипуляционная составляющая реляционной модели данных включает множество *методов обработки отношений* (единственных допустимых этой моделью структур данных), выполнение которых должно позволить реляционной СУБД «вычислять» результаты реализации SQL-запросов к базе данных.

В реляционной модели определены два базовых «механизма» реализации таких методов — это *реляционная алгебра*, основанная на теории множеств, и *реляционное исчисление*, основанное на математической логике. Реляционная алгебра оперирует понятием «*алгебраическое выражение*», а реляционное исчисление — понятием «*формула*». Любой запрос к базе данных может быть записан либо алгебраически с помощью соответствующего реляционного выражения, либо представлен формулой реляционного исчисления — оба этих представления эквивалентны в том смысле, что формула всегда может быть преобразована в соответствующее ей выражение, а выражение — в соответствующую ему формулу.

И реляционно-алгебраическое выражение, и формула реляционного исчисления «замкнуты» относительно понятия *отношение* — их операндами могут быть только отношения, отношениями являются и результаты их вычисления, что позволяет использовать выражения и формулы в качестве операндов других выражений или формул без ограничений глубины вложенности. В результате становится возможным описание очень сложного запроса к базе данных одним выражением реляционной алгебры или одной формулой реляционного исчисления, что позволяет говорить о большой выразительной мощности двух этих базовых средств манипуляционного компонента реляционной модели данных, составляющих основу языка запросов.

Реляционная алгебра представляет собой процедурный аспект языка SQL и позволяет задать последовательность выполнения логических операций, необходимых для выполнения запроса, а реляционное исчисление дает инструмент для описания условий истинности результата исполнения запроса или ограничения целостности, то есть поддерживает декларативный аспект этого языка.

Язык запросов считается *реляционно-полным*, если одним его оператором можно описать любой запрос, представленный одним выражением реляционной алгебры или одной формулой реляционного исчисления.

Учитывая прикладной характер настоящего издания, ограничимся кратким обзором элементов реляционной алгебры Э. Кодда и реляционного исчисления кортежей в объеме, достаточном для понимания технологии нормализации реляционной базы данных и освоения базовых конструкций языка SQL. Более фундаментально эти вопросы рассмотрены в [7].

2.4.4.1. Реляционная алгебра

Реляционная алгебра базируется на традиционных теоретико-множественных операциях (*пересечение*, *объединение*, *вычитание* и *декартово умножение*) и дополнена четырьмя операциями (*ограничение*, *проекция*, *деление* и *соединение*), специфичными для обработки реляционных данных. Все эти операции обрабатывают *отношения*, которые (по определению) являются *множествами кортежей*.

Кроме этих операций, в реляционную алгебру включают операцию *переименования атрибутов* (*AS*), позволяющую корректно формировать схему (заголовок) результирующего отношения, и операцию *присваивания* (:=), позволяющую сохранять в базе данных результаты вычисления алгебраических выражений.

Объединение R := R1 ∪ R2 — результирующее отношение R включает все кортежи, входящие хотя бы в одно из отношений-операндов R1 или R2.

Пересечение R := R1 ∩ R2 — результирующее отношение R включает все кортежи, входящие в оба отношения-операнда R1 и R2.

Вычитание R := R1 − R2 — результирующее отношение R включает все кортежи, входящие в отношение-операнд R1, такие, что ни один из них не входит в отношение-операнд R2.

Расширенное декартово произведение R := R1×R2 — кортежи результирующего отношения R производятся путем попарного соединения (конкатена-

ции, или сцепления) всех кортежей отношений-операндов R1 и R2. Арность результирующего отношения будет равной сумме арностей всех перемножаемых отношений-операндов, а мощность — произведению их мощностей.

Операндами первых трех операций могут быть только *совместимые* отношения, то есть такие отношения, схемы которых (арность кортежей, имена и типы соответствующих атрибутов) одинаковы. Это ограничение объясняется тем, что результатом операций является отношение, а в отношении все кортежи должны иметь одинаковые схемы. Отношения-операнды, схемы которых отличаются только именами атрибутов, становятся полностью совместимыми после применения к ним операций переименования.

Операция расширенного декартова произведения отношений применима к отношениям, схемы которых не имеют совпадающих атрибутов, и трактуется иначе, чем базовая теоретико-множественная операция декартова произведения, результатом которой является *множество пар* элементов перемножаемых отношений. Реляционная модель не использует понятия «пара кортежей», и по этой причине реляционную операцию называют *расширенным декартовым произведением*. Эта операция не имеет какого-либо содержательного смысла и введена в состав манипуляционной составляющей модели по той причине, что через нее определяются действительно полезные специальные *операции соединения* отношений.

Ограничение R := R1 WHERE *условие* — результирующее отношение R включает подмножество кортежей отношения-операнда R1, удовлетворяющих заданному *условию* (любому корректному логическому выражению).

Проекция R := R1 PROJECT *список атрибутов* — схема результирующего отношения R включает только те атрибуты исходного отношения R, которые включены в *список атрибутов*; если ни один из атрибутов этого *списка* не обладает свойством уникальности, в результирующем отношении потенциально возможны кортежи-дубликаты, которые (при их наличии) удаляются из результирующего отношения.

Деление R := R1 DIVIDE BY R2 — *бинарное* отношение-операнд R1 делится на *унарное* отношение-операнд R2; результирующее *унарное* отношение R включает значения *первого атрибута* кортежей отношения R1 такие, что множество значений *второго атрибута* кортежей этого отношения (при фиксированном значении первого атрибута) включает множество значений единственного атрибута кортежей отношения R2.

Соединение R:=R1 JOIN R2 ON *условие* — кортежи результирующего отношения R образуются путем соединения (конкатенации, или сцепления) кортежей отношений-операндов R1 и R2, удовлетворяющих заданному *условию* (любому корректному логическому выражению). Выполнение операции соединения отношений можно рассматривать как операцию их расширенного декартова произведения с последующей фильтрацией множества кортежей полученного промежуточного отношения по заданному *условию*.

В манипуляционной составляющей реляционной модели определено несколько разновидностей операции соединения:

– ***внутреннее соединение*** (*inner join*) — соединяются только те кортежи отношений-операндов, для которых выполняется заданное *условие*;

– ***левое*** (*left join*) и ***правое*** (*right join*) ***соединение —*** результирующее отношение будет *безусловно* содержать все кортежи левого (или соответственно правого) отношения-операнда, в том числе и те, для которых нет «пары» в другом отношении-операнде, при этом «недостающие» атрибуты в таких кортежах результирующего отношения получат неопределенные *NULL*-значения;

– ***внешнее соединение*** (*foreign* или *outer join*) — одновременно и ***левое***, и ***правое*** соединение;

– ***экви-соединение*** (*equal join*) — такое соединение, *условие* которого содержит оператор сравнения «*равно*»;

– ***естественное соединение*** (*natural join*) — экви-соединение двух отношений, имеющих одинаковые атрибуты (как правило, это первичный и внешний ключи соединяемых отношений), равенство которых и является *условием* соединения кортежей (при этом совпадающий атрибут в схеме результирующего отношения не дублируется).

В таблице 1.1 приведены примеры, иллюстрирующие применение операций реляционной алгебры для реализации запросов к базе данных, моделирующей контингент студентов университета (листинг 1.3).

Таблица 1.1

Примеры выражений реляционной алгебры

№	Выражение	Результирующее отношение	Семантика
1	1st_Year_Groups:=Groups WHERE Group.Year = 1	Множество кортежей отношения *Groups*, для которых атрибут *Year* принимает значение, равное константе 1	Список групп студентов первого года обучения
2	2nd_Year_Good_Students:= (Groups INNER JOIN Students ON Groups.ID_Group = Students.Group) WHERE Groups.Year = 2 AND Student.Rating > 50%	Множество кортежей отношения, полученного в результате соединения отношений *Students* и *Groups*, для которых атрибут *Rating* принимает значение, превышающее 50	Список студентов 2-го курса, имеющих высокий рейтинг (полная информация о студентах и их группах)
3	2nd_Year_Bad_Students := ((Groups INNER JOIN Students ON Groups.ID_Group = Students.Group) WHERE Groups.Year = 2 AND Students.Rating < 30%) PROJECT Groups.GroupName, Students.StudentName	Бинарное отношение — проекция отношения, полученного в результате соединения отношений *Students* и *Groups*, для которых атрибут *Rating* принимает значение, меньшее 50, на атрибуты *GroupName* и *StudentName*	Список студентов 2-го курса, имеющих низкий рейтинг (только имя группы и имя студента)
4	All_Group_Scolarships:= ((Groups INNER JOIN Students ON Groups.ID_Group = Students.Group) PROJECT Groups.GroupName, Students.Scolarship) DIVIDE BY (Students PROJECT Students.Scolarship)	Результат операции деления — унарное отношение *All_Group_Scolarships*: список наименований тех групп, студенты которых получают стипендии всех возможных размеров	

2.4.4.2. Реляционное исчисление кортежей

Реляционное исчисление кортежей базируется на концепции *правильно построенной формулы* (*WFF — Well Formed Formula*), при построении которой используются *кортежные переменные*, *предикаты* и *кванторы*, и понятии *целевого списка* (*Target List*), используемом для формирования схемы результирующего отношения, описанного такой формулой.

Кортежные переменные

Областью определения кортежной переменной является тело некоторого отношения базы данных, а ее допустимым значением может быть любой кортеж этого отношения. Для именования и определения переменной будем использовать конструкцию **RANGE** *переменная* **IS** *отношение*, при этом *переменная* наследует схему кортежа своего *отношения* и допускает обращение к любому своему атрибуту по расширенному имени переменной: *переменная.имя_атрибута*.

WFF-формулы

WFF-формулы используются для описания условий, накладываемых на значения кортежных переменных, и представляют собой логические выражения, принимающие значения *true* или *false*.

Логические выражения WFF-формул включают *предикаты сравнения* скалярных значений атрибутов переменных или констант, *кванторы* существования *EXISTS* и всеобщности *FORALL*, а также операторы отрицания *NOT*, конъюнкции *AND*, дизъюнкции *OR* и импликации *IF ... THEN* с учетом их приоритетов и с возможностью расстановки скобок.

WFF-формула вида EXISTS *var* (*WFF-form*) принимает значение *true*, если в области определения переменной *var* найдется *хотя бы один* кортеж, для которого формула *WFF-form* принимает значение *true*.

WFF-формула вида FORALL *var* (*WFF-form*) принимает значение *true*, если *для всех кортежей* переменной *var* формула *WFF-form* принимает значение *true*.

Пусть на отношениях *Groups* и *Students* (листинг 1.3) заданы кортежные переменные: **RANGE** *Group* **IS** *Groups* и **RANGE** *Student* **IS** *Students*. Таблица 1.2 иллюстрирует области истинности и семантику результатов вычисления WFF-формул, заданных на этих кортежных переменных.

Целевые списки

Целевой список (*Target List*) — это список атрибутов кортежной переменной, образующих схему результирующего отношения, представленного WFF-формулой. *Выражением* реляционного исчисления кортежей называется конструкция вида *target_list* WHERE *WFF-формула*. Значением такого выражения является отношение, схема которого определяется целевым списком *target_list*, а тело — областью истинности *WFF-формулы*.

Следующие два выражения реляционного исчисления кортежей предписывают сформировать тернарные отношения на базе WFF-формул, приведенной в примерах № 5 и 7 таблицы 1.2:

Groups.Group_Name, Groups.Year, Students.Stud_Name
WHERE *Groups.Monitor = Students.ID_Stud*
Groups.Group_Name, Groups.Year, Groups.Department WHERE
FORALL *Groups* (*Groups.ID_Group = Students.Group AND Students.Rating>50*)

Таблица 1.2

Примеры WFF-формул исчисления кортежей

№	WFF-формула	Область истинности формулы	Семантика
1	*Group.Year* = 1	Множество кортежей отношения *Groups*, для которых атрибут *Year* принимает значение, равное константе 1	Полная информация обо всех студенческих группах первого года обучения
2	*Group.Year* > 1 AND *Group.Year* < 3	Множество кортежей отношения *Groups*, для которых атрибут *Year* принимает значение в заданном диапазоне	Полная информация о группах 2-го курса
3	*Student.Rating* > 50%	Множество кортежей отношения *Students*, для которых атрибут *Rating* принимает значение, превышающее 50	Полная информация о студентах, чей персональный рейтинг больше 50%
4	*Student.Scholarship* = 0	Множество кортежей отношения *Students*, для которых атрибут *Scholarship* принимает нулевое значение	Полная информация о студентах, не получающих стипендию
5	*Group.Monitor = Student.ID_Stud*	Множество пар кортежей отношений *Groups* и *Students*, для которых совпадают значения указанных атрибутов	Полная информация о студентах, являющихся старостами групп, и о группах, в которых эти студенты являются старостами
6	EXISTS *Groups* (*Groups.ID_Group = Students.Group AND Students.Scholarships = 0*)	Множество кортежей отношения *Groups*, связанных хотя бы с одним кортежем отношения *Students*, в котором атрибут *Scholarship* принимает нулевое значение	Полная информация о группах, в которых имеется хотя бы один студент, не получающий стипендию
7	FORALL *Groups* (*Groups.ID_Group = Students.Group AND Students.Rating>50*)	Множество кортежей отношения *Groups*, для которых во всех связанных с ними кортежах отношения *Students* атрибут *Rating* принимает значение, большее 50	Полная информация о группах, все студенты которых имеют персональный рейтинг, превышающий 50%

Завершая обзор манипуляционной составляющей реляционной модели данных, отметим, что *выражения реляционного исчисления*, в отличие от *выражений реляционной алгебры*, не определяют процедуры получения результирующего отношения, предоставляя СУБД самостоятельно принять соответствующие процедурные решения.

Контрольные вопросы и задания

1. Перечислите структуры данных, допустимые в R-модели.
2. Каковы базовые ограничения целостности R-модели?
3. Поясните использование ограничений UNIQUE и PRIMARY KEY, накладываемых на значения атрибутов отношений.
4. Как может измениться арность и мощность отношения после применения к нему операции проекции?
5. Определите зависимости арности и мощности результата перемножения отношений от соответствующих параметров отношений-операндов.
6. Используя листинг 1.1 и данные таблиц 1.1 и 1.2, подготовьте собственные примеры выражений реляционной алгебры и WFF-формул реляционного исчисления кортежей.

2.5. Объектные модели данных

Современный (постреляционный) этап развития АИС связан с использованием объектно-ориентированных технологий разработки программных систем и созданием СУБД нового поколения, унаследовавших все лучшее от дореляционных и реляционных систем. Постреляционные СУБД поддерживают объектные и объектно-реляционные модели данных и обеспечивают разработчикам возможность использовать объектно-ориентированные языки программирования (такие, например, как C++, Java, Perl и Python), что дает таким системам технологические преимущества по сравнению с реляционными СУБД.

Рассмотрение объектно-ориентированных моделей данных выходит за рамки этого издания, поэтому приведем лишь два примера постреляционных СУБД, дающих представление о специфических особенностях таких систем.

PostgreSQL [5] создавалась как классическая реляционная СУБД в рамках проекта POSTGRES, начало реализации проекта — 1986 г. В 1996 г. система была существенно переработана и получила свое современное название. В частности, была обеспечена совместимость со стандартом SQL92, расширен набор встроенных типов данных, а также оптимизирована система управления транзакциями (вместо блокировки таблиц было применено многоверсионное управление параллельным доступом), что позволило существенно повысить производительность системы.

Были внесены изменения в модель данных (и соответствующие дополнения в диалект используемого системой языка PSQL), что, собственно, и позволяет считать PostgreSQL объектно-реляционной системой: например, появилась возможность определения отношения множественного наследования дочерними таблицами атрибутов родительских таблиц.

Постреляционная СУБД ***Cachè*** [20], первый выпуск которой состоялся в конце 1997 г., поддерживает *транзакционную многомерную модель данных* (TMDM), которая позволяет оптимально хранить данные в виде многомерных разреженных массивов, и представлять их так, как это требуется приложению. Cachè использует три альтернативных способа доступа к данным: прямой, объектный и реляционный.

Прямой доступ к данным на уровне физической модели обеспечивает максимальную производительность и полный контроль со стороны программиста (подобно тому, как это было сделано в спецификации CODASYL) и позволяет создавать сверхбыстрые алгоритмы обработки данных.

Объектный доступ к данным позволяет естественным образом использовать объектно-ориентированный подход как при моделировании предметной области, так и на этапе реализации. TMDM поддерживает объектную логическую модель в полном соответствии с рекомендациями ODMG (множественное наследование, инкапсуляция и полиморфизм).

Реляционный доступ с использованием встроенного Cachè-SQL позволяет Cachè успешно конкурировать с реляционными системами. Как только в системе определяется класс объектов, сервер многомерных данных автоматически генерирует их реляционное описание, дающее возможность обращения к ним с использованием SQL. Аналогично при импорте в систему описания реляционной БД этот сервер автоматически генерирует объектное описание данных, открывая тем самым доступ к ним как к объектам. При этом исключается дублирование данных, а все операции по редактированию проводятся только над одним экземпляром данных.

Cachè позволяет разработчику комбинировать способы доступа к данным: например, для описания бизнес-логики приложения или создания пользовательского интерфейса АИС более эффективным может оказаться объектный доступ, реляционный доступ может использоваться для обеспечения совместимости и интеграции с инструментальными системами, использующими реляционные БД, а прямой доступ — для реализации операций, в которых применение серверных SQL-процедур не может обеспечить требуемую производительность.

ЧАСТЬ 2.
ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

Как уже отмечалось, база данных — лишь один из многих компонентов АИС и в этом смысле не является «самостоятельным» программным объектом, из чего следует, что проект БД всегда интегрирован в проект разрабатываемой АИС.

Проектирование базы данных — многоэтапный процесс, реализация которого связана с решением двух основных проблем, объединяемых понятиями логического и физического проектирования.

В процессе *логического проектирования* разработчик решает задачу отображения реальных объектов и процессов предметной области в абстрактные объекты логической модели данных. Такое отображение должно быть семантически адекватным моделируемой предметной области и при этом должно быть эффективным, технологичным и иметь соответствующую языковую поддержку средствами СУБД.

Результаты *физического проектирования* базы данных должны обеспечить эффективное хранение данных на внешних запоминающих устройствах и высокую производительность реализации пользовательских запросов. На этом этапе решаются задачи отображения абстрактных объектов логической модели данных на объекты физической модели, поддерживаемые СУБД на файловом уровне, а также задачи формирования дополнительных структур данных (индексов, статистик и пр.), обеспечивающих эффективную трансляцию языкового описания пользовательских запросов к базе данных и высокопроизводительный доступ к запрашиваемой информации.

Стадии и результаты проекта базы данных приведены на рисунке 2.1.

Основной задачей стадии *технического задания* является согласование требований к проектируемой АИС, в том числе и требований к обрабатываемой системой информации.

На этой стадии проводится детальный анализ бизнес-процессов предметной области АИС, по результатам которого выполняется ее *функциональная декомпозиция*: классифицируются конечные пользователи, определяются их ролевые функции в проектируемой системе, формируется структура информационных сервисов, предоставляемых системой каждой категории пользователей, прорабатываются сценарии их взаимодействия.

Результаты функциональной декомпозиции проектируемой АИС документируются и передаются для дальнейшей детализации разработчикам следующей стадии проекта. Одним из способов документирования и графического представления функциональной структуры АИС на ранних стадиях проекта является UML-*диаграмма вариантов использования* (называемая также *UseCase-диаграммой* и *диаграммой прецедентов*), дополненная описанием соответствующих сценариев.

Модель АИС, сформированная на стадии технического задания, отражает пользовательские представления о работе проектируемой системы и называется *внешней моделью* (точнее — множеством внешних моделей, ассоциируемых с различными категориями пользователей).

Внешняя модель является основой для объектной декомпозиции (структуризации) предметной области АИС, выполняемой на следующей стадии — ста-

дии *эскизного проекта*. В результате объектной декомпозиции формируется *концептуальная модель*, представляющая объекты предметной области, информация о которых существенна, то есть должна быть предъявлена пользователям АИС в результате выполнения запросов к базе данных. Концептуальная ER-модель (п. 1.3) описывается в системе терминов *сущность*, *атрибут* и *связь*.

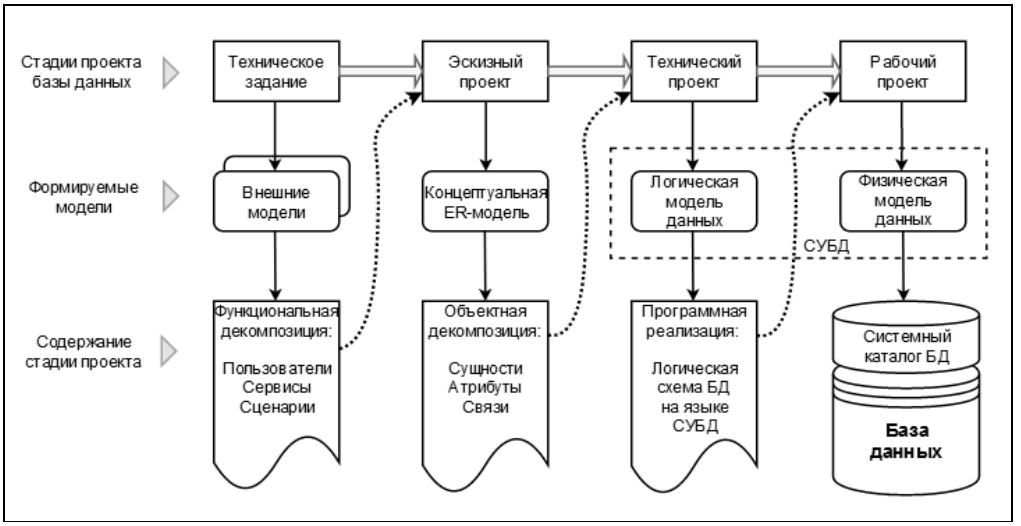


Рис. 2.1
Стадии проекта базы данных

Если две начальные стадии проекта базы данных можно считать «докомпьютерными» и не зависящими от программно-аппаратного обеспечения проектируемой АИС, то следующая стадия (стадия *технического проекта*) является первой из стадий программной реализации базы данных. На этой стадии концептуальная модель преобразуется в *логическую модель данных* и получает программную реализацию на некотором высокоуровневом языке, поддерживаемом СУБД. Разумеется, к этому моменту разработчиками АИС уже должны быть приняты решения об общей архитектуре АИС, типе логической модели данных и выборе соответствующей СУБД.

На завершающей стадии проекта (стадии *рабочего проекта*) СУБД транслирует логическую модель в низкоуровневую *физическую модель* данных и сохраняет параметры этой модели в системном каталоге БД. Настройка и оптимизация параметров физической модели производятся администратором базы данных по результатам мониторинга работы пользователей АИС в процессе ее эксплуатации.

Структура физической модели данных и средства ее оптимизации рассмотрены в четвертой части данного учебника.

ГЛАВА 3. ЭСКИЗНЫЙ ПРОЕКТ. РАЗРАБОТКА КОНЦЕПТУАЛЬНОЙ ER-МОДЕЛИ

3.1. Два уровня объектной декомпозиции

В соответствии с базовыми принципами проектирования сложных объектов, объектную декомпозицию системы целесообразно проводить несколькими этапами на двух иерархических уровнях.

На первом этапе производится декомпозиция объектов верхнего уровня иерархии, в результате которой формируется множество так называемых *локальных представлений*, группирующих объекты предметной области по определенным критериям с целью упрощения последующей разработки ER-модели.

Возможны различные подходы к такой группировке, один из вариантов базируется на результатах функциональной декомпозиции системы, представленной, например, в форме UML-диаграммы вариантов использования (UseCase-Diagram). В зависимости от сложности UseCase-модели локальные представления могут быть сформированы на основе каждого базового варианта использования или на основе множества вариантов использования, ассоциированных с одной пользовательской ролью. В более сложных случаях разработчик ER-модели может принять решение о более детальной декомпозиции на этом уровне, например с использованием вложенности одних локальных представлений в другие.

Для документального оформления результатов этого этапа декомпозиции можно использовать UML-диаграмму пакетов (Package Diagram), в которой множество локальных представлений будет соответствовать множеству поименованных «пакетов», связанных между собой отношениями вложенности или зависимости.

На следующем этапе проводится более детальная декомпозиция каждого локального представления (пакета), в результате разрабатываются локальные ER-модели, представляющие соответствующие пакеты в виде множества взаимосвязанных сущностей. Для документального оформления концептуальных моделей используется *ER-диаграмма* — граф-схема специального вида, представляющая *сущности* (именованные узлы графа) и *связи* между ними (именованные дуги графа, помеченные специальными символами). Для отображения ER-диаграмм возможно также использование графической нотации UML-диаграмм классов: на этих диаграммах аналогом *сущности* является *пассивный концептуальный класс*, не содержащий методов и обозначаемый стереотипом entity.

Завершающий этап объектной декомпозиции связан с объединением локальных ER-моделей в единую модель. Основное содержание этого этапа — формальная унификация общих компонентов различных локальных моделей (исключение дубликатов сущностей, согласование имен подобных сущностей и состава их атрибутов, уточнение типов атрибутов, видов связей и пр.). Задача унификации локальных моделей наиболее актуальна для крупномасштабных проектов, в которых ER-модели локальных представлений могут разрабатываться параллельно и независимо различными командами проектировщиков.

3.2. Сущности и атрибуты

Сущность — это абстракция (информационная модель) реального объекта предметной области, а *атрибут* сущности — абстракция одного из свойств (характеристик) моделируемого объекта. Множество всех атрибутов сущности должно полностью определять характеристики объекта, существенные в контексте проектируемой АИС.

Сущность представляет собой множество однотипных объектов, каждый из которых соответствует в ER-модели одному *экземпляру сущности*, а атрибут сущности представляет множество допустимых значений определенной характеристики моделируемого объекта. Для *каждого экземпляра сущности* определен соответствующий набор *экземпляров атрибутов*. ER-модель не допускает дублирования экземпляров сущности, что соответствует требованию однократного хранения информации о каждом объекте в проектируемой базе данных.

Описательные атрибуты сущностей представляют свойства моделируемых объектов, их значения предъявляются пользователям в качестве *результата* выполнения запроса к базе данных, которым производится выборка требуемых пользователю экземпляров сущности и визуализация (или иная, более сложная обработка) значений их свойств.

Атрибуты другой категории предназначены для идентификации экземпляров сущностей, такие атрибуты называются *идентифицирующими* или *ключевыми*. Ключевые атрибуты (называемые также *ключами*), в отличие от описательных, являются не результатом, а *средством* реализации запросов выборки экземпляров сущности, для которых значения ключей соответствуют заданным ограничениям. Роль ключей могут выполнять некоторые из описательных атрибутов, допускается также объединение нескольких атрибутов в один *составной ключ.*

Различают *первичные ключи*, обладающие свойством уникальности в пределах сущности и однозначно идентифицирующие каждый ее экземпляр, и *вторичные ключи* — идентификаторы *групп экземпляров сущности*.

Первичные ключи

В каждой сущности должен быть определен как минимум один первичный ключ, что гарантирует отсутствие дубликатов среди ее экземпляров. Если несколько атрибутов сущности объективно обладают свойством уникальности, один из них (как правило, самый экономичный) объявляется первичным ключом, а остальные получают статус «возможных первичных ключей», не теряя при этом свойства своей уникальности[4].

Возможно и более радикальное решение — ни один из уникальных описательных атрибутов сущности не получает статуса первичного ключа, а на эту

[4] Требование «экономичности» первичных ключей обосновывается способом реализации связей между сущностями в логической (реляционной) модели данных, предусматривающим создание дополнительных *внешних ключей* в подчиненных таблицах, в которые будут «копироваться» значения первичных ключей связанных с ними главных таблиц. Физическая модель данных также требует экономичного формата представления первичных ключей для эффективной реализации индексных структур, ускоряющих поиск информации в базах данных.

роль назначается дополнительный «искусственный» атрибут, не представляющий никаких свойств моделируемого сущностью объекта предметной области и используемый исключительно для идентификации экземпляров этой сущности.

Поддержка уникальности первичных ключей обеспечивается сервером баз данных, в языке SQL имеются специальные типы ограничений целостности (*unique* и *primary key*) и специальные автоинкрементные типы данных для искусственных ключей.

Вторичные ключи

Вторичные ключи обеспечивают возможность *параметрического поиска* экземпляров сущности, соответствующих заданным значениям ее атрибутов. Состав вторичных ключей сущностей определяется пользовательскими запросами к базе данных, требующими группировки экземпляров сущностей по определенным критериям, включающим соответствующие атрибуты.

Рисунок 2.2 иллюстрирует классификацию атрибутов на примере сущности *Сотрудник*. Все атрибуты сущности, за исключением атрибута *Employee_ID*, имеют статус *описательных*, так как представляют свойства сотрудников предприятия, существенные для системы кадрового учета.

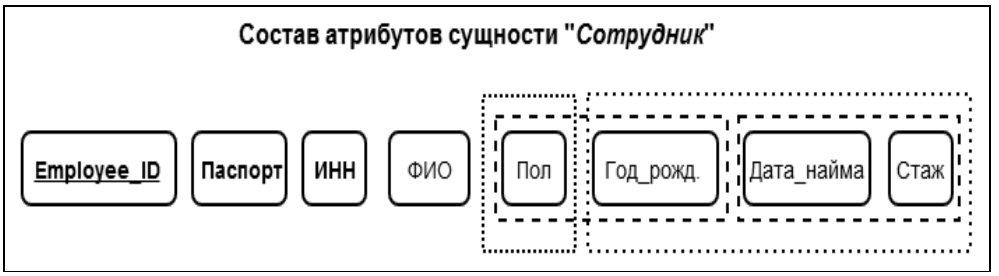


Рис. 2.2
Пример описания сущности ER-модели

Описательные атрибуты *Паспорт* и *ИНН* обладают свойством уникальности, и каждый из них мог бы выполнять функции первичного ключа, но разработчик ER-модели принял другое решение: эти атрибуты объявлены *возможными первичными ключами*, а на роль *первичного ключа* назначен дополнительный «искусственный» атрибут *Employee_ID* (его имя в обозначении подчеркнуто), который более экономичен, выполняет исключительно техническую функцию идентификации, не ассоциируется ни с одним из свойств сотрудника и никогда не будет «показан» конечным пользователям АИС.

Атрибуты *Пол* и *Год_рождения* вместе образуют *составной вторичный ключ*, необходимость которого может быть обусловлена, например, требованием ежегодного формирования справки о сотрудниках предприятия, подлежащих призыву на срочную воинскую службу. Наличие составного вторичного ключа (*Дата_найма, Стаж*) позволит формировать планы повышения квалификации молодых специалистов предприятия, а составной вторичный ключ (*Год_Рождения, Дата_найма, Стаж*) будет полезен при планировании профессиональной переподготовки сотрудников предпенсионного возраста.

По результатам анализа бизнес-процессов могут быть приняты и другие решения, связанные с определением вторичных ключей. Например, если в функции отдела кадров входит подбор персонала для выполнения работ с тяжелыми условиями труда или производится ежегодная рассылка поздравлений с Международным женским днем 8 марта всем сотрудницам предприятия, то атрибут *Пол* также может получить статус *вторичного ключа* (в данном случае *атомарного*, а не *составного*).

Информация о составе вторичных ключей всех сущностей ER-модели должна быть детально проработана и отражена в проектной документации, эта информация будет необходима разработчикам физической модели данных для принятия решений о создании индексов, ускоряющих поиск и группировку данных.

3.3. Связи между сущностями

Связь в ER-модели — это абстракция некоторого семантического отношения между реальными объектами предметной области, существенная в контексте проектируемой базы данных и обеспечивающая возможность *навигационного поиска*, то есть поиска экземпляров одних сущностей по их связям с экземплярами других сущностей.

Например, если в ER-модели подсистемы кадрового учета предприятия определены связи между сущностями *Сотрудники* — *Отделы* и *Сотрудники* — *Должности*, то реализация этих связей позволит соответствующими запросами к базе данных определить место работы и должность каждого «экземпляра» сотрудника, состав сотрудников каждого «экземпляра» отдела, а также штатное расписание отделов (за исключением вакантных должностей).

Связи между сущностями — это именованные элементы ER-модели, их, как правило, именуют глаголами, кратко обозначающими отношения между связанными объектами. В рассмотренном выше примере связь *Сотрудники* — *Отделы* может быть названа «*работает в*», а связь *Сотрудники* — *Должности* может получить имя «*занимает*».

В процессе выявления и именования связей производится их классификация: отнесение каждой связи к одному из *видов связи*, определение параметров *арности* и *кратности* связей, а также определение и именование описательных *атрибутов* связей (при их наличии).

Наиболее общим *видом* связи между сущностями является связь *ассоциации*, все остальные виды связи в определенном смысле являются ее частными случаями, выделяемыми в отдельные категории из-за специфических особенностей их последующей реализации в логической модели данных.

Из других видов связи можно отметить связи, моделирующие иерархические отношения между объектами — это *связь агрегации*, представляющая отношения типа «целое — часть», и *связь обобщения*, используемая для описания иерархий наследования типа «общее — частное» («предок — потомок»). Для обозначения видов связей на ER-диаграмме соответствующие дуги граф-схемы

помечаются специальными символами (отличающимися в различных системах графической нотации ER-диаграмм).

Арность связи определяется количеством сущностей, участвующих в связи. В рассмотренном выше примере обе связи — *бинарные*. Примером *тернарной* связи может служить связь между тремя сущностями *Клиент* — *Договор* — *Услуга*: каждый экземпляр этой связи содержит информацию об одном виде услуг, предоставляемом одному клиенту в рамках одного из заключенных с ним договоров. В *тетрарной* связи участвуют четыре сущности, в *пентарной* — пять, в общем случае связь может быть определена как ***n****-арная*, где ***n*** — количество сущностей, участвующих в связи.

В реализации каждая связь между сущностями ER-модели представляется множеством своих экземпляров — ***n****-арных кортежей*, каждый из которых составлен из ***n*** экземпляров сущностей, участвующих в связи. Количество *экземпляров сущностей*, участвующих в одном *экземпляре связи*, определяют *кратность* связи (называемую также *степенью* или *порядком* связи).

Кратность связей определяется по результатам семантического анализа предметной области в соответствии с простыми правилами, приведенными ниже, для случая бинарной связи.

Если экземпляр одной сущности не может быть связан более чем с одним экземпляром связанной с ней другой сущности, то между этими сущностями определяется симметричная связь *кратности* «***один-к-одному***», обозначаемая на ER-диаграмме как «***1:1***», где «***1***» и «***1***» — параметры кратности соответствующих *концов ассоциации*.

Если экземпляр первой сущности может быть связан более чем с одним экземпляром второй сущности, а экземпляр второй сущности — не более чем с одним экземпляром первой, то между этими сущностями определяется асимметричная связь *кратности* «***один-ко-многим***», направленная от первой сущности ко второй и обозначаемая на ER-диаграмме как «***1:M***». Если такая связь на диаграмме направлена от второй сущности к первой, то она будет иметь кратность «***многие-к-одному***» и будет обозначена как «***M:1***».

Если каждый экземпляр любой из связанных сущностей может быть связан более чем с одним экземпляром другой сущности, то между этими сущностями определяется симметричная связь *кратности* «***многие-ко-многим***», обозначаемая на ER-диаграмме как «***M:N***».

В обозначениях параметров кратности концов ассоциации «***1***» следует трактовать как «*не более одного, в том числе ноль*», «***M***» и «***N***» — как «*неопределенно много, в том числе ноль или один*». Из этого, в частности, следует необязательность участия некоторых (в том числе и всех) экземпляров сущности в экземплярах связи, обозначенной на ER-диаграмме для этой сущности.

Для более точного указания на ER-диаграмме параметров кратности концов ассоциаций можно использовать следующие стандартизованные обозначения: «***n***» (любое натуральное число) — *строго равно* ***n***; «***m ... n***» — *любое натуральное число в заданном диапазоне*; «***1 ...****» — *не менее 1*.

Приведенные ниже примеры иллюстрируют использование параметров кратности концов ассоциации для бинарной связи *Сотрудники* — *Должности* в ER-диаграмме модели системы кадрового учета.

Пример 1. Если на предприятии запрещено штатное совместительство, кратность связи может быть определена как «***M:1***», но более строго — как «***1 ...* : 0 ... 1***», что подчеркивает тот факт, что хотя бы одну должность сотрудник занимать все-таки должен (иначе какой же он «сотрудник»?), и при этом допускается хранение информации о вакантных должностях, не занятых ни одним из сотрудников.

Пример 2. Если штатное совместительство разрешено без ограничений, кратность данной ассоциации будет обозначена как «***M:N***» или, более точно, «***1 ...* : 1 ...****».

Пример 3. Если штатное совместительство сотрудников ограничено, например, тремя должностями, кратность данной ассоциации будет обозначена как «***1 ...* : 1 ... 3***».

Пример 4. Если при формировании штатного расписания действуют ограничения «*не более десяти сотрудников на одной должности*» и «*совместительство запрещено*», то кратность соответствующей ассоциации будет обозначена как «***0 ... 10 : 0 ... 1***».

В отличие от сущностей, связи могут не иметь собственных описательных атрибутов, но в случае их наличия они также должны быть включены в ER-модель, поименованы и соответствующим образом обозначены на ER-диаграмме. Например, для связи *Сотрудники* — *Должности* в рассмотренном выше примере могут быть определены описательные атрибуты *Доля_должностной_ставки*, *Дата_назначения* и *Номер_приказа*, которые не представляют ни свойств сотрудников, ни свойств должностей — они характеризуют связь между этими сущностями.

Примеры обозначений на ER-диаграммах сущностей и связей различных видов в различных системах графической нотации приведены на рисунках 2.3–2.7.

ER-диаграмма, представленная на рисунке 2.3, изображена в стиле графической нотации UML-диаграмм классов. Три класса-сущности, имена которых помечены специальным стереотипом *entity*, моделируют справочники образовательных уровней, специальностей и форм обучения — они связаны с сущностью *Groups*, моделирующей группы студентов, отношениями ассоциации кратности «***1:M***», а классы-сущности *Groups* и *Students* связаны отношением агрегации (частный случай ассоциации) кратности «***1:M***».

Кратность «***1***» («*строго один*») концов ассоциаций обозначает обязательность связей со стороны сущностей-справочников: для каждой студенческой группы обязательно должна быть определена ее принадлежность строго одной специальности, одной форме обучения и одному образовательному уровню.

Кратность «*» концов ассоциации («*неопределенно много, в том числе и ноль*») обозначает необязательность связей со стороны сущности *Groups*: в базе данных могут быть представлены такие образовательные уровни (например, «*начальное профессиональное обучение*» или «*адъюнктура*») и такие формы

обучения (например, «*экстернат*» или «*индивидуальное репетиторство*»), по которым не сформировано ни одной студенческой группы.

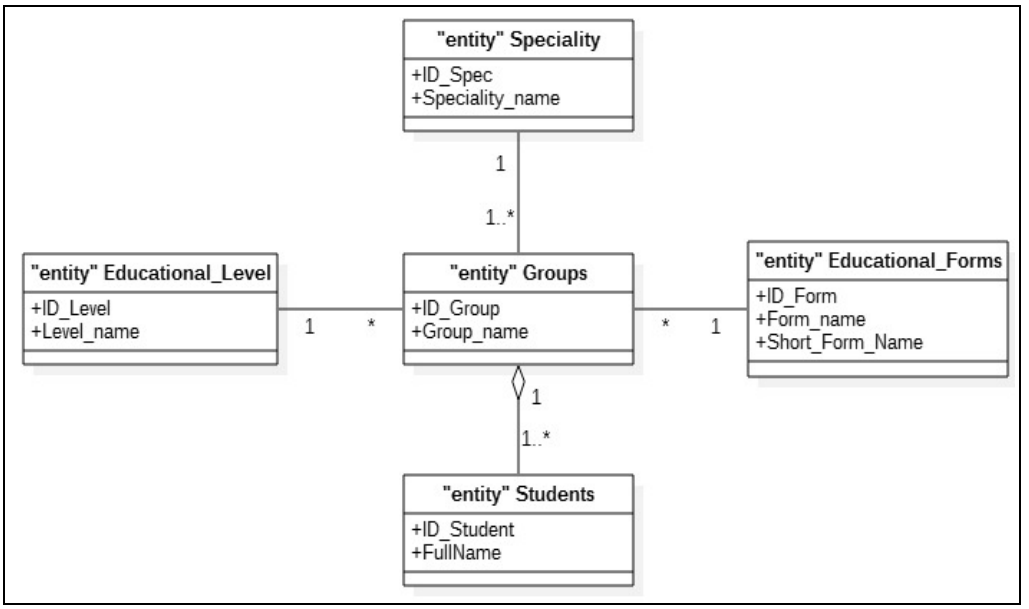


Рис. 2.3
Фрагмент ER-диаграммы модели «Контингент студентов»

Кратность «***1 ... ****» концов ассоциации («*неопределенно много, но не менее единицы*») обозначает обязательность соответствующей связи: по любой специальности должна быть сформирована хотя бы одна студенческая группа, включающая хотя бы одного студента.

Заметим, что кратность конца агрегации со стороны сущности *Students* правильнее было бы обозначить не «***1 ... ****», а, например, «***10 ... 30***», что определяло бы минимально и максимально допустимое количество студентов в группе.

На рисунке 2.4 представлены два варианта изображения бинарной связи кратности «***M:N***» на фрагменте ER-диаграммы системы кадрового учета.

На рисунке 2.5 представлена ER-диаграмма, моделирующая структуру контрольных заданий, обеспечивающих систему тестирования студентов по простейшей схеме — «выбор правильного ответа из нескольких предложенных». Сущность *Type_of_Test* — это модель классификатора тестов (например, «пробный», «контрольный» и «аттестационный»), а экземпляры сущности *Them_plan* представляют множество разделов тематического плана учебной дисциплины.

Для каждой темы может быть сформировано множество тестов различных типов, при этом тест включает множество заданий (*Task*), каждое из которых состоит из строго одного вопроса (*Question*) и множества возможных ответов (*Ansver*), некоторые из которых являются правильными (описательный атрибут экземпляра связи заданий с правильными ответами получит значение «*is_true*»).

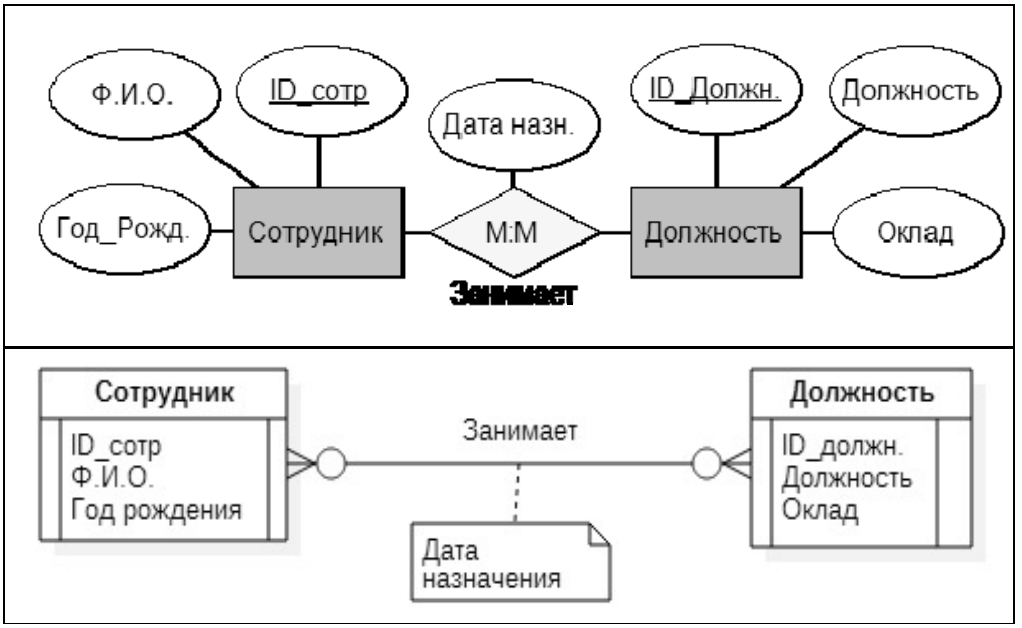


Рис. 2.4
Два способа обозначений связей кратности «***M:N***»

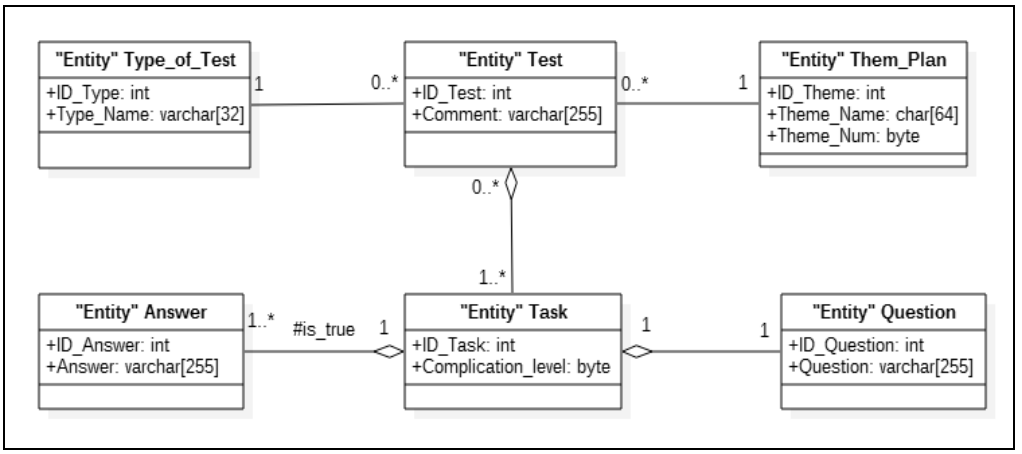


Рис. 2.5
Фрагмент ER-диаграммы модели «Тестирование»

На рисунке 2.6 изображен фрагмент ER-диаграммы предметной области «Библиотечный каталог» в нотации П. Чена.

Объекты хранения в библиотечном фонде представлены двумя категориями: *Книги* и *Журналы*, что отображено на ER-диаграмме соответствующими связями вида «обобщение», на которых стрелка направлена от сущности-«потомка» к сущности-«предку».

Все атрибуты и связи сущности-«предка» *Библ. фонд* наследуются сущностями-«потомками» *Книги* и *Журналы*, при этом каждый из «потомков» мо-

жет иметь собственные атрибуты и участвовать в разных связях с другими сущностями.

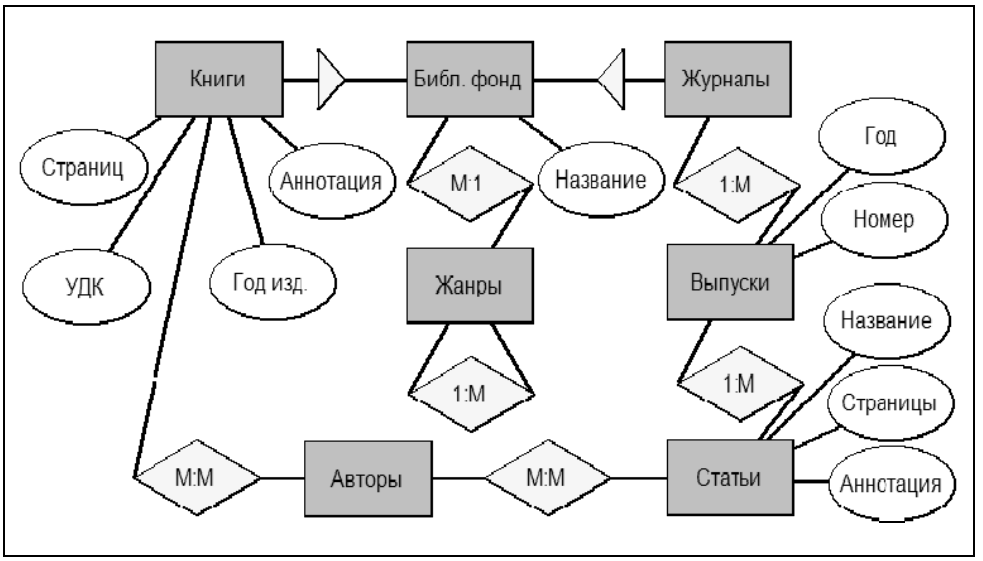


Рис. 2.6
Пример использования унарных связей и связей вида «обобщение»

Для сущности *Жанры* определена *унарная связь* кратности «***1:M***» — это связь между различными экземплярами этой сущности, позволяющая построить *дерево жанров*, в котором один *жанр*-«*предок*» может быть связан с неопределенно большим количеством *жанров*-«*потомков*», каждый из которых может выступать в роли «*предка*» по отношению к другим экземплярам этой сущности.

3.4. Слабые сущности

Слабой называют сущность, экземпляры которой не могут существовать вне связей с экземплярами других (*сильных*) сущностей. Как правило, слабая сущность не является моделью каких-либо реальных объектов предметной области, а «заменяет» собой на ER-диаграмме связь кратности «***M:N***» между сильными сущностями, представляющими реальные объекты.

В результате такой «замены» слабая сущность оказывается связанной с сильными сущностями отношениями кратности «***M:1***» и при этом наследует имя «замененной» связи и все ее описательные атрибуты (при их наличии). Иными словами, каждый экземпляр ***n****-арной связи* кратности «***M:N***» между сильными сущностями заменяется ***n*** экземплярами *бинарной связи* кратности «***N:1***» между слабой сущностью и сильными сущностями, участвовавшими в исходной ***n***-арной связи.

Слабая сущность является «слабой» лишь в отношениях с теми ***n*** сущностями, связь между которыми она заменила, во всех остальных отношениях она подобна другим сущностям ER-модели. Слабая сущность должна иметь соб-

ственный первичный ключ и может участвовать в связях с другими слабыми сущностями в качестве сильной сущности (рис. 2.7).

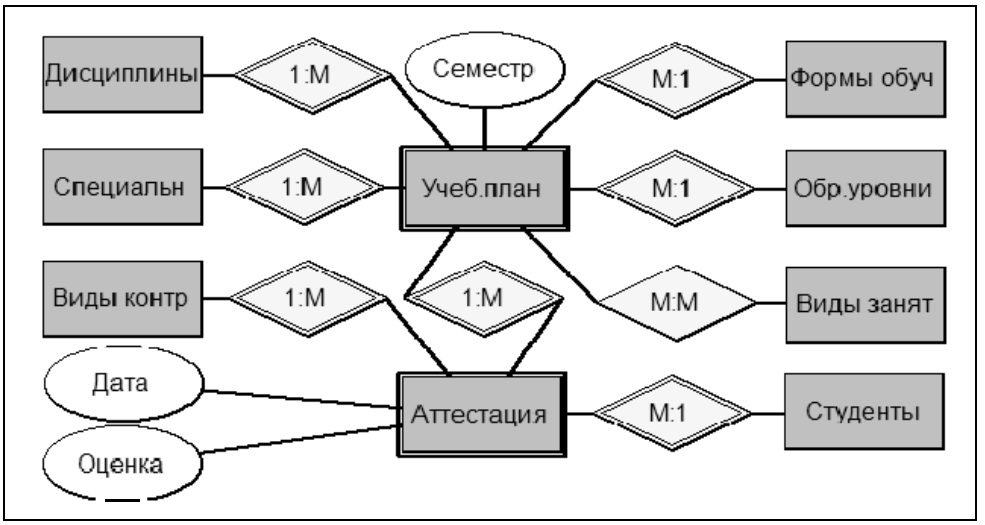


Рис. 2.7
Пример использования «слабых сущностей»

При разработке ER-модели для каждого атрибута сущности или связи дополнительно должны быть определенны такие их параметры, как предполагаемый тип данных и множество допустимых значений, а в необходимых случаях и другие ограничения, которые могут оказаться полезными разработчикам базы данных на последующих этапах ее проектирования и программной реализации.

3.5. Пример разработки ER-модели

Для иллюстрации содержания и результатов выполнения начальных стадий проекта базы данных рассмотрим подсистему учета работы с клиентами интернет-провайдера, которая уже использовалась в качестве примера при обсуждении концепций иерархической (п. 2.3.1) и сетевой (п. 2.3.2) моделей данных.

Техническое задание — первая стадия проекта АИС.

На этой стадии проекта проводится анализ бизнес-процессов интернет-провайдера, по результатам которого определяются основные пользовательские роли и проводится функциональная декомпозиция проектируемой АИС.

Результаты проведенной функциональной декомпозиции представлены на укрупненной UseCase-диаграмме, приведенной на рисунке 2.8.

Внешними пользователями подсистемы являются *Гость* и *Клиент*, которым доступен сервис просмотра тарифных планов, а *Клиенту* и дополнительно сервисы управления договорами и заявками на обслуживание.

Внутренние пользователи — это *Руководитель*, *Менеджер* по работе с клиентами, сотрудники *Call-центра*, а также *Руководитель* и *Сотрудники отдела технической поддержки*.

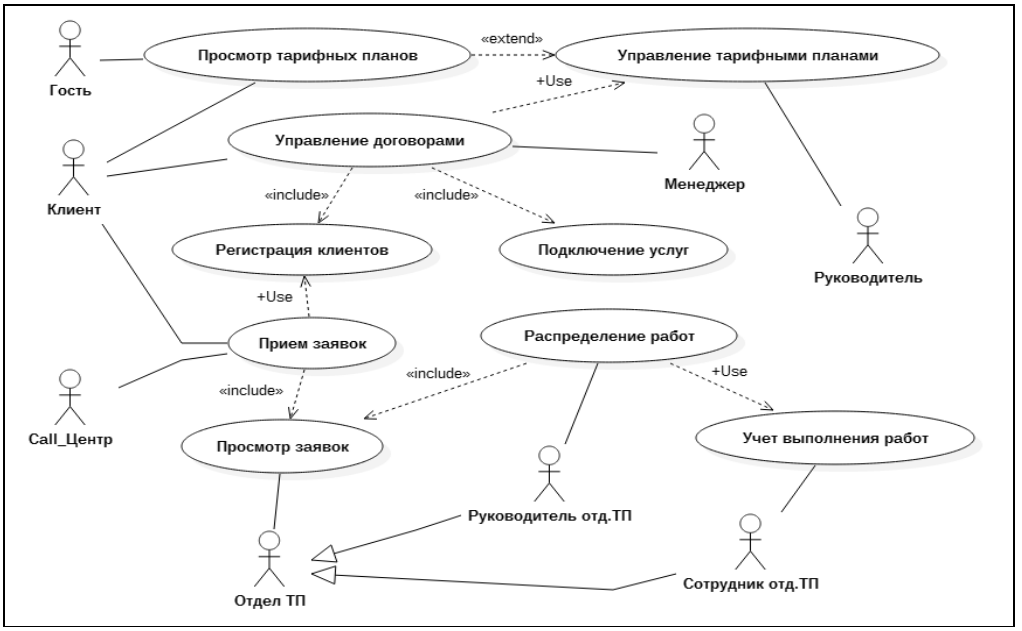


Рис. 2.8
UseCase-диаграмма подсистемы учета работы
с клиентами интернет-провайдера

Руководитель формирует и корректирует тарифные планы.

Сотрудники *Call-центра* принимают, оформляют и регистрируют поступающие от клиентов заявки на обслуживание.

Руководитель отдела технической поддержки анализирует поступившие заявки, распределяет работы по их исполнению между своими сотрудниками и контролирует выполнение работ.

Сотрудникам этого отдела доступны сервисы просмотра заявок и регистрации выполненных ими работ.

Эскизный проект — третья стадия проекта базы данных.

На этой стадии проекта проводится объектная (структурная) декомпозиция предметной области, в результате которой формируется ее информационная ER-модель.

ER-модель представляет проектируемую базу данных на концептуальном уровне как множество взаимосвязанных сущностей, атрибуты которых полно и адекватно описывают свойства моделируемых объектов, а связи между сущностями обеспечивают выполнение системой требований к ее функциональным характеристикам, выработанным на стадии технического задания.

При выполнении крупномасштабных проектов рекомендуется проводить структурную декомпозицию системы в три этапа, последовательно увеличивая степень детализации рассмотрения ее свойств. По завершении каждого этапа производится контроль информативности полученного варианта концептуальной модели.

Следуя этой рекомендации и считая (весьма условно) наш учебный проект «крупномасштабным», получим следующие результаты проведения структурной декомпозиции, иллюстрируемые рисунками 2.9–2.13.

На первом этапе по результатам анализа UseCase-модели АИС (рис. 2.8) произведена декомпозиция проектируемой базы данных на три взаимосвязанных *локальных представления* (или в терминах языка UML — на три *пакета*): *Тарифные планы*, *Договоры с клиентами* и *Обработка заявок* (рис. 2.9).

Пакет *Тарифные планы* обеспечивает информационную поддержку сервисов, востребованных пользовательскими ролями *Руководитель, Гость* и *Клиент*, для формирования, редактирования и просмотра тарифных планов. Пакет включает подчиненный ему пакет *Услуги*, в который, в свою очередь, включен пакет *Тарифы*. Такая структура пакета *Тарифные планы* позволит формировать перечень базовых услуг, оказываемых интернет-провайдером своим клиентам, включать услуги в различные тарифные планы и определять цены услуг (возможно, отличающиеся для различных тарифных планов).

Пакет *Договоры с клиентами* поддерживает функции управления клиентской базой, используемые ролями *Менеджер* и *Клиент*. Подчиненный пакет *Договоры* содержит информацию о заключенных с клиентами договорах. Для формирования предмета и условий договоров этот пакет использует данные пакетов *Услуги* и *Тарифы*, подчиненных пакету *Тарифные планы*, на что указывают соответствующие отношения зависимости (*Use*) между пакетами.

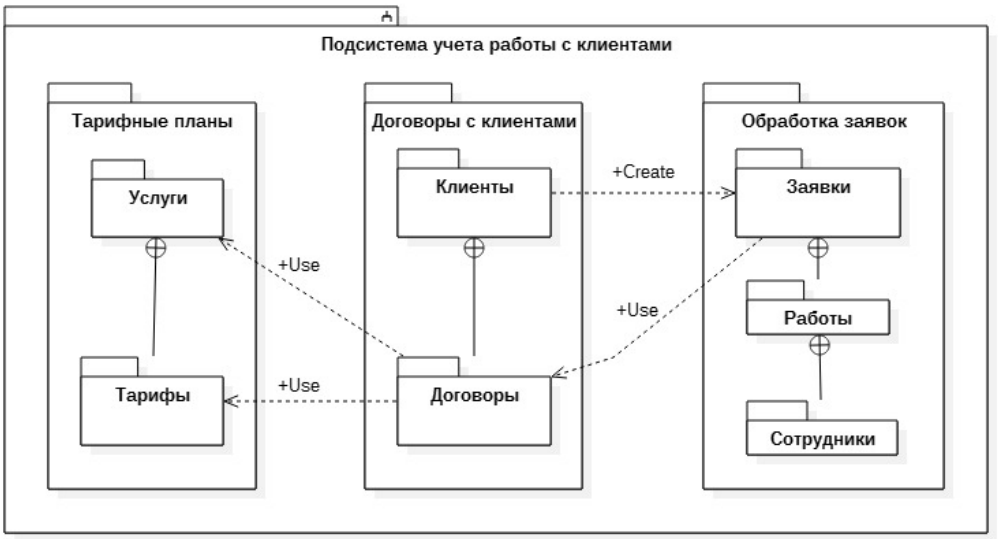


Рис. 2.9
UML-диаграмма пакетов подсистемы учета работы с клиентами

Пакет *Обработка заявок* обеспечивает регистрацию заявок на обслуживание, поступающих от клиентов, а также планирование и учет выполнения работ, связанных с исполнением заявок. Эти сервисы используются клиентами и сотрудниками Call-центра, а также сотрудниками отдела технической поддержки.

Подчиненный пакет *Заявки* связан отношением зависимости типа *Create* с пакетом *Клиенты*, что отражает факт подачи заявки клиентом и потребует установления связей между соответствующими сущностями при разработке ER-модели. Этот же пакет *Заявки* связан отношением зависимости *Use* с пакетом *Договоры*, что обеспечит доступ к содержанию соответствующего договора как на этапе регистрации заявки, так и при планировании работ по ее исполнению.

На втором этапе структурной декомпозиции были разработаны локальные ER-модели для каждого из трех пакетов, сформированных на предыдущем этапе.

ER-модель пакета *Тарифные планы* (рис. 2.10) содержит классификатор услуг, представленный сущностями *Услуги* и *Категории*, связанными отношением кратности «***M:1***». Такое решение даст возможность гибкого управления классификатором услуг в процессе эксплуатации БД без внесения изменений в ее структуру.

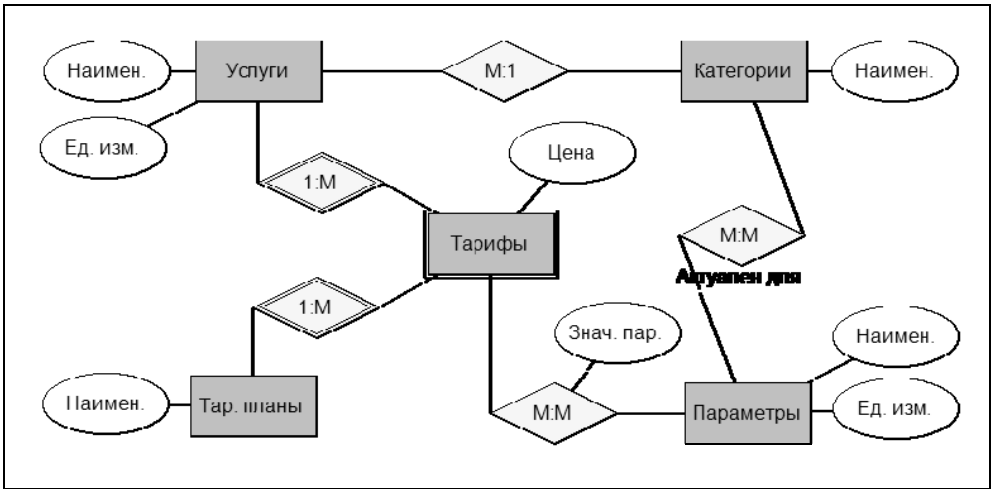


Рис. 2.10
ER-диаграмма пакета *Тарифные планы*

Изменение состава экземпляров и связей между экземплярами этих двух сущностей позволит неограниченно расширять (или сужать до единицы) как перечень категорий оказываемых услуг, так и перечень услуг каждой категории.

Например, прайс-лист одного интернет-провайдера может включать такие категории услуг, как «*Цифровое телевидение*», «*Доступ в Интернет*» и «*Телефония*» (которая может включать услуги «*Мобильная связь*» и «*IP-телефония*»). У другого провайдера все может быть иначе.

Экземпляры сущности *Параметры* представляют перечень наименований различных параметров услуг всех категорий с указанием единиц измерения каждого параметра. Например, параметром услуги «*Просмотр*», принадлежащей категории «*Цифровое телевидение*», может быть «*Количество каналов*»,

а параметрами услуги «*Мобильный Интернет*» категории «*Доступ в Интернет*» — «*Скорость передачи данных*» и «*Ограничение объема передаваемых данных*».

Связь между сущностями *Параметры* и *Категории* позволяет определить состав параметров, актуальных для каждой категории услуг. Кратность ***M:N***, заданная для этой связи, подтверждает тот факт, что услуги одной категории могут иметь много параметров, и один параметр может быть актуальным для нескольких категорий услуг. Заметим, что в такой модели все услуги, отнесенные к одной категории, должны иметь единый набор параметров.

Сущность *Тарифные планы* — это, по существу, перечень наименований пакетов услуг, предоставляемых интернет-провайдером своим клиентам, а каждый экземпляр слабой сущности *Тарифы* представляет одну из услуг в составе одного из таких «пакетов».

Заметим, что атрибут *Цена* представляет свойство *Тарифа*, а не *Услуги*, из чего следует, что одинаковые услуги в различных тарифных планах могут предоставляться по разным ценам.

Связь кратности «***M:N***» между сущностями *Тарифы* и *Параметры* определяет значения параметров услуг, предоставляемых по каждому тарифному плану. Один и тот же параметр одной и той же услуги может иметь различные значения в разных тарифных планах, что, естественно, может повлиять на цену этой услуги. Например, значения параметра «*Ограничение объема передаваемых данных*» услуги «*Мобильный Интернет*» могут существенно отличаться в тарифных планах «*Бюджетный*» и «*Безлимитный*».

Основная цель проведения детального анализа результатов концептуального моделирования — оценка информативности и адекватности разработанных локальных ER-моделей. Пример такого анализа для пакета *Тарифные планы* приведен выше, оценку качества ER-моделей для двух остальных пакетов (рис. 2.11 и 2.12) читателю предлагается провести самостоятельно.

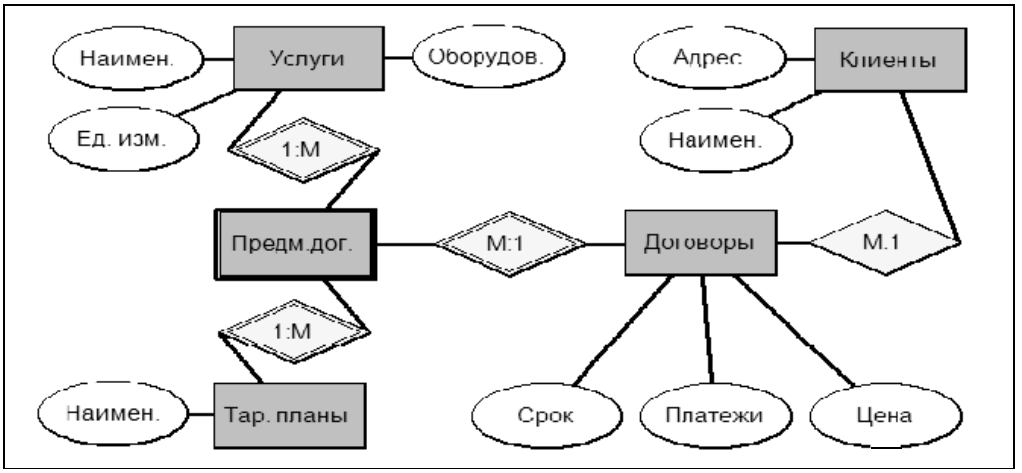


Рис. 2.11
ER-диаграмма пакета *Договоры с клиентами*

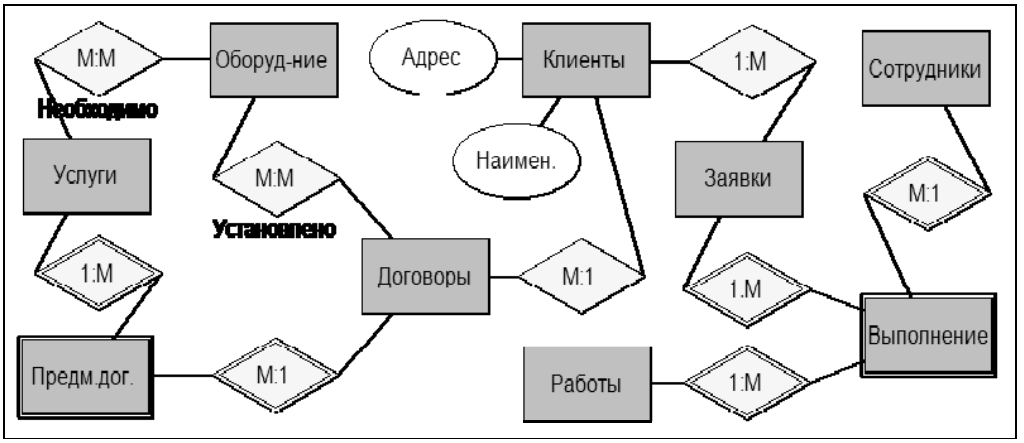


Рис. 2.12
ER-диаграмма пакета *Обработка заявок*

На третьем этапе, завершающем стадию эскизного проекта базы данных, все три локальные модели были объединены в единую концептуальную ER-модель. В процессе такого объединения был уточнен состав сущностей модели, унифицированы имена и типы данных их атрибутов, определены ключевые атрибуты сущностей, имена и кратности связей между ними, атрибуты связей.

Результат объединения локальных ER-моделей в единую ER-модель предметной области проектируемой АИС представлен на рисунке 2.13.

Контрольные вопросы и задания

1. Определите понятие «*связь*» как элемент ER-модели. Как классифицируются связи между сущностями? В каких случаях целесообразно использовать связи видов «*ассоциация*», «*агрегация*» и «*обобщение*»?

2. Определите понятие «*кратность связи*». Определите кратности связей между сущностями *Test*, *Task*, *Question* и *Answer* (рис. 2.5).

3. Определите понятие «*слабая сущность*». Какую роль выполняют сущности *Предмет договора* и *Услуги_Тарифных_Планов* (рис. 2.13)?

4. На рисунке 2.10 приведена ER-диаграмма пакета *Тарифные планы*. Перестройте эту диаграмму с использованием связей вида «*Обобщение*» для условий, когда категорий услуг всегда ровно три, а параметры услуг одной категории не могут входить в состав параметров услуг других категорий.

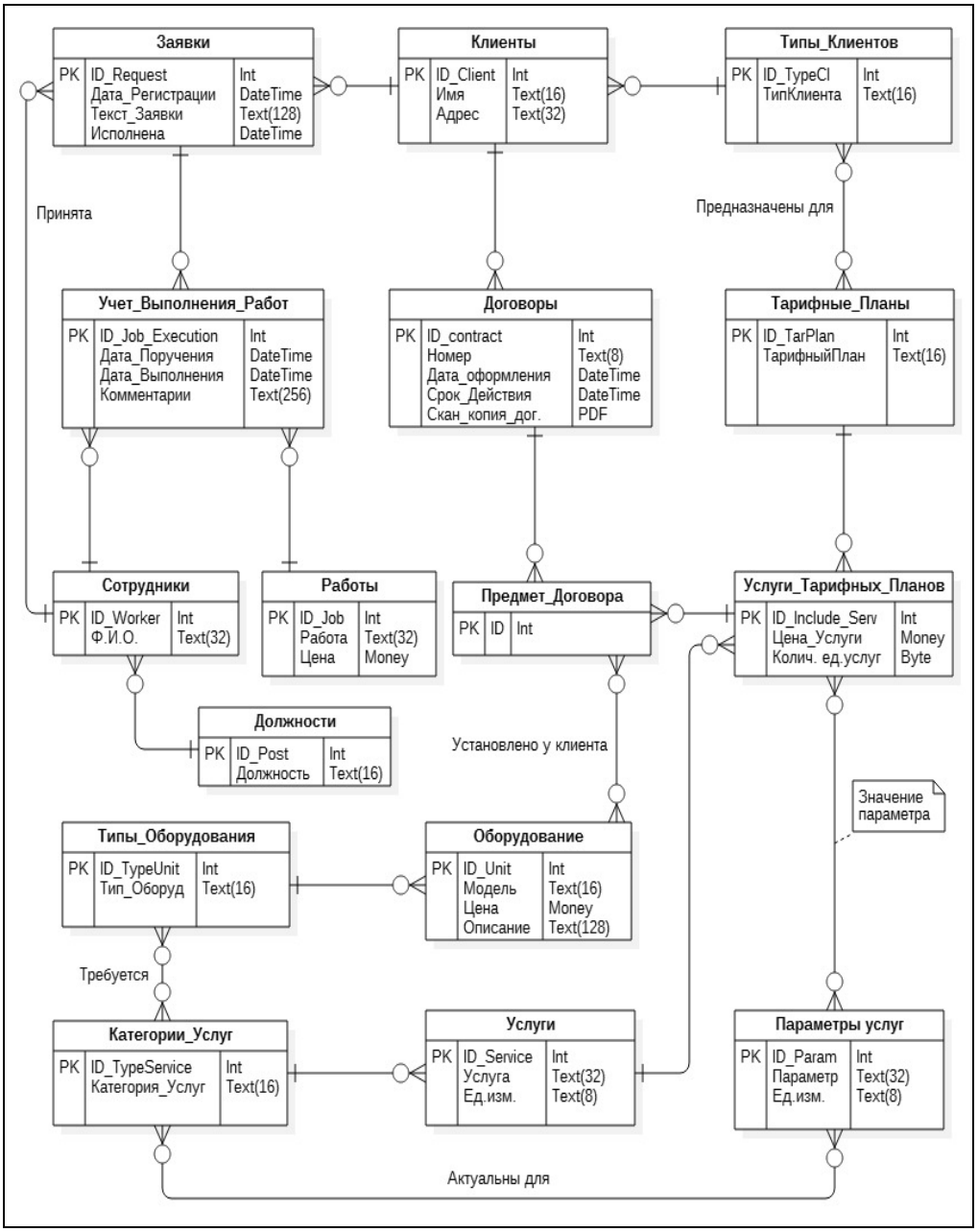


Рис. 2.13
Объединенная ER-диаграмма

ГЛАВА 4. ТЕХНИЧЕСКИЙ ПРОЕКТ. РАЗРАБОТКА РЕЛЯЦИОННОЙ МОДЕЛИ ДАННЫХ

Реляционная модель относится к категории логических моделей данных и формируется на следующей стадии разработки базы данных — стадии технического проекта (рис. 2.1). На этой стадии концептуальная ER-модель предметной области АИС преобразуется в схему (*R-модель* или *R-схему*) реляционной базы данных, которая затем получает программную реализацию на языке SQL в среде СУБД, поддерживающей функционирование АИС.

Процесс преобразования *ER-модели* в *R-схему* БД реализуется двумя последовательными этапами: вначале ER-модель преобразуется в исходную R-схему, а затем проводится анализ исходной R-схемы, по результатам которого может быть принято решение о необходимости ее дальнейшего преобразования (так называемой *нормализации*) с целью улучшения эксплуатационных характеристик проектируемой базы данных.

Первый этап такого преобразования достаточно формализован — он реализуется в соответствии с простыми правилами формирования реляционных структур данных (отношений) по описанию сущностей и связей ER-модели.

Второй этап связан с проведением процедуры нормализации, которая, хотя и базируется на разработанной Э. Коддом теории нормальных форм отношений, требует проведения неформального анализа семантики предметной области для выявления зависимостей между атрибутами сущностей, то есть, по существу, требует возврата на стадию эскизного проекта базы данных.

4.1. Преобразование ER-модели в исходную схему реляционной БД

На первом этапе разработки реляционной модели данных описание сущностей ER-модели и связей между ними преобразуется во множество схем взаимосвязанных отношений. Технология такого преобразования достаточно проста и может быть представлена последовательностью типовых шагов и правил, применяемых на каждом шаге.

Шаг 1. **Формирование схем отношений**

Каждая сущность ER-модели преобразуется в соответствующую схему отношения:

– отношению присваивается имя;

– каждый из агрегированных атрибутов сущности ER-модели (при наличии у сущности таких атрибутов) преобразуется во множество «атомарных» атрибутов соответствующего отношения;

– для каждого атрибута отношения определяются:

- имя атрибута;
- соответствующий скалярный тип данных (из множества типов данных, поддерживаемых СУБД);

– в необходимых случаях для атрибутов определяются:

- ограничения обязательности NOT NULL и значения по умолчанию DEFAULT;

• ограничения уникальности UNIQUE (для возможных ключей);

• проверяемые ограничения целостности CONSTRAINT … CHECK;

– определяется первичный ключ отношения:

• из числа возможных ключей отношения (атрибутов со свойством UNIQUE) выбирается первичный ключ;

• в необходимых случаях для этой цели в схему отношения добавляется более экономичный искусственный атрибут автоинкрементного типа данных;

• для единственного атрибута отношения, назначенного первичным ключом, задается ограничение целостности PRIMARY KEY.

В результате выполнения первого шага преобразования ER-модели сформировано множество схем отношений проектируемой базы данных, представляющих соответствующие объекты предметной области. Все атрибуты сущностей получили свою реализацию в атрибутах соответствующих отношений:

– агрегированные атрибуты (при их наличии) декомпозированы;

– для каждого атрибута определены *имя* и *скалярный тип данных*;

– свойства атрибутов отображены в ограничения целостности;

– в схеме каждого отношения задан единственный первичный ключ.

Полученный промежуточный результат еще не является полноценной реляционной моделью, так как не отображает информацию о связях между сущностями, представленную в ER-модели, и, как следствие, не позволяет идентифицировать кортежи отношений по их связям с другими кортежами.

Для интеграции разрозненных схем отношений в единую схему реляционной базы данных необходимо определить для каждой пары связанных отношений *ограничения ссылочной целостности* FOREIGN KEY, реализация которых базируется на концепции *внешних ключей* отношений.

Шаг 2. **Определение внешних ключей**

Каждая связь между сущностями ER-модели реализуется парой «*первичный ключ — внешний ключ*» соответствующих отношений R-модели, сформированных на предыдущем шаге преобразования.

Внешний ключ — это дополнительный атрибут, включаемый в схему ссылающегося (подчиненного, или дочернего) отношения для реализации ссылок на кортежи родительского (главного) отношения.

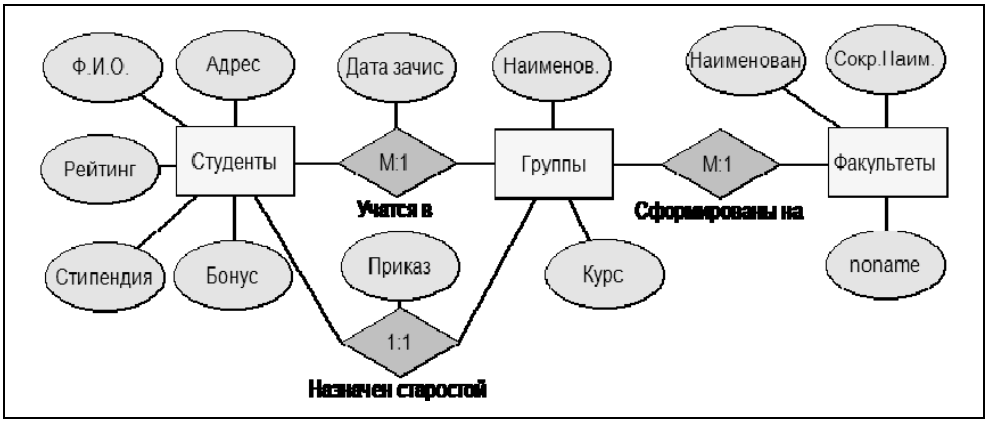
Внешний ключ дочернего отношения должен быть совместим по типу и домену с первичным ключом родительского отношения и может наследовать все его свойства, за исключением свойства уникальности. Значение внешнего ключа в кортежах дочернего отношения должно быть равным значению первичного ключа в связанном с ними кортеже родительского отношения, что, собственно, и позволяет реализовать ссылки между этими кортежами путем задания ограничений целостности FOREIGN KEY для внешних ключей дочерних отношений.

Для практического использования концепции внешних ключей необходимо конкретизировать понятия *родительского* и *дочернего* отношений, которые определяются на основе анализа кратности связей между сущностями ER-модели в соответствии с тремя *базовыми правилами*.

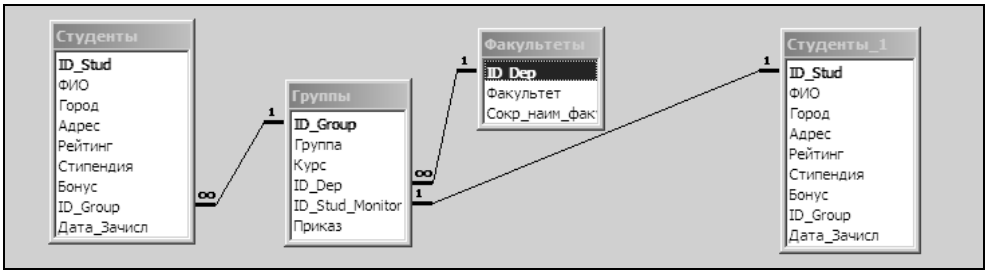
Правило № 1. Если между парой сущностей ER-модели установлена асимметричная связь кратности «***1:M***» («*один ко многим*»), то дочерним считается отношение, реализующее сущность, участвующую в связи со стороны ***M*** («*много*»), а другое отношение в этой паре считается родительским.

Правило № 2. Если между парой сущностей ER-модели установлена симметричная связь кратности «***1:1***», то дочерним считается отношение, имеющее потенциально меньшую мощность.

Правила реализации в R-модели связей кратностей «***1:M***» и «***1:1***» между сущностями ER-модели иллюстрируются на примере модели системы учета контингента студентов (рис. 2.14). Так как студенческих групп всегда меньше, чем студентов, для реализации связи «*назначен старостой*» кратности «***1:1***» в качестве дочернего принято отношение «*Группы*» с внешним ключом «*ID_Group_Monitor*».



а



б

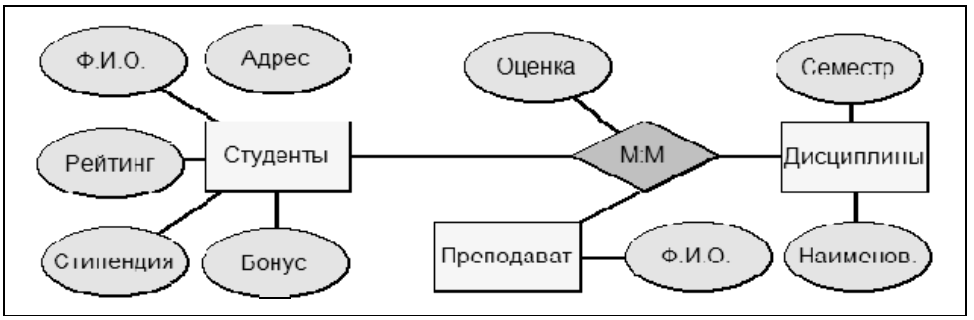
Рис. 2.14

Модель системы учета контингента студентов:
а — ER-диаграмма; *б* — схема реляционной БД.

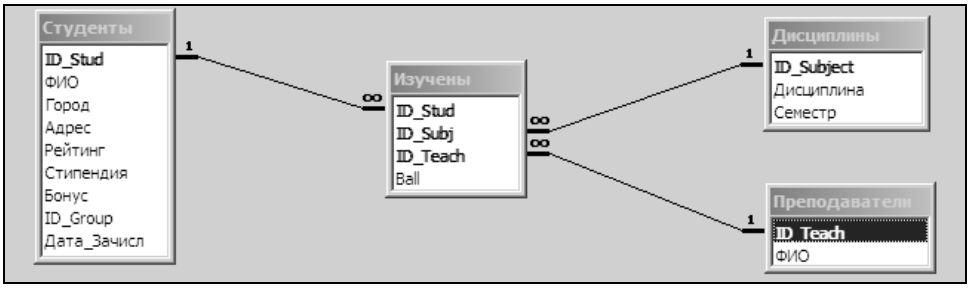
Правило № 3. Если между ***n*** сущностями ER-модели установлена связь кратности «***N:M***» («*многие ко многим*»), то схема базы данных дополняется ***n***-арным *ассоциативным* отношением, которое получает статус дочернего отношения со схемой, содержащей ***n*** внешних ключей, а все ***n*** отношений, участвующих в связи, получают статус родительских. В роли первичного ключа ассоциативного отношения выступает составной атрибут, включающий все ***n*** его внешних ключей — любое подмножество этого составного атрибута может

дублироваться в различных кортежах ассоциативного отношения, но все вместе они наделяются свойством уникальности и получают ограничение целостности PRIMARY KEY.

Такое решение вызвано «бедностью» реляционной модели данных, не имеющей естественных механизмов реализации связей множественной кратности: по существу, каждый экземпляр ***n***-арной связи кратности «***N:M***» заменяется ***n*** экземплярами связей кратности «***1:M***». Иллюстрация правила реализации связей множественной кратности приведена на рисунке 2.15 на примере модели системы учета успеваемости студентов.



а



б

Рис. 2.15

Модель системы контроля успеваемости студентов:
а — ER-диаграмма; *б* — схема реляционной БД.

Правила № 4 и *5* уточняют применение рассмотренных выше трех базовых правил для реализации иерархических и сетевых структур данных средствами реляционной модели.

Правило № 4 определяет способ реализации иерархических связей типа «обобщение» между сущностями ER-модели, когда дочерняя сущность представляет множество объектов, каждый из которых является частным случаем другого объекта, представленного родительской сущностью.

Дочерняя сущность наследует атрибуты и связи своей родительской сущности, но может иметь и свои собственные атрибуты и участвовать в связях с другими сущностями, не связанными с родительской сущностью (так, например, связаны дочерние сущности *Книги* и *Журналы* с родительской сущностью *Библиотечный фонд* на рисунке 2.16, представляющем фрагмент модели автоматизированной библиотечной системы).

Кратность такой связи — «***1:1***», из чего следует (согласно рассмотренному выше *базовому правилу № 2*), что внешние ключи должны быть добавлены в схемы дочерних отношений, так как мощность любого из них будет меньше мощности родительского отношения, которая равна сумме мощностей всех ее дочерних отношений.

Специфика реализации иерархических связей типа обобщение заключается в том, что *дочерние отношения не имеют собственных первичных ключей — их роль выполняют внешние ключи, наследующие значения первичных ключей связанных кортежей родительского отношения* и соответственно получающие ограничение целостности PRIMARY KEY (рис. 2.16*б*).

Правило № 5 определяет способ реализации иерархических и сетевых *унарных* связей, то есть связей между различными экземплярами одной сущности.

Между экземплярами сущности *Категории* (рис. 2.16*а*) задана иерархическая связь «*предок*» — «*потомок*» кратности «***1:M***» — это означает, что любой экземпляр сущности может быть «потомком» какого-либо одного экземпляра этой же сущности и одновременно «предком» одного или нескольких ее экземпляров.

Например, категории «*Учебники для вузов*» и «*Учебники для общеобразовательных школ*» могут быть «потомками» категории «*Учебная литература*», и при этом категория «*Учебники для вузов*» может быть «предком» для категорий «*Учебники по гуманитарным дисциплинам*» и «*Учебники по техническим дисциплинам*». Заметим, что наличие такого рода связи не исключает возможности экземплярам сущности не иметь ни «предков», ни «потомков».

Применение базового *правила № 1* к отношению, представляющему сущность *Категории*, потребует присвоения ему одновременно двух статусов: родительского и дочернего отношений. В результате в схему этого отношения (рис. 2.16*б*) будет добавлен внешний ключ *Код предка*, значением которого для кортежа-«потомка» будет значение первичного ключа *Код категории* из другого кортежа, представляющего кортеж-«предок» данного кортежа-«потомка». Такие внешние ключи называют *рефлексивными*.

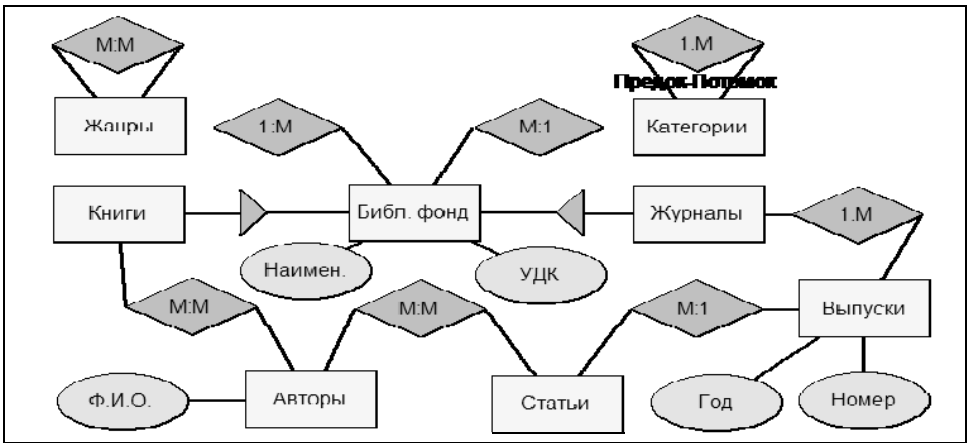
В качестве примера реализации унарной связи множественного наследования можно рассмотреть сущность *Жанры* (рис. 2.16*а*). Кратность этой связи определена как «***M:N***», что дает возможность любому жанру выступать как в роли «потомка», так и в роли «предка» любого количества других жанров. Например, жанр «*Фэнтези*» может быть объявлен «потомком» жанров «*Фантастика*» и «*Сказки народов мира*», и такое проектное решение может облегчить поиск нужной книги читателям, не отягощенным глубокими познаниями в области библиографии.

Кратность «***M:N***» такой связи позволяет применить к ней базовое *правило № 3*, согласно которому в базу данных добавляется ассоциативное бинарное отношение, схема которого сформирована из двух атрибутов — внешних ключей, представляющих первичные ключи двух связываемых кортежей основного отношения.

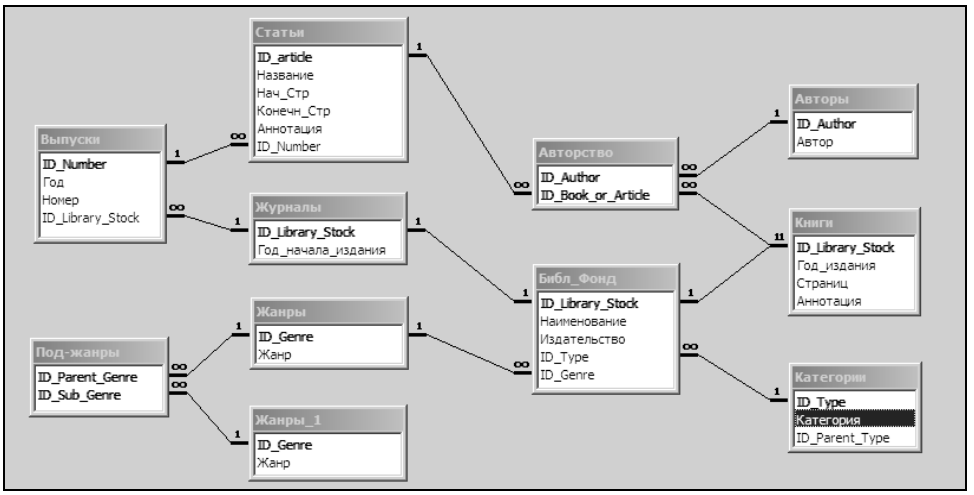
Так, например, схема ассоциативного отношения «*Под_жанры*» (рис. 2.16*б*) включает два таких атрибута — «*Код_жанра_предка*» и «*Код_жанра_потомка*».

Шаг 3. **Представление описательных атрибутов связей**

Правило № 6. Если для связи между сущностями ER-модели определены описательные атрибуты, то соответствующие атрибуты добавляются в схему того отношения, в которое был добавлен внешний ключ, реализующий эту связь (рис. 2.15 и 2.16).



а



б

Рис. 2.16
Модель библиотечного каталога:
а — ER-диаграмма; *б* — схема реляционной БД.

4.2. Пример разработки исходной схемы реляционной БД

В качестве примера продолжим рассмотрение подсистемы учета работы с клиентами интернет-провайдера, ER-диаграмма которой приведена на рисунке 2.13. Применяя рассмотренные выше *правила преобразования* к сущностям и связям этой ER-модели, получим R-модель, представленную на рисунке 2.17.

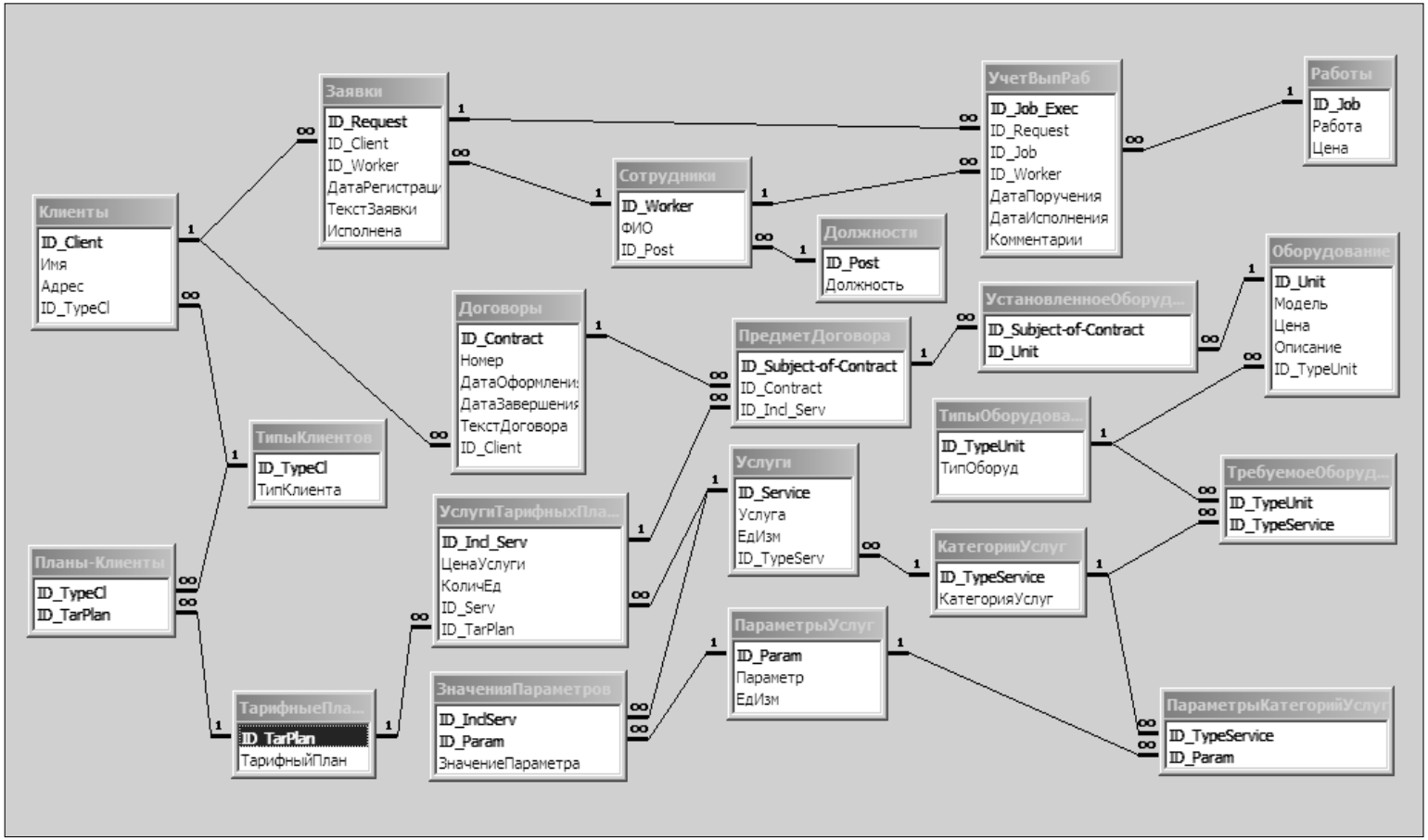


Рис. 2.17
Исходная R-модель подсистемы учета работы с клиентами интернет-провайдера

Контрольные вопросы и задания

1. Перечислите *правила преобразования* сущностей ER-модели в схемы соответствующих отношений R-модели данных.

2. Определите понятие «*внешний ключ отношения*». Как в R-модели реализуются *связи кратности* «***1:M***», «***1:M***» и «***M:N***», установленные между сущностями ER-модели?

3. Как в R-модели реализуются *иерархические связи* вида «*потомок*» — «*предок*», установленные между сущностями ER-модели (рис. 2.16)?

4. Как в R-модели отображаются *атрибуты связей* между сущностями ER-модели?

5. Какова роль отношений *Предмет договора* и *Услуги_Тарифных_Планов* в схеме БД, приведенной на рисунке 2.17?

6. *Задание*: напишите выражения реляционной алгебры, реализующие следующие запросы к базе данных, схема которой приведена на рисунке 2.17:

- список неисполненных заявок, поступивших от клиентов (*имя клиента, номер заявки* и *дата ее регистрации*);
- список тарифных планов, предназначенных для клиентов категории «корпоративные клиенты»;
- список оборудования, установленного у определенного клиента, заключившего договор (*имя клиента, № договора, тарифный план, услуга, модель оборудования*).

7. На рисунке 2.10 приведена ER-диаграмма пакета «*Тарифные планы*». *Задание*:

- перестройте эту диаграмму с использованием связей вида «*Обобщение*» для условий, когда категорий услуг всегда ровно три, а параметры услуг одной категории не могут входить в состав параметров услуг других категорий;
- используя правила преобразования ER-модели в R-модель, разработайте схему реляционной базы данных, соответствующую полученной ER-модели.

4.3. Нормализация реляционной базы данных

4.3.1. Аномальное поведение слабоструктурированных БД

Если ER-модель адекватно представляет свойства моделируемых объектов предметной области, то настолько же информационно-адекватной будет и реляционная БД, схема которой получена путем формальных преобразований описаний сущностей и связей ER-модели в схемы соответствующих отношений. Такая БД будет способной к реализации информационных запросов пользователей проектируемой АИС, что, однако, не исключает проявления различного рода аномалий в процессе ее эксплуатации.

Аномалии могут приводить в лучшем случае к излишнему дублированию данных и снижению производительности работы АИС, а в худшем — к нарушениям целостности базы данных и безвозвратной потере хранимой в ней информации при вполне корректном выполнении типичных модифицирующих

операций, таких, например, как вставка или удаление кортежей отношений или изменение значений их атрибутов.

Причины такого «аномального поведения» базы данных следует искать в недостатках структуры ее R-схемы, полученной путем формального преобразования ER-модели и унаследовавшей от нее все основные характеристики, в том числе и недостатки структурного характера. В отличие от R-схемы базы данных, процесс разработки ER-модели на стадии эскизного проекта формализован довольно слабо, всегда имеются альтернативные варианты получения его результатов, качество которых во многом определяется опытом разработчика.

Следующий (намеренно утрированный) пример иллюстрирует ситуацию, в которой некачественная декомпозиция предметной области при разработке ER-модели приводит к созданию схемы базы данных, которая, оставаясь информационно адекватной и удовлетворяющей всем базовым требованиям и ограничениям реляционной модели данных, оказывается совершенно непригодной к эксплуатации по причине явного проявления в ней аномалий пополнения, удаления и изменения данных.

Пусть разработчику (не отягощенному знаниями в области технологий проектирования программных систем) поставлена задача создания базы данных для учета успеваемости студентов.

Анализируя предметную область, разработчик получил доступ к «бумажному» документу «*Экзаменационная ведомость*» и обнаружил, что в этом документе содержится вся информация, необходимая для оперативного учета и анализа успеваемости студентов. В результате была создана ER-модель, содержащая единственную сущность «*Экзамены*», описательные атрибуты которой были скопированы с соответствующих информационных полей экзаменационной ведомости:

Экзамены (*Семестр, Группа, Дисципл., Студент, Препод., Дата, Оценка*)

В качестве возможного первичного ключа этой сущности может быть использован составной атрибут (*Семестр, Дисципл., Студент, Дата*), что позволяет уникально идентифицировать каждый экземпляр сущности и соответствует реальной ситуации с приемом экзаменов: в частности, включение атрибута *Дата* в состав первичного ключа позволит зарегистрировать факты и результаты повторной сдачи студентом экзаменов по дисциплинам.

Информационная адекватность такой модели не вызывает сомнений: каждый экземпляр этой сущности представляет один экзамен, принятый у одного студента по одной из дисциплин, изучаемых в одном семестре, при этом регистрируется дата проведения экзамена, полученная студентом оценка и реквизиты преподавателя, принявшего экзамен.

Аналитические возможности такой модели также соответствуют требованиям заказчика: модель позволяет формировать разнообразные рейтинговые списки студентов, их группировку по различным критериям и вычисление соответствующих интегральных показателей успеваемости.

На основании этой ER-модели была разработана R-схема базы данных, включающая единственное отношение с набором атрибутов, аналогичным ат-

рибутам сущности, и дополнительным искусственным атрибутом автоинкрементного типа данных, которому присвоен статус первичного ключа.

Схема такого отношения соответствует всем базовым требованиям реляционной модели данных:

– атрибуты отношения являются атомарными, им присвоены соответствующие скалярные типы данных;

– в отношении имеется первичный ключ, что гарантирует отсутствие в нем кортежей-дубликатов;

– в определении ограничений ссылочной целостности нет необходимости, так как R-схема содержит единственное отношение.

При всей простоте и информационной адекватности полученной R-схемы она не обеспечивает эффективного хранения информации по причине излишнего дублирования данных: например, фамилия студента будет дублироваться во всех кортежах, описывающих сданные этим студентом экзамены; также будут многократно повторяться наименование дисциплины и фамилия преподавателя.

Кроме этого, попытки выполнения операций, модифицирующих базу данных, приведут к проявлению следующих аномалий:

– *аномалия пополнения* — невозможно вставить кортеж, описывающий студента, еще не сдавшего экзамен, или дисциплину, по которой экзамен еще не сдавался, так как атрибуты, входящие в состав возможного ключа, не могут иметь неопределенных значений;

– *аномалия удаления* — удаление кортежа, описывающего некоторого студента (например, при его отчислении), может привести к потере информации о дисциплине (в том случае, если отчисленный студент был единственным, кто сдавал экзамен по этой дисциплине);

– *аномалия изменения* — если, например, студент(ка) меняет фамилию, то потребуется внести соответствующие изменения в несколько десятков кортежей.

Совсем другая ER-модель той же самой «экзаменационной ведомости», полученная в результате проведения более детальной декомпозиции предметной области на стадии эскизного проекта, представлена на рисунке 2.15*а*.

Схема реляционной БД (рис. 2.15*б*), сформированная на основе этой ER-модели, лишена рассмотренных выше недостатков первоначальной схемы, что избавит БД от многих аномалий в процессе ее эксплуатации.

4.3.2. Процедура нормализации отношений

Приведенный выше пример иллюстрирует проблемы, возникающие при эксплуатации слабоструктурированных баз данных, но не дает ответа на три базовых вопроса.

1. Как определить на стадии проектирования базы данных, будут ли проявляться аномалии в процессе ее эксплуатации?

2. Как преобразовать исходную схему отношения к форме, в которой проявление аномалий будет минимальным и маловероятным?

3. Как при таком преобразовании не потерять информационную адекватность исходной R-модели, унаследованную от ER-модели?

Ответы на эти вопросы дает так называемый *реляционный подход* к проектированию базы данных путем ее последовательной нормализации, в основе которого лежит *теория нормальных форм* отношений.

В теории определены 6 нормальных форм (НФ): 1-я НФ, 2-я НФ, 3-я НФ, НФБК (нормальная форма Бойса — Кодда, называемая также «усиленной третьей нормальной формой»), 4-я НФ и 5-я НФ, при этом каждая последующая НФ считается более сильной по сравнению с предыдущей. 1-я и 2-я НФ считаются слабыми нормальными формами, для большинства приложений оказывается достаточным приведение отношений базы данных к сильной НФБК.

Соответствие схемы отношения одной из сильных НФ дает определенные гарантии отсутствия аномалий в процессе эксплуатации базы данных, при этом чем сильнее эта НФ, тем менее вероятным будет проявление аномалий.

Теория нормальных форм отношений дает следующий ответ на первый из перечисленных выше вопросов: *проявление аномалий модификации данных в некотором отношении будет сведено к минимуму, если схема этого отношения находится в одной из сильных нормальных форм*. Метод определения НФ отношения базируется на результатах семантического анализа зависимостей между его атрибутами (п. 4.3.3): чем меньше таких зависимостей, тем в более сильной НФ находится отношение.

Эта же теория дает ответ и на второй вопрос*: для приведения отношения к более сильной НФ следует провести его декомпозицию на несколько взаимосвязанных отношений, в которых будут отсутствовать нежелательные зависимости между атрибутами*.

Декомпозиция отношений проводится путем многократного применения к ним реляционно-алгебраической операции проекции (п. 2.4.4.1) на соответствующие подмножества атрибутов.

Гарантией сохранения информационной адекватности схемы нормализованной базы данных является строгое следование *правилу декомпозиции без потерь* (п. 4.3.4) в процессе проведения декомпозиции отношений. Определение этого правила также дает теория нормальных форм отношений.

Нормализацией реляционной базы данных *называется такое информационно-эквивалентное преобразование ее схемы, в результате которой отношения, находящиеся в слабых НФ, декомпозируются на несколько взаимосвязанных отношений, каждое из которых находится в более сильной НФ,* что гарантирует отсутствие проявления аномалий (или, по крайней мере, сведение к минимуму их негативных последствий) в процессе эксплуатации БД.

Типичный алгоритм нормализации реляционной базы данных может быть представлен следующей циклической последовательностью этапов:

1) для каждого отношения исходной базы данных проводится анализ зависимостей между его атрибутами (п. 4.3.3), по результатам которого определяется, в какой из НФ находится это отношение (п. 4.3.5);

2) отношения, находящиеся в сильных НФ (обычно это НФБК и выше), сохраняют свои исходные схемы;

3) каждое отношение, находящееся в слабой НФ, декомпозируется на несколько взаимосвязанных отношений с учетом *правила декомпозиции без потерь* (п. 4.3.4);

4) для каждого из отношений, полученных на 3-м этапе нормализации, выполняются этапы с 1-го по 3-й до тех пор, пока в схеме БД не останется отношений, находящихся в слабых НФ.

4.3.3. Зависимости между атрибутами отношений

В теории нормальных форм отношений определены три вида зависимостей между атрибутами: это *функциональные* зависимости, на которых базируются определения первых четырех нормальных форм (от 1-й НФ до НФБК), *многозначные* зависимости, используемые при определении 4-й НФ, и зависимости *проекции-соединения*, определяющие принадлежность отношения к 5-й НФ.

В учебнике рассматриваются только функциональные и многозначные зависимости и, соответственно, только первые пять нормальных форм отношений, имеющих наибольшее практическое применение при проектировании реляционных баз данных.

Определение 1. ***Функциональная зависимость***

Пусть в схеме отношении **R** определены атрибуты **A** и **B** (простые или составные). *Атрибут **B** функционально зависит от атрибута **A***, если в любой момент времени во всех кортежах отношения **R** каждому значению **a** атрибута **A** соответствует ровно одно связанное с ним значение **b** атрибута **B**.

Для обозначения функциональных зависимостей (далее — ФЗ) будем использовать формулу вида «**A → B**», в левой части которой записывается определяющий атрибут, а в правой — зависимый. Формула читается как «*атрибут B функционально зависит от атрибута A*». Используются также и графические способы обозначения зависимостей, примеры которых приведены на рисунке 2.18.

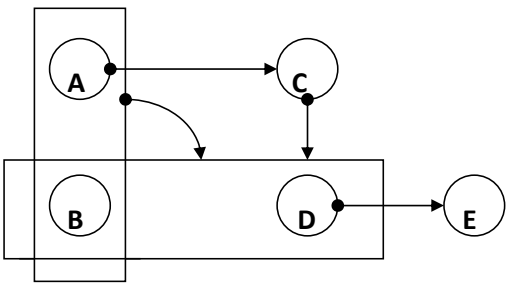


Рис. 2.18
Графические обозначения функциональных зависимостей
A → C; (A,B) → (B,D); (C → B,D); D → E

Для выявления зависимостей между атрибутами отношений на первом этапе нормализации БД необходимо провести семантический анализ свойств реальных объектов, описываемых соответствующими атрибутами.

При этом следует понимать, что наличие совпадающих пар значений атрибутов в разных кортежах отношения не является доказательством того, что между этими атрибутами существует ФЗ, но, с другой стороны, если в нескольких кортежах отношения одному и тому же значению атрибута **A** соответствуют различные значения атрибута **B**, можно считать доказанным утверждение об отсутствии ФЗ между этими атрибутами.

Множество ФЗ, выявленных по результатам семантического анализа моделируемого объекта, может оказаться избыточным, то есть может содержать зависимости, которые не вносят никакой новой информации, так как являются следствием наличия других зависимостей между атрибутами отношения. Перед тем как приступить к определению НФ отношения и его последующей нормализации, необходимо исключить избыточные ФЗ из исходного набора, получив так называемое *минимальное покрытие*.

Минимальным покрытием называют такое множество ФЗ между атрибутами отношения, которое не содержит избыточных ФЗ, выводимых из других ФЗ на основе рассмотренных выше правил вывода, и при этом удовлетворяет следующим требованиям:

– правая (зависимостная) часть любой ФЗ минимального покрытия является простым (не составным) атрибутом;

– левая часть любой ФЗ минимального покрытия может быть составным атрибутом, но при этом должна обладать свойством *минимальности* — это означает, что удаление любого атрибута из левой части любой ФЗ порождает множество ФЗ, не эквивалентное их исходному набору;

– удаление любой ФЗ из минимального покрытия порождает множество ФЗ, не эквивалентное их исходному набору.

Выявление избыточных ФЗ производится по результатам их формального анализа на основе следующих *правил вывода* ФЗ.

Правило № 1 (*рефлексивность*): если $\mathbf{B} \subseteq \mathbf{A}$, то $\mathbf{A} \rightarrow \mathbf{B}$ — подмножество атрибутов отношения функционально зависит от любого их множества, включающего данное подмножество. Другими словами, в левую (определяющую) часть существующей ФЗ можно добавлять любые атрибуты, при этом будут определены новые (корректные, хотя и избыточные) ФЗ.

Правило № 2 (*пополнение*): если существует ФЗ $(\mathbf{A},\mathbf{C}) \rightarrow \mathbf{B}$, то существует также и ФЗ $(\mathbf{A},\mathbf{C}) \rightarrow (\mathbf{B},\mathbf{C})$ — в правую (зависимостную) часть существующей ФЗ можно добавлять атрибуты, представленные в ее левой (определяющей) части, при этом будет определена новая (корректная, хотя и избыточная) ФЗ.

Правило № 3 (*транзитивность*): если существуют ФЗ $\mathbf{A} \rightarrow \mathbf{B}$ и $\mathbf{B} \rightarrow \mathbf{C}$, то существует также корректная, хотя и избыточная ФЗ $\mathbf{A} \rightarrow \mathbf{C}$.

Первые три правила (называемые *аксиомами Армстронга*) составляют базовый набор свойств ФЗ, на основе которых сформулированы еще четыре правила вывода ФЗ.

Правило № 4 (*декомпозиция*): если существует ФЗ **A→(B,C)**, то существуют ФЗ **A→B** и **A→C**. *Доказательство*. Согласно *правилу рефлексивности* существует ФЗ (**B,C**)**→B**, следовательно, по *правилу транзитивности* существует ФЗ **A→B**; аналогично, существует ФЗ (**B,C**)**→C** и, следовательно, существует ФЗ **A→C**.

Правило № 5 (*объединение*): если существуют ФЗ **A→B** и **A→C**, то существует также и ФЗ **A→(B,C)**. *Доказательство*. Согласно *правилу рефлексивности* из **A→C** следует (**A,B**)**→C**, а по правилу *пополнения* из (**A,B**)**→C** следует (**A,B**)**→(B,C)** и из **A→B** следует **A→(A,B)**. Следовательно, по *правилу транзитивности* **A→(B,C)**.

Правило № 6 (*композиция*): если существуют ФЗ **A→B** и **C→D**, то существует и ФЗ (**A,C**)**→(B,D)**. *Доказательство*. Последовательно применяя правила *рефлексивности* и *пополнения* к двум исходным ФЗ, получим корректные ФЗ (**A,C**)**→(B,C)** и (**B,C**)**→(B,D)**, из которых *по правилу транзитивности* может быть выведена ФЗ (**A,C**)**→(B,D)**.

Правило № 7 (*накопление* или *псевдотранзитивность*): если существуют ФЗ **A→(B,C)** и **B→D**, то существует и ФЗ **A→(B,C,D)**. *Доказательство*. Последовательно применяя правила *рефлексивности* и *пополнения* к ФЗ **B→D**, получим новую ФЗ (**B,C**)**→(B,C,D)**. Применяя *правило транзитивности* к исходной ФЗ **A→(B,C)** и новой ФЗ (**B,C**)**→(B,C,D)**, получим ФЗ **A→(B,C,D)**.

4.3.4. Правило декомпозиции без потерь

Как уже отмечалось, процедура нормализации отношения выполняется путем его декомпозиции на два или более отношения, схемы которых должны удовлетворять требованиям более сильных нормальных форм по сравнению со схемой исходного отношения.

Чтобы не потерять информационной адекватности исходной R-модели, декомпозиция нормализуемого отношения должна быть обратимой, то есть должна обеспечивать следующие результаты:

во-первых, при декомпозиции следует обеспечить существование связи между формируемыми отношениями, из чего следует, что в схемах этих отношений должны присутствовать общие атрибуты, образующие как минимум одну пару «*первичный ключ — внешний ключ*»;

во-вторых, в результате выполнения операции естественного соединения этих отношений должно быть сформировано отношение, равное (то есть совпадающее по схеме и составу кортежей) исходному отношению, подвергнутому процедуре декомпозиции.

Обратимость декомпозиции иллюстрируется следующим (абстрактным) примером. Пусть задано отношение **R{A,B,C}**, все атрибуты которого могут трактоваться как простые или составные. В результате декомпозиции этого отношения путем двукратного применения к нему реляционно-алгебраической *операции проекции* получены два отношения, связанные по их общему атрибуту: **B**: **R1 := R** PROJECT **(A,B)** и **R2 := R** PROJECT **(B,C)**.

Операция естественного соединения этих отношений порождает отношение **R3 := R1** NATURAL JOIN **R2** [ON **R1.B = R2.B**]. Проведенная декомпозиция будет считаться обратимой при условии, если **R3 = R**, в противном случае имеем некорректную декомпозицию исходного отношения, выполнение которой приведет к потере информационной адекватности R-модели.

Пример некорректной и необратимой *декомпозиции с потерями* иллюстрируется рисунком 2.19.

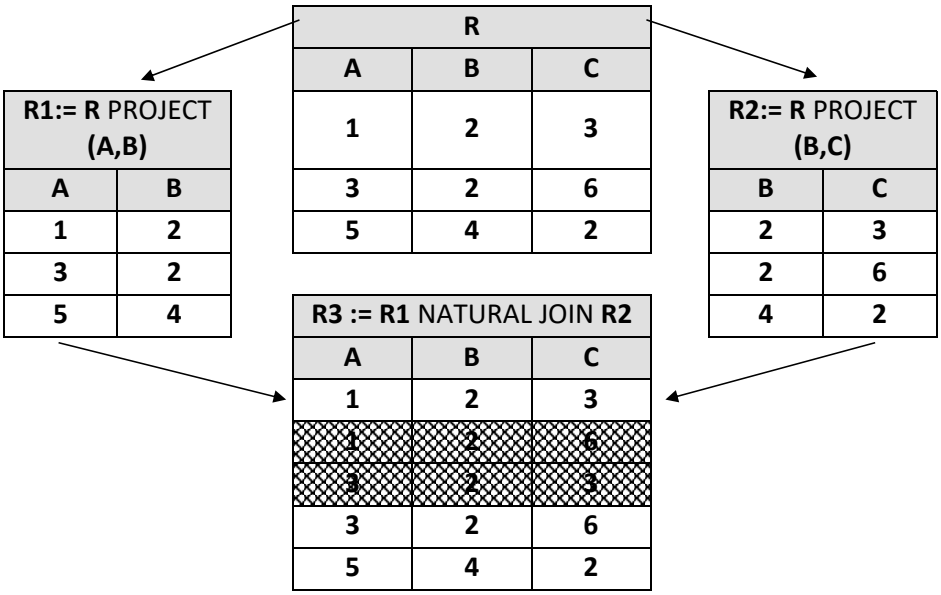


Рис. 2.19
Пример некорректной (необратимой) декомпозиции

Имеем некоторую реализацию отношения **R{A,B,C}** с определенными значениями атрибутов числового типа в трех его кортежах. Декомпозиция этого отношения путем его проецирования на два различных подмножества атрибутов приведет к формированию отношений **R1{A,B}** и **R2{B,C}**, связанных по их общему атрибуту **B**. В результате естественного соединения отношений **R1** и **R2** будет создано отношение **R3**, кортежи которого формируются путем сцепления каждого кортежа отношения **R1** с теми кортежами отношения **R2**, для которых выполняется условие естественного соединения **R2.B = R1.B**. Полученное отношение **R3** содержит два «лишних» кортежа и не совпадает с исходным отношением **R**.

Приведенный пример лишь иллюстрирует тот факт, что не всякая декомпозиция отношения является допустимой, но не позволяет практически применить его для проверки корректности декомпозиции. Заметим, что в распоряжении разработчика БД, приступающего к ее нормализации, имеются только схемы отношений, и при этом невозможно спрогнозировать все возможные реализации их кортежей с конкретными значениями атрибутов.

Правило декомпозиции без потерь (известное как *теорема Хита*) формулируется с использованием результатов анализа функциональных зависимостей между атрибутами отношения.

Определение 2. ***Правило декомпозиции без потерь***

Отсутствие потерь при декомпозиции отношения гарантируется в том случае, если от общего атрибута результирующих отношений функционально зависит хотя бы один атрибут из оставшихся.

Если в отношении **R{A,B,C}** существуют ФЗ **A→B** или **C→B**,
то **R = (R** PROJECT **(A,B))** NATURAL JOIN **(R** PROJECT **(B,C))**

Анализируя отношение **R** (рис. 2.19), можно заметить, что в двух кортежах этого отношения одинаковым значениям атрибута **B** соответствуют различные значения атрибутов **A** и **C.** Из этого можно сделать вывод об отсутствии ФЗ **A→B** и **C→B** и, как следствие, о некорректности проведения декомпозиции отношения по общему атрибуту **B**, что и подтверждается составом кортежей отношения **R3**.

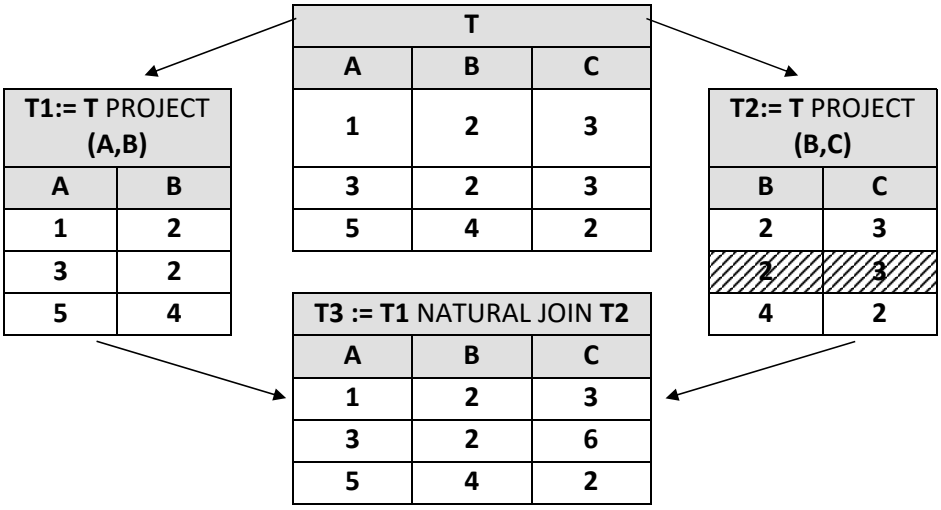


Рис. 2.20
Пример декомпозиции без потерь

В другом примере (рис. 2.20) в отношении **T** существует ФЗ **B→C** (наличие этой ФЗ выявлено в результате семантического анализа предметной области и только подтверждено соответствующими значениями атрибутов в выборке кортежей этого отношения — в двух его кортежах одинаковым значениям атрибута **C = 3** соответствуют одинаковые значения атрибута **B = 2**).

В соответствии с правилами выполнения реляционной операции проекции, кортеж-дубликат удаляется из отношения **T2**, что позволяет получить в результате естественного соединения отношений **T1** и **T2** отношение **T3**, идентичное исходному отношению **T**.

4.3.5. Нормальные формы отношений

Определение 3. ***1-я нормальная форма***

Отношение находится в 1-й НФ, если оно удовлетворяет базовым ограничениям реляционной модели данных: все атрибуты отношения атомарны и в отношении отсутствуют кортежи-дубликаты.

Отсутствие кортежей-дубликатов требует наличия в схеме отношения хотя бы одного *возможного ключа* — простого или составного атрибута, обладающего свойством уникальности и однозначно идентифицирующего кортеж. В отношении, находящемся в 1-й НФ, существуют ФЗ каждого из его атрибутов от всех возможных ключей.

Определение 4. ***Полная функциональная зависимость***

Функциональная зависимость **A→B** называется полной, если атрибут **B** функционально не зависит от любого подмножества атрибута **A**.

Определение 5. ***Детерминант***

Детерминантом называется левая (определяющая) часть *полной функциональной зависимости*.

Определение 6. ***Ключевые и неключевые атрибуты***

Атрибуты отношения, входящие в состав возможных ключей, называются *ключевыми* (или *основными*) атрибутами; остальные атрибуты отношения называются *неключевыми* (или *неосновными*).

Определение 7. ***2-я нормальная форма***

Отношение находится во 2-й НФ, если оно находится в 1-й НФ и существуют *полные ФЗ* каждого *неключевого* атрибута от всех *возможных ключей* этого отношения.

Очевидно, что если все возможные ключи отношения атомарны, то это отношение уже находится во 2-й НФ.

Если отношение не находится во 2-й НФ, следует провести такую его декомпозицию, при которой пары атрибутов, участвующих в неполных зависимостях, будут выделены в отдельные отношения.

Определение 8. ***Транзитивная функциональная зависимость***

В отношении **R{A,B,C}** существует транзитивная ФЗ **A→B**, если имеются ФЗ **A→C** и **C→B** и при этом отсутствует ФЗ **C→A**.

Определение 9. ***3-я нормальная форма***

Отношение находится в 3-й НФ, если оно находится во 2-й НФ и при этом в отношении отсутствуют *транзитивные ФЗ неключевых* атрибутов от любого из *возможных ключей*.

Заметим, что *отсутствие любых ФЗ между неключевыми атрибутами* отношения гарантирует отсутствие транзитивных зависимостей его неключевых атрибутов от возможных ключей. Следовательно, для приведения к 3-й НФ отношения, находящегося во 2-й НФ, необходимо провести такую его декомпозицию, при которой пары взаимозависимых неключевых атрибутов будут выделены в отдельные отношения.

Определение 10. ***Нормальная форма Бойса — Кодда*** **(НФБК)**

Отношение находится в НФБК, если оно находится в 3-й НФ и при этом каждый детерминант отношения является его возможным ключом.

В отношении, находящемся в 3-й НФ, гарантированно отсутствуют неполные ФЗ (*определение 4*) неключевых атрибутов от возможных ключей, а также любые ФЗ между неключевыми атрибутами. При этом, однако, не исключается наличие ФЗ между ключевыми атрибутами, входящими в состав возможного ключа, что может приводить к проявлению различных аномалий в процессе эксплуатации БД. НФБК устраняет этот «структурный недостаток» схемы отношения, запрещая наличие в отношении атрибутов-детерминантов (*определение 5*), являющихся «частями» составных возможных ключей.

Для приведения к НФБК отношения, находящегося в 3-й НФ, следует провести такую его декомпозицию, при которой пары взаимосвязанных ключевых атрибутов (при наличии в этом отношении внутриключевых связей) будут выделены в отдельные отношения. Заметим, что если в отношении, находящемся в 3-й НФ, отсутствуют составные возможные ключи, это отношение уже находится и в НФБК.

4-я нормальная форма базируется на концепции многозначных зависимостей между атрибутами отношения.

Определение 11. ***Многозначная зависимость***

В отношении **R{A,B,C}** существует многозначная зависимость атрибута **A** от атрибута **B** (обозначается как **A→→B**) в том случае, если множество значений атрибута **B**, соответствующее паре значений атрибутов **A** и **C**, зависит только от атрибута **A** и не зависит от атрибута **C**.

Определение 12. ***4-я нормальная форма***

Отношение **R{A,B,C}** находится в 4-й НФ, если при существовании *многозначной зависимости* **A →→ B** существуют *функциональные зависимости* всех остальных атрибутов этого отношения от атрибута **A**.

Приведение к 4-й НФ отношения, находящегося в НФБК, также производится путем его декомпозиции, исключающей многозначные зависимости в каждом из производных отношений. Соответствующие примеры рассмотрены в п. 4.3.6.

4.3.6. Пример нормализации реляционной базы данных

Продолжим рассмотрение модели экзаменационной ведомости, использованной ранее (п. 4.3.1), для иллюстрации аномального поведения слабоструктурированной базы данных.

Пусть задана следующая схема исходного отношения R_0:

R_0 (*Сем., Дисц., Студ., Курс, Спец., Группа, Препод., Дата, Оценка*)

Анализируя реальные процессы проведения экзаменационных сессий (с учетом несущественных упрощений этих процессов, вполне допустимых в условиях рассмотрения демонстрационного примера), получим следующий *исходный набор ФЗ* между атрибутами отношения R_0.

ФЗ-1: ***Сем.*** $\rightarrow$ ***Курс*** — номер семестра однозначно определяет номер курса, на котором учится группа студентов.

ФЗ-2: ***Группа*** $\rightarrow$ (***Курс, Спец.***) — наименование группы однозначно определяет номер курса и специальность для этой группы.

ФЗ-3: ***Студ.*** $\rightarrow$ (***Группа, Спец.***) — студент может входить в состав только одной группы и обучаться только по одной специальности.

ФЗ-4: (***Сем., Спец., Дисц., Группа***) $\rightarrow$ ***Препод.*** — не допускается прием экзамена в одной группе одной специальности по одной дисциплине в одном семестре несколькими преподавателями.

ФЗ-5: (***Сем., Группа., Дисц.***) $\rightarrow$ (***Дата, Препод.***) — группа сдает экзамен по дисциплине строго в соответствии с расписанием экзаменационной сессии — одному преподавателю и в один день.

ФЗ-6: (***Студ., Дисц., Сем.***) $\rightarrow$ (***Дата, Препод., Оценка***) — не допускается индивидуальная (в том числе повторная и другому преподавателю) сдача экзаменов студентами вне группового расписания.

На следующем этапе нормализации отношения R_0 следует сформировать *минимальное покрытие* ФЗ (п. 4.3.3), исключив из исходного набора избыточные ФЗ, которые могут быть получены из других ФЗ применением к ним соответствующих *правил вывода* (п. 4.3.4).

Зависимости **ФЗ-2, ФЗ-3, ФЗ-5** и **ФЗ-6** не соответствуют одному из требований, предъявляемых к минимальному покрытию, так как *содержат составные атрибуты* в своих правых частях. Исключаем эти ФЗ из минимального покрытия, заменив их новыми ФЗ, полученными путем применения к исключаемым зависимостям *правила декомпозиции*. Заметим, что новые ФЗ не противоречат семантике предметной области.

Декомпозиция **ФЗ-2** — ***Группа*** $\rightarrow$ (***Курс, Спец.***):

ФЗ-2.1 — ***Группа*** $\rightarrow$ ***Курс***;

ФЗ-2.2 — ***Группа.*** $\rightarrow$ ***Спец***.

Декомпозиция **ФЗ-3** — ***Студ.*** $\rightarrow$ (***Группа, Спец.***):

ФЗ-3.1 — ***Студ.*** $\rightarrow$ ***Группа***;

ФЗ-3.2 — ***Студ.*** $\rightarrow$ ***Спец.***

Зависимость **ФЗ-3.2** является избыточной, так как может быть получена из **ФЗ-3.1** и **ФЗ-2.2** по *правилу транзитивности* — исключаем **ФЗ-3.2** из минимального покрытия.

Декомпозиция **ФЗ-5** — **(*Сем., Группа., Дисц.*) → (*Дата, Препод.*)**:

ФЗ-5.1 — **(*Сем., Группа., Дисц.*) → *Дата***;

ФЗ-5.2 — **(*Сем., Группа., Дисц.*) → *Препод.***

Декомпозиция **ФЗ-6** — **(*Студ., Дисц., Сем.*) → (*Дата, Препод., Оценка*)**:

ФЗ-6.1 — **(*Студ., Дисц., Сем.*) → *Дата***;

ФЗ-6.2 — **(*Студ., Дисц., Сем.*) → *Препод.***;

ФЗ-6.3 — **(*Студ., Дисц., Сем.*) → *Оценка***.

Зависимости **ФЗ-4** и **ФЗ-5.2** являются избыточными, так как могут быть выведены из **ФЗ-6.2** по *правилу рефлексивности*, поэтому исключаем их из минимального покрытия.

Также исключаем из минимального покрытия и зависимость **ФЗ-5.1 —** она выводится из **ФЗ-6.1** по этому же правилу.

В результате получим следующее неизбыточное множество функциональных зависимостей отношения R_0, составляющих *минимальное покрытие*:

ФЗ-1 — *Сем.* → *Курс*;

ФЗ-2.1 — ***Группа* → *Курс***;

ФЗ-2.2 — ***Группа.* → *Спец***.;

ФЗ-3.1 — ***Студ.* → *Группа***;

ФЗ-6.1 — **(*Студ., Дисц., Сем.*) → *Дата***;

ФЗ-6.2 — **(*Студ., Дисц., Сем.*) → *Препод.***;

ФЗ-6.3 — **(*Студ., Дисц., Сем.*) → *Оценка***.

Анализируя правые (определяющие) части зависимостей, вошедших в минимальное покрытие, можно сделать вывод о том, что единственным *возможным ключом* отношения R_0 является составной атрибут (***Студ., Дисц., Сем.***), следовательно (*определение 6*), атрибуты ***Студ., Дисц., Сем.*** являются *основными атрибутами*, а остальные атрибуты — ***Курс, Спец., Группа, Препод., Дата, Оценка*** — получают статус *неосновных атрибутов* отношения.

R_0 (***Студ., Дисц., Сем., Курс, Спец., Группа, Препод., Дата, Оценка***)

1-я нормальная форма

Наличие в отношении R_0 первичного ключа, а также тот факт, что все атрибуты этого отношения атомарны (то есть не предполагается использование элементов их внутренней структуры в запросах к БД), позволяют сделать вывод о том (*определение 3*), что ***отношение R_0 находится в первой нормальной форме***.

2-я нормальная форма

Отношение ***R_0 не находится во второй нормальной форме***, этому препятствует наличие *неполных зависимостей* **ФЗ-1** и **ФЗ-3.1** неосновных атрибутов ***Курс*** и ***Группа*** от компонентов первичного ключа ***Сем.*** и ***Студ.*** (*определе-*

ние 7), а также наличие *неполной транзитивной зависимости* между атрибутами ***Студ.*** → ***Группа*** → ***Спец.*** (*определение 8*).

Для приведения БД ко 2-й НФ следует декомпозировать отношение R_0 таким образом, чтобы атрибуты, участвующие в неполных зависимостях, были выделены в отдельные отношения.

Выполним операции проецирования отношения R_0 на соответствующие подмножества его атрибутов:

R_1 := R_0 PROJECT **(*Сем., Курс*)**;

R_2 := R_0 PROJECT **(*Студ., Группа, Спец.*)**;

R_3 := R_0 PROJECT **(*Студ., Дисц., Сем., Препод., Дата, Оценка*)**.

При выполнении этих операций соблюдено *правило декомпозиции без потерь* (*определение 2*), так как в декомпозируемом отношении R_0 существуют зависимости некоторых его атрибутов от общих атрибутов пар результирующих отношений R_1 — R_3 и R_2 — R_3.

Наличие общих атрибутов в парах полученных отношений позволяет «собрать» исходное отношение операцией их естественного соединения:

R_0 := (R_1 NATURAL JOIN **(R_2** NATURAL JOIN **R_3** ON **R_2.Студ.** = **R_2.Студ.)**
ON **R_1.Сем.** = **R_3.Сем.)**

Бинарное отношение **R_1 (*Сем., Курс*)** находится во 2-й НФ (как, впрочем, и во всех остальных, более сильных НФ, по причине своей бинарности).

Отношение **R_2 (*Студ., Группа, Спец.*)** не имеет составных возможных ключей, следовательно, это отношение также находится во 2-й НФ.

В отношении **R_3 (*Студ., Дисц., Сем., Препод., Дата, Оценка*)** остался составной первичный ключ, но это отношение тоже находится во 2-й НФ, так как в нем отсутствуют неполные функциональные зависимости между неосновными атрибутами от ключа отношения.

3-я нормальная форма

Отношение **R_3** уже находится в 3-й НФ (*определение 9*), так как в нем отсутствуют взаимозависимости между неосновными атрибутами и, как следствие, отсутствуют и транзитивные зависимости неосновных атрибутов от ключа.

Отношение **R_2** не находится в 3-й НФ, так как в нем существует транзитивная зависимость неосновного атрибута ***Спец.*** от первичного ключа ***Студ.*** **(*Студ.* → *Группа* → *Спец.*)**. Декомпозируем это отношение на два бинарных отношения **R_4 (*Студ., Группа*)** и **R_5 (*Группа, Спец.*)**, связанных по общему атрибуту ***Группа***. Каждое из этих отношений находится в 3-й, а также и во всех более сильных НФ.

Нормальная форма Бойса — Кодда

В отношении **R_3 (*Студ., Дисц., Сем., Препод., Дата, Оценка*)**, находящемся в 3-й НФ, единственным детерминантом является составной ключ **(*Студ., Дисц., Сем.*)**, так как в этом отношении отсутствуют взаимозависимости между основными атрибутами и зависимости основных атрибутов от неосновных. Следовательно (*определение 10*), отношение **R_3** находится в НФБК.

В результате проведенной нормализации исходное отношение **R_0 (*Студ., Дисц., Сем., Курс, Спец., Группа, Препод., Дата, Оценка*)**, находившееся только в 1-й НФ, было декомпозировано на четыре взаимосвязанных отношения, каждое из которых приведено к НФБК:

R_1 (*Сем., Курс*) — номера семестров по курсам обучения;

R_3 (*Студ., Дисц., Сем., Препод., Дата, Оценка*) — успеваемость студентов;

R_4 (*Студ., Группа*) — распределение студентов по группам;

R_5 (*Группа, Спец.*) — распределение групп по специальностям.

На каждом шаге декомпозиции строго выполнялось *правило декомпозиции без потерь*, общие атрибуты результирующих отношений составили пары «первичный ключ — внешний ключ», что позволяет восстановить исходное отношение последовательным выполнением операций естественного соединения над результирующими отношениями:

R_0 := (R_1 NATURAL JOIN **R_3** ON **R_1.Сем.** = **R_3.Сем.)** NATURAL JOIN

(R_4 NATURAL JOIN **R_5** ON **R_4.Группа** = **R_5.Группа)**

ON **R_4.Студ.** = **R_3.Студ.)**

4-я нормальная форма

Несмотря на приведение схемы БД к достаточно сильной НФБК, при формировании кортежей отношения **R_3** не исключено проявление *аномалии пополнения* — регистрация результатов сдачи экзамена по одной дисциплине потребует дублирования наименования этой дисциплины столько раз, сколько студентов изучали эту дисциплину. Причина такого дублирования — отсутствие 4-й НФ в этом отношении.

В отличие от рассмотренных выше нормальных форм отношений, 4-я НФ базируется на концепции *многозначных зависимостей* между атрибутами.

Если считать корректным утверждение о том, что каждый студент обязан сдавать экзамены по всем дисциплинам, соответствующим учебному плану его специальности, то в отношении **R_3** существует *многозначная зависимость* **Студ.→→Дисц.** (*определение 11*), так как множество значений атрибута **Дисц.**, соответствующее паре значений атрибутов **Студ.** и **Сем.**, зависит только от атрибута **Студ.** и не зависит от атрибута **Сем.**

В соответствии с *определением 12* отношение **R_3** *не находится в 4-й НФ*, так как в этом отношении при существовании *многозначной зависимости* **Студ.→→Дисц.** *не существует функциональных зависимостей* всех остальных атрибутов от атрибута **Студ.** (в противном случае отношение **R_3** не находилось бы даже во 2-й НФ, требующей наличия *полных* функциональных зависимостей всех неосновных атрибутов от составного ключа).

Для приведения схемы БД к 4-й НФ декомпозируем отношение **R_3** на четыре отношения: отношения-справочники **R_6**, **R_7** и **R_8** и связывающее их отношение **R_9**:

R_6 (*ID_Stud, Студ*) — контингент студентов;

R_7 (*ID_Disc., Дисц.*) — перечень дисциплин;

R_8 (*Сем.*) — семестры;

R_9 (*ID_Stud, ID_Disc., Сем., Препод., Дата, Оценка*) — оценки.

Отношения **R_6 (*ID_Stud, Студ*)** и **R_8 (*Сем.*)** должны быть удалены из схемы БД, так как они дублируют имеющиеся отношения **R_4 (*Студ., Группа*)** и **R_1 (*Сем., Курс*)**.

Искусственные ключи ***ID_Stud*** и ***ID_Disc.*** (числового типа данных) добавлены в схемы отношений в дополнение к естественным возможным ключам (текстового типа) для снижения негативного эффекта от неизбежного дублирования данных в кортежах отношения **R_9**, их наличие не повлияет на результаты проведенного анализа функциональных зависимостей и декомпозиции отношений.

Результирующая схема нормализованной базы данных, приведенной к 4-й НФ, показана на рисунке 2.21.

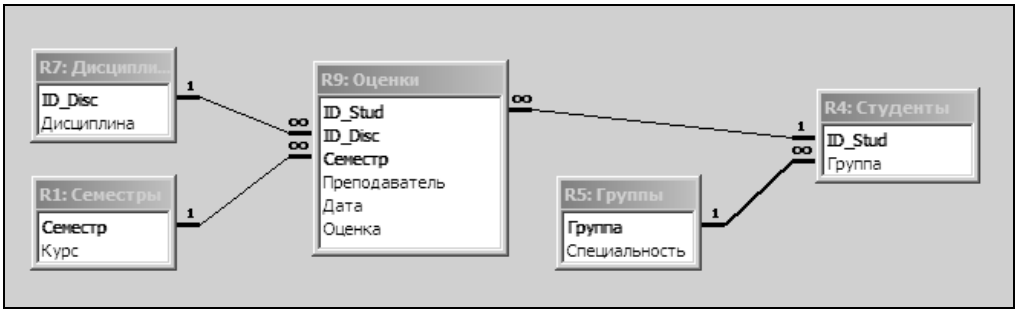


Рис. 2.21

Схема БД, полученная в результате нормализации исходного отношения **R_0**

Завершая рассмотрение технологий проектирования реляционных баз данных, попытаемся сравнить два подхода к реализации этих технологий, долгое время развивавшихся параллельно в условиях конкуренции друг с другом.

Последний из рассмотренных примеров иллюстрирует классический «реляционный» подход к разработке логической модели данных путем нормализации так называемого универсального отношения, включающего атрибуты всех объектов предметной области. Теория нормальных форм отношений разрабатывалась применительно именно к такому подходу, не требовавшему предварительной декомпозиции предметной области на начальных стадиях проекта базы данных.

Как видно из рисунка 2.21, в результате нормализации универсального отношения была сформирована реляционная модель, фактически отражающая структуру сущностей предметной области и связей между ними, но такое решение было получено путем формального анализа и преобразования зависимостей между атрибутами универсального отношения на основе правил их вывода, базирующихся на трех *аксиомах Армстронга*, и последовательной декомпозиции отношений с использованием правила декомпозиции без потерь, известного как *теорема Хита*.

Следует заметить, что применение *формализованного метода нормализации* требует проведения *неформального семантического анализа* предметной области для выявления исходного множества зависимостей между атрибутами универсального отношения.

Параллельно с классическим реляционным подходом разрабатывался инфологический подход, предполагающий проведение семантического анализа на начальной стадии проекта базы данных, по результатам которого формируется объектная структура предметной области, то есть выделяются и именуются все объекты, выявляются и специфицируются их существенные свойства, классифицируются межобъектные связи. Схемы исходных отношений проектируемой БД, получаемые путем копирования структур соответствующих сущностей ER-модели, как правило, уже находятся в 3-й НФ (а иногда и в НФБК), что существенно сокращает процедуру нормализации, а в ряде случаев сводит ее к простой проверке «нормальности» схемы БД.

Критерием завершения процесса нормализации БД является приведение всех входящих в нее отношений к одной из сильных нормальных форм. В большинстве случаев оказывается достаточно НФБК, гарантирующей отсутствие перечисленных выше аномалий, однако не исключающей дублирования данных в случае наличия многозначных зависимостей между атрибутами. Для устранения многозначных зависимостей отношение приводят к 4-й НФ.

На последнем этапе нормализации проводится контроль наличия в схеме базы данных и удаление дублирующих отношений, атрибуты которых уже присутствуют в других отношениях (как это и было сделано с отношениями $\mathbf{R_6}$ и $\mathbf{R_8}$ в рассмотренном выше примере).

Контрольные вопросы и задания

1. Дайте определения функциональной и многозначной зависимостей.
2. Сформулируйте аксиомы Армстронга и докажите на их основе правила вывода функциональных зависимостей между атрибутами отношения.
3. Что называют минимальным покрытием и каковы требования к нему? Получите свою версию минимального покрытия для отношения $\mathbf{R_0}$ из рассмотренного выше примера нормализации.
4. Сформулируйте теорему Хита. Что гарантирует отсутствие потерь при декомпозиции отношения?
5. Приведите определения и собственные примеры нормальных форм отношений.

ГЛАВА 5. ПРОЕКТНЫЙ ПРАКТИКУМ

5.1. Общие методические указания

Основная задача проектного практикума — ознакомление с технологией проектирования реляционных БД и получение опыта использования CASE-средств в процессе выполнения учебных программных проектов.

Структура и содержание. Проект базы данных студентом выполняется индивидуально в соответствии с утвержденной темой проекта. Типовые варианты тем приведены в п. 3.4.3, студент вправе предложить собственный вариант темы проекта, который должен быть согласован с преподавателем.

В составе проекта выполняются начальные этапы разработки несложной базы данных, результаты которых представляются соответствующими моделями: UML-модель вариантов использования; UML-диаграмма пакетов; ER-модель; R-модели — исходная и нормализованная; схема реляционной БД, реализующая нормализованную R-модель в среде одного из доступных серверов баз данных.

Проектная документация оформляется в форме технического отчета, включающего все перечисленные выше графические диаграммы и описание процесса нормализации исходной R-модели.

Защита проекта проводится в форме публичного доклада по материалу представленного отчета. В процессе защиты оценивается полнота и качество выполнения заданий, грамотность использования инструментальных средств, правильность и обоснованность выводов по результатам работы, качество оформления графических материалов.

Программное обеспечение. Проектная часть выполняется с использованием UML-ориентированных CASE-средств, реализация схемы БД — в любом из доступных серверов баз данных. Решение о выборе ПО студент принимает самостоятельно.

Основная часть работы над проектом студентом выполняется самостоятельно. На практических занятиях рассматриваются учебные примеры выполнения и документирования проекта базы данных, обсуждается технология разработки моделей, заслушиваются сообщения разработчиков и анализируются представленные ими результаты выполнения этапов проекта.

Занятие № 1. Подготовительный этап:

– обсуждение примеров выполнения проектов баз данных, рассмотренных в п. 3.5 и 4.2 учебника;

– согласование и утверждение тем учебных проектов.

Занятие № 2. Стадия технического задания:

– определение состава пользователей АИС и базовых функций, реализуемых АИС в интересах пользователей;

– разработка UML-диаграммы вариантов использования.

Занятие № 3. Стадия эскизного проекта:

– разработка UML-диаграммы пакетов АИС;

– разработка ER-моделей для локальных представлений (пакетов);

– объединение локальных ER-моделей.

Занятие № 4. Стадия технического проекта:

– разработка исходной реляционной модели данных;

– нормализация исходной реляционной модели;

– программная реализация схемы реляционной БД.

Занятие № 5. Защита проектов с демонстрацией результатов.

5.2. Типовые варианты тем учебных проектов

Тема № 1. УПРАВЛЕНИЕ РАБОТАМИ		
Пользователи		*Автоматизируемые бизнес-процессы*
руководители; менеджеры; исполнители; клиенты (заказчики)		ведение клиентской базы; оперативный учет поступления заказов от клиентов; кадровый учет; планирование и распределение работ между исполнителями; мониторинг и контроль исполнения работ; формирование аналитической и отчетной документации
Варианты заданий		*Дополнительные требования*
1.1	Управление проектами	структуризация проектов; специализация исполнителей; повышение квалификации и профессиональная переподготовка сотрудников
1.2	Интернет-провайдер	call-центр и техподдержка клиентов; услуги и тарифные планы; категории клиентов; оборудование, установленное у клиентов
1.3	Компьютер-сервис	прайс-лист (сервис, ремонт, установка и настройка программного обеспечения); прием заявок; выездные работы; доставка оборудования
1.4	Малое промышленное предприятие (по отраслям — по выбору студента)	номенклатура изделий; технологическая документация; складской учет материалов и комплектующих; складской учет готовой продукции
1.5	Фермерское сельхозпредприятие (по отраслям — по выбору студента)	номенклатура производства; учет основных фондов; договоры с заказчиками и поставщиками; складской учет готовой продукции
Тема № 2. УПРАВЛЕНИЕ ОБРАЗОВАНИЕМ		
Пользователи		*Автоматизируемые бизнес-процессы*
руководители; преподаватели (учителя); технические сотрудники; студенты (учащиеся, слушатели)		управление контингентом студентов (учащихся, слушателей); кадровый учет преподавательского состава; планирование; распределение и контроль исполнения работ; формирование аналитической и отчетной документации

Продолжение табл.

Варианты заданий		*Дополнительные требования*
2.1	Электронный дневник школьника	тематические планы изучения предметов по параллелям с учетом специализаций в старших классах; регистрация результатов текущей успеваемости учащихся и выполнения контрольных работ; формирование рейтинговых списков по предметам
2.2	Распределение годовой учебной нагрузки преподавателей вуза	учебные планы по специальностям, образовательным уровням и формам обучения; дисциплины по семестрам и кафедрам; объем в часах по видам занятий; итоговый контроль; контингент студентов; штатное расписание преподавателей кафедр
2.3	Мониторинг успеваемости студентов	учебные планы по специальностям, образовательным уровням и формам обучения; дисциплины по кафедрам и семестрам; итоговый контроль; контингент студентов по специальностям, образовательным уровням и формам обучения
2.4	Центр профессиональной переподготовки специалистов	управление образовательными программами; бухгалтерский учет (прием платежей от слушателей, оплата работы преподавателей); регистрация выдачи сертификатов
2.5	Администрирование компьютерных классов	оборудование и ПО рабочих мест; закрепление учебных дисциплин; расписание занятий в компьютерных классах; загруженность классов в течение семестра
Тема № 3. АВТОМАТИЗИРОВАННЫЕ БИБЛИОТЕЧНЫЕ СИСТЕМЫ		
Пользователи		*Автоматизируемые бизнес-процессы*
библиотекари; читатели		учет движения библиотечного фонда (поступление, списание); поиск объектов библиотечного фонда; регистрация читателей; регистрация выдачи/возврата; анализ читательского спроса
Варианты заданий		*Дополнительные требования*
3.1	Абонемент публичной библиотеки	система поиска книг, ориентированная на «неподготовленного» читателя: по авторам (с учетом соавторства); по названию и году издания книг; по редактируемому классификатору с возможностью «привязки» одной книги к нескольким жанрам
3.2	Абонемент университетской библиотеки	учебные планы по специальностям и образовательным уровням; контингент читателей (студентов); категории учебно-методических изданий; поиск по авторам, названиям, категориям, специальностям и учебным дисциплинам с учетом рекомендаций

Продолжение табл.

3.3	Читальный зал периодических изданий	редактируемый линейный классификатор изданий (например: научно-технические, литературно-художественные, детские, информационно-рекламные и др.); поиск статей в выпусках изданий по классификатору, авторам и названиям статей; предварительный просмотр аннотаций статей
3.4	Читальный зал научно-технических изданий	редактируемый иерархический многоуровневый классификатор категорий изданий (например, книги (научные монографии, учебники и др.), журналы (патентные, научные, популярные и др.)); редактируемый иерархический классификатор отраслей науки и техники; поиск по категориям и отраслям науки и техники
Тема № 4. СПОРТИВНЫЕ СОРЕВНОВАНИЯ		
Пользователи		*Автоматизируемые бизнес-процессы*
участники; болельщики; руководители; администраторы; аналитики		регистрация участников; планирование; оперативный учет результатов; анализ
Варианты заданий		*Дополнительные требования*
4.1	Командные спортивные соревнования (вид спорта — по выбору разработчика)	одна спортивная лига по одному виду спорта; хранение истории нескольких спортивных сезонов; регистрация и учет состояния спортивных арен; регистрация команд — участников спортивных соревнований в каждом сезоне; регистрация спортсменов — участников команд; рейтинги спортсменов с учетом их личных достижений в спортивных матчах
4.2	Командные спортивные соревнования (вид спорта — по выбору разработчика)	несколько спортивных лиг; один спортивный сезон; регистрация и учет состояния спортивных арен; регистрация команд — участников спортивных соревнований; регистрация спортсменов — участников команд; рейтинги
4.3	Индивидуальные спортивные соревнования (вид спорта — по выбору разработчика)	одна спортивная лига; хранение истории нескольких спортивных сезонов; регистрация спортсменов — участников соревнований в каждом сезоне; рейтинги спортсменов с учетом их личных достижений
4.4	Индивидуальные спортивные соревнования (вид спорта — по выбору разработчика)	несколько спортивных лиг по одному виду спорта; один спортивный сезон; регистрация спортсменов — участников спортивных соревнований; рейтинги спортсменов с учетом их личных достижений

Продолжение табл.

Тема № 5. ЗДОРОВЬЕ, ОТДЫХ, ТУРИЗМ		
Пользователи		*Автоматизируемые бизнес-процессы*
гости; пациенты (клиенты); руководители; врачи (тренеры); аналитики		формирование прейскуранта услуг; кадровый учет; регистрация клиентов; планирование; оперативный учет; анализ
Варианты заданий		*Дополнительные требования*
5.1	Регистратура поликлиники	электронные амбулаторные карты пациентов поликлиники; расписание приема врачей-специалистов; запись пациентов на прием к врачам; диагнозы и назначения
5.2	Ветеринарная лечебница	специализация ветеринаров; классификатор животных; расписание приема ветеринаров; запись пациентов на прием; диагнозы и назначения
5.3	Спортивно-оздоровительный комплекс	прайс-лист (секции, группы, цены); оборудование и специализация спортзалов; формирование групп; контроль посещения занятий и прием платежей от клиентов; финансовая отчетность
5.4	Горнолыжный курорт	прайс-лист; продажа путевок; размещение клиентов; аренда спортивного инвентаря; дополнительные услуги
5.5	Туристическое агентство	прайс-лист; туры; маршруты; отели и транспортное обслуживание; экскурсии
Тема № 6. ТРАНСПОРТ		
Пользователи		*Автоматизируемые бизнес-процессы*
гости; клиенты; руководители; менеджеры		формирование прайс-листа; кадровый учет; регистрация клиентов; оперативный учет; анализ
Варианты заданий		*Дополнительные требования*
6.1	Автосалон	поиск автомобиля по маркам, моделям, комплектациям, ценам; управление продажами: продажа автомобилей с пробегом; подбор комплектации и продажа новых автомобилей; оформление договоров с клиентами; Trade IN (продажа нового автомобиля с одновременной покупкой автомобиля клиента)

Продолжение табл.

6.2	Прокат автомобилей	поиск автомобиля по категориям, маркам, моделям, ценам, типам двигателя и кузова; регистрация клиентов; регистрация парка автомобилей; выдача/возврат; финансовый анализ
6.3	Прокат велосипедов	размещение велопарковок (в черте города); выбор велосипеда по типам, моделям; наличие велосипедов на велопарковках; выдача/возврат; финансовый анализ
6.4	Агрегатор такси	адресный справочник; регистрация водителей и автомобилей; прием заказов; расчет стоимости поездки; учет исполнения заказов; финансовый учет и анализ
6.5	Автовокзал	автопарк; междугородние маршруты; расписание рейсов; продажа билетов
Тема № 7. ТОРГОВО-СКЛАДСКОЙ УЧЕТ		
Пользователи		*Автоматизируемые бизнес-процессы*
гости; клиенты; кладовщики; продавцы; менеджеры		классификация товаров по категориям; формирование прайс-листа; кадровый учет; регистрация клиентов; учет поставок и отпуска товаров
Варианты заданий		*Дополнительные требования*
7.1	Оптовый склад продуктов питания	учет предельных сроков реализации товаров; списание просроченных товаров; персональные скидки постоянным клиентам
7.2	Универсальный интернет-магазин	формирование и контроль исполнения заказов; доставка товаров покупателям
7.3	Магазин по продаже компьютерной и оргтехники	контроль совместимости комплектующих; комплектование товаров по заявкам покупателей
7.4	Мебельный магазин	комплектование товаров; выполнение дизайн-проектов; доставка товаров покупателям; сборка товаров у покупателей
Тема № 8. ОБЩЕСТВЕННОЕ ПИТАНИЕ		
Пользователи		*Автоматизируемые бизнес-процессы*
гости; клиенты; официанты; менеджеры		классификация блюд и напитков по категориям; формирование меню, состав блюд; кадровый учет; прием и учет исполнения заказов

Продолжение табл.

Варианты заданий		*Дополнительные требования*
8.1	Ресторан	специализация ресторана («кухня»); предварительные заказы; банкеты
8.2	Диетическое питание	контроль состава блюд; рецепты приготовления блюд; расчет калорийности блюд

ЧАСТЬ 3.
ПРОГРАММИРОВАНИЕ БАЗ ДАННЫХ

Поддержка языка управления данными — одна из важнейших функций СУБД, обеспечивающая разработчика языковыми средствами описания схемы базы данных и формирования запросов, связанных с извлечением или модификацией хранимой в ней информации. При этом программист формулирует запросы в терминах логической модели данных, а встроенный в СУБД транслятор преобразует высокоуровневый программный код в низкоуровневые процедуры, оперирующие соответствующими структурами физической модели.

Ниже будут рассмотрены основные конструкции языка SQL (Structured Query Language — структурированный язык запросов), различные диалекты которого поддерживаются реляционными СУБД. Процесс трансляции SQL-запроса обсуждается в четвертой части данного учебника.

ГЛАВА 6. ОСНОВЫ ЯЗЫКА SQL

В структуре языка SQL выделяют три группы операторов, называемых *подъязыками SQL*:

- **DDL** (Data Definition Language) — язык определения данных;
- **DCL** (Data Control Language) — язык управления доступом к данным;
- **DML** (Data Manipulation Language) — язык манипулирования данными.

6.1. DDL — язык определения данных

Язык DDL используется для описания структуры именованных логических объектов базы данных — баз данных, таблиц, представлений, индексов, процедур, функций, ограничений целостности, пользователей и т. д. Язык DDL включает три оператора (часто называемые *командами*, в отличие от *операторов* языка DML): CREATE, ALTER и DROP, используемых соответственно для создания, модификации и удаления объектов.

Использование DDL-команд (в простейших вариантах их синтаксических конструкций) иллюстрируется листингом 3.1.

```
а) CREATE MyDatabase;
б) CREATE TABLE (Key IDENTITY PRIMARY KEY, Col INT, Data CHAR(60));
в) CREATE NONCLUSTERED INDEX Ind ON MyTable(Col);
г) CREATE VIEW MyView(Data) AS SELECT Data FROM MyTable WHERE Col > 6;
д) CREATE LOGIN SimpleClerk WITH PASSWORD = 'simplePassword';
е) CREATE USER Clerk FOR LOGIN SimpleClerk;
ж) ALTER USER Clerk WITH NAME = BigClerk;
и) DROP VIEW MyView;
к) CREATE ROLE All_Clerks;
л) ALTER ROLE All_Clerks ADD MEMBER BigClerk;
```

Листинг 3.1
Примеры использования DDL-команд

Комментарии к примерам:

а) создается база данных MyDatabase, все параметры файловой структуры которой определены по умолчанию;

б) создается таблица MyTable со схемой из трех столбцов: первичный ключ Key целочисленного автоинкрементного типа, столбец Col целочисленного типа и столбец Data строкового типа;

в) в таблице MyTable создается индекс Ind по столбцу Col;

г) создается унарное *представление* MyView, строки которого содержат значение столбца Data тех строк таблицы MyTable, для которых значение Col > 6;

д) создается *учетная запись* SimpleClerk с паролем SimplePassword;

е) создается *пользователь* Clerk, связанный с учетной записью SimpleClerk;

ж) пользователь Clerk получает новое имя BigClerk;

и) удаляется представление MyView;

к) создается *пользовательская роль* (группа пользователей) All_Clerks;

л) пользователь BigClerk становится членом роли All_Clerks.

6.2. DCL — язык управления доступом

Язык DCL используется для управления разрешениями (permissions) доступа к объектам базы данных, таких как таблицы, представления, хранимые процедуры и функции, а также разрешениями на выполнение DCL-команд со стороны субъектов доступа — пользователей, ролей, хранимых процедур и функций.

DCL включает три команды: GRANT (предоставить доступ), DENY (запретить доступ) и REVOKE (отменить ранее выданное разрешение), управляющие разрешениями следующих типов:

– SELECT — разрешение на чтение строк таблицы или представления;

– INSERT — разрешение на вставку строк в таблицу;

– DELETE — разрешение на удаление строк таблицы;

– UPDATE — разрешение на изменение (данных в строках таблицы или программного SQL-кода хранимых процедур и функций);

– REFERENCES — разрешение субъекту доступа создавать внешние ключи в подчиненных таблицах без права доступа к главным таблицам;

– EXECUTE — разрешение на выполнение хранимых процедур и функций.

Правила применения и синтаксические конструкции DCL-команд детально рассмотрены в пятой части данного учебника, листинг 3.2 содержит простейшие примеры их использования.

```
а) GRANT SELECT ON MyTable(Data) TO All_Clerks;
б) DENY INSERT, DELETE, UPDATE ON MyTable TO All_Clerks;
в) GRANT SELECT ON MyTable TO BigClerk WITH GRANT OPTION;
г) DENY ALL ON MyTable TO BigClerk CASCADE;
д) REVOKE ALL ON MyTable TO BigClerk;
```

Листинг 3.2
Примеры использования DCL-команд

Комментарии к примерам:

а) предварительно созданной пользовательской роли All_Clerks предоставляется право чтения столбца Data в таблице MyTable;

б) пользовательской роли All_Clerks запрещается вставка, удаление и изменение данных во всех столбцах всех строк таблицы MyTable;

в) пользователю BigClerk предоставляется право чтения строк таблицы MyTable с правом предоставления этого права другим субъектам доступа (WITH GRANT OPTION);

г) пользователю BigClerk запрещаются все операции над таблицей MyTable, при этом запрет каскадно (CASCADE) распространяется на всех субъектов доступа, получивших данные права от пользователя BigClerk;

д) отменяются ранее выданные пользователю BigClerk разрешения (как запрещающие, так и предоставляющие права доступа).

6.3. DML — язык манипулирования данными

В отличие от процедурных языков программирования, язык SQL (точнее его DML-подмножество) в своей основе является языком *декларативного* типа, и текст SQL-запроса к базе данных не содержит описания алгоритма получения результата, а только *декларирует требования* к этому *результату*. Например, SQL-запрос вида **Select * From T Where T.x>T.y** требует произвести выборку из таблицы **T** только тех ее строк, в которых значение поля **x** больше значения поля **y**, не описывая при этом алгоритма выполнения такой операции. Современные версии языка SQL содержат ряд процедурных расширений, некоторые из них обсуждаются в п. 4.2.

Язык DML содержит 4 составных оператора, наименования которых определяют основное их предназначение:

– оператор **SELECT** — обеспечивает *выборку* множества кортежей отношений (строк таблиц), удовлетворяющих заданным ограничениям, и (возможно) обработку выбранных данных;

– оператор **INSERT** — обеспечивает *вставку* (добавление) в таблицы новых строк с заданными значениями их атрибутов;

– оператор **UPDATE** — обновляет значения атрибутов в строках таблиц, удовлетворяющих заданным ограничениям;

– оператор **DELETE** — удаляет из таблиц строки, удовлетворяющие заданным ограничениям.

Среди перечисленных DML-операторов оператор **SELECT**, обеспечивающий выборку строк таблиц, является основным в том смысле, что семантически (не синтаксически!) он оказывается «вложенным» в каждый из остальных операторов: выборка строк производится перед их удалением (**DELETE**), обновлением значений (**UPDATE**) или генерацией перед их вставкой (**INSERT**) в таблицу.

Составной оператор выборки **SELECT** включает 6 именованных разделов: **SELECT**, **FROM**, **WHERE**, **ORDER BY**, **GROUP DY** и **HAVING**, из которых только первые два раздела являются обязательными.

Результатом выполнения оператора **SELECT** является виртуальное отношение, схема которого определяется списком атрибутов, заданным в разделе **SELECT** этого оператора, а состав кортежей — параметрами остальных его разделов.

Синтаксис и семантика оператора **SELECT** иллюстрируются примерами SQL-запросов (листинги 3.3–3.6), использующими учебную базу данных, схема которой представлена на рисунке 3.2.

6.3.1. Простейшие SQL-запросы

Листинг 3.3 иллюстрирует синтаксис SQL-запросов, в результате выполнения которых из единственной таблицы БД производится выборка:

а) всех строк и всех столбцов таблицы;

б) всех строк и четырех столбцов таблицы;

в) всех строк таблицы с добавлением вычисляемого поля Стоимость;

г) товаров, количество которых превышает минимально допустимый запас;

д) заказов, размещенных позднее 1 января 2015 г.;

е) заказов, размещенных в диапазоне дат от 1 января до 31 декабря 2015 г.;

ж) заказов, размещенных в декабре 2015 г. (используются встроенные функции обработки темпоральных типов данных);

и) имен и номеров телефонов представителей поставщиков и клиентов, имена которых включают заданный набор текстовых символов;

к) имен и номеров телефонов представителей поставщиков, которые при этом не являются представителями клиентов.

```
а) SELECT * FROM Склад;
б) SELECT Товар, ОптоваяЦена, Количество, МинимальныйЗапас FROM Склад;
в) SELECT Товар, ОптоваяЦена*Количество AS Стоимость FROM Склад;
г) SELECT Товар, ОптоваяЦена*Количество AS Стоимость
   FROM Склад WHERE Количество> МинимальныйЗапас;
д) SELECT Код_Заказа,ДатаРазмещения FROM Заказы
   WHERE Заказы.ДатаРазмещения>#01-01-2015#;
е) SELECT Код_Заказа, ДатаРазмещения FROM Заказы
   WHERE ДатаРазмещения BETWEEN #01-01-2015# AND #31-12-2015#;
ж) SELECT Код_Заказа, ДатаРазмещения FROM Заказы
   WHERE YEAR(ДатаРазмещения)=2015 AND MONTH(ДатаРазмещения)=12;
и) SELECT DISTINCT Представитель, Телефон FROM Представители
   WHERE Представитель Like «*Peter*»;
к) SELECT DISTINCT Представитель, Телефон FROM Представители
   WHERE Код_Клиента IS NULL AND Код_Поставщика IS NOT NULL.
```

Листинг 3.3

Примеры простейших «однотабличных» SQL-запросов

Следующие комментарии к рассмотренным выше примерам поясняют отдельные языковые конструкции и правила написания SQL-запросов.

Раздел SELECT представляет реляционно-алгебраическую *операцию проекции* отношения, указанного в разделе FROM, на список атрибутов, указанных в разделе SELECT. В этом разделе допускается использование *вычисляемых полей* — атрибутов, отсутствующих в базовых таблицах и вычисляемых в процессе выполнения запроса. Имена вычисляемых атрибутов указываются после ключевого слова AS, при наличии пробелов в имени оно должно заключаться в прямые скобки.

Следует заметить, что SQL-модель данных отличается от классической реляционной модели тем, что допускает наличие повторяющихся кортежей в результирующих отношениях. Если, например, в списке атрибутов раздела

SELECT отсутствуют возможные ключи, не исключена вероятность появления в результирующем отношении кортежей-дубликатов, и это не будет нарушением требований SQL-модели данных. Для исключения дублирующих кортежей в разделе SELECT следует указать параметр DISTINCT, как это сделано в двух последних примерах.

Раздел WHERE реализует реляционно-алгебраическую *операцию ограничения*. Параметром этого раздела является так называемое *условие ограничения* — любое корректное логическое выражение, которое будет вычисляться для каждого кортежа отношения, указанного в разделе FROM: в результирующем отношении останутся только те кортежи, для которых это выражение примет значение «истина». В качестве операндов логических выражений могут использоваться константы и имена любых атрибутов отношений, указанных в разделе FROM, а также составленные из имен атрибутов логические выражения, использующие операторы AND, OR и NOT.

В логических выражениях могут использоваться стандартные предикаты сравнения (**=, <, <=, >, >=, <>**), предикаты **between**, **IS NULL**, **LIKE** (для сравнения строковых данных), а также предикаты **IN**, **ALL** и **EXIST**, которые будут рассмотрены позднее при обсуждении примеров использования подчиненных (вложенных) запросов.

При формировании логических выражений условий ограничения следует учитывать еще одну специфическую особенность SQL-модели, допускающей неопределенные (NULL) значения атрибутов в кортежах отношений. В этих условиях вычисление условия ограничения производится не в булевой, а в трехзначной (тернарной) логике со значениями **true, false** и **unknown** в соответствии с таблицей истинности (табл. 3.1).

Таблица 3.1

Таблица истинности в трехзначной логике

true OR **unknown** = **true**
true AND **unknown** = **unknown**
unknown OR **unknown** = **unknown**
unknown AND **unknown** = **unknown**
false OR **unknown** = **unknown**
false AND **unknown** = **false**
NOT **unknown** = **unknown**

Для сравнения данных дата-временных типов допускается использовать стандартный набор скалярных предикатов сравнения (**=, <, <=, >, >=, <>**) и предиката between, так как внутренним представлением данных этого типа является число: для даты — количество дней, прошедших с некоторой начальной даты до указанной даты, для времени — количество временных единиц (например, сотых долей секунды), прошедших с начала суток до заданного времени.

По этой же причине допускается применять весь набор арифметических операций к данным дата-временных типов: например, можно вычислить новую

дату путем сложения даты с числом или вычислить длину временного интервала путем вычитания двух дат.

Дата-временные константы помещаются внутри пары символов **#,** строковые константы заключаются в кавычки (одинарные или двойные).

Для обработки строковых данных может быть использован стандартный набор встроенных функций, обеспечивающих слияние и расщепление строк, выделение подстрок в строках, вычисление длины строки и т. д.

6.3.2. SQL-запросы с соединением (JOIN) таблиц

Листинг 3.4 представляет более сложные SQL-запросы, в которых выборка производится из виртуальных таблиц, получаемых в результате соединения нескольких реальных таблиц базы данных путем применения к ним реляционно-алгебраической операции JOIN в ее различных модификациях.

```
а) SELECT Категория,Товар,Поставщик, ОптоваяЦена,
        Количество,ЕдиницаИзмерения,
        ОптоваяЦена*Количество AS [Стоимость складского запаса]
    FROM (Поставщики INNER JOIN (Категории INNER JOIN Склад
        ON Категории.Код_Категории = Склад.Код_Категории)
        ON Поставщики.Код_Поставщика = Склад.Код_Поставщика)
        INNER JOIN ЕдиницыИзмерения
        ON ЕдиницыИзмерения.Код_Ед = Склад.Код_ЕдИзм
    WHERE Количество>0
    ORDER BY Категория ASC, ОптоваяЦена*Количество DESC;
б)  SELECT Города.Город, Страны.Страна, Регионы.Регион
    FROM Регионы, Страны, Города;
в)  SELECT Города.Город, Страны.Страна, Регионы.Регион
    FROM Регионы, Страны, Города
    WHERE Города.Код_Страны=Страны.Код_Страны
            AND Страны.Код_Региона=Регионы.Код_Региона;
г)  SELECT Города.Город, Страны.Страна
    FROM Регионы INNER JOIN (Страны INNER JOIN Города
            ON Страны.Код_Страны = Города.Код_Страны)
            ON Регионы.Код_Региона = Страны.Код_Региона;
д) SELECT Города.Город, Страны.Страна FROM Страны LEFT JOIN Города
            ON Страны.Код_Страны = Города.Код_Страны;
е) SELECT Города.Город, Страны.Страна FROM Страны RIGHT JOIN Города
            ON Страны.КодСтраны = Города.КодСтраны;
```

Листинг 3.4
Примеры SQL-запросов с соединением таблиц

Комментарии к примерам, приведенным в листинге 3.4:

а) раздел FROM реализует реляционно-алгебраическую операцию внутреннего соединения (INNER JOIN) четырех взаимосвязанных таблиц, в качестве

условий соединения которых используются равенства значений первичных ключей главных таблиц и внешних ключей соответствующих подчиненных таблиц:

– операция внутреннего соединения производит конкатенацию (сцепление) только тех кортежей таблиц, в которых обнаруживается такое равенство;

– результат выборки представляется в отсортированном виде: кортежи упорядочены по двум критериям — по категории товара (в алфавитном порядке по возрастанию) и по суммарной стоимости складского запаса товара (в порядке убывания);

б) в разделе FROM явно не указан тип соединения таблиц, их имена просто разделены запятой — синтаксически это обозначает выполнение реляционно-алгебраической *операции расширенного декартова произведения*, производящей виртуальную таблицу, в которой каждая строка таблицы Страны сцеплена со всеми строками таблицы Города:

– пример иллюстрирует лишь синтаксические возможности языка, результат операции не имеет никакого смысла и явно противоречит не только семантике предметной области, но и естественным географическим представлениям о расположении регионов, стран и городов;

– мощность результирующей таблицы будет равной произведению мощностей всех трех базовых таблиц и составит 30 000 строк для примера из учебной базы данных, в таблицах которой представлены 10 регионов, 30 стран и 100 городов;

в) в этом примере сделана попытка устранения семантического недостатка предыдущего SQL-запроса:

– вначале состав кортежей таблицы получен декартовым перемножением трех базовых таблиц;

– затем состав кортежей ограничивается условием равенства первичных ключей главных таблиц и соответствующих внешних ключей подчиненных таблиц;

– формально эта попытка вполне успешна, так как результат запроса дает правильный перечень из 100 городов с указанием регионов и стран, в которых действительно расположены эти города;

– однако вряд ли следует считать правильным метод получения этого результата — вначале производится таблица мощностью 30 000 строк, а затем из нее удаляются лишние 29 900 строк;

г) результат выполнения этого запроса настолько же правилен, как и предыдущего, однако здесь предлагается совсем другой способ его достижения: в разделе FROM явно указана операция внутреннего соединения трех таблиц, что предписывает транслятору подобрать эффективный процедурный план выполнения этой операции, использующий, например, индексные структуры данных вместо полного перебора кортежей таблиц по методу вложенных циклов;

д) в отличие от внутреннего соединения (INNER JOIN), левое соединение (LEFT JOIN) производит таблицу, содержащую *все строки* левой базовой таблицы, в том числе и те, для которых в правой таблице отсутствуют соответству-

ющие строки. При этом недостающие поля результирующей таблицы получат неопределенные NULL-значения. В рассматриваемом примере в результирующей таблице будут представлены все страны, в том числе и те, в которых нет городов (разумеется, не в географическом смысле);

е) операция RIGHT JOIN произведет таблицу, в которой будут представлены все города, в том числе и те, которые оказались не связанными ни с одной из стран.

6.3.3. SQL-запросы с объединением (UNION) таблиц

В отличие от оператора JOIN, сцепляющего строки таблиц и производящего таблицу «суммарной» арности, оператор UNION выполняет операцию *объединения* таблиц, в результате которой формируется таблица «суммарной» мощности.

Естественным ограничением на выполнение этой операции является требование *совместимости* объединяемых таблиц, в основе которого лежат базовые ограничения реляционной модели данных, требующие, в частности, идентичности схем всех кортежей отношения и наличия среди атрибутов хотя бы одного уникального ключа.

Как уже отмечалось, SQL-модель данных отличается от классической реляционной модели и поддерживает только минимальное требование совместимости: *все объединяемые оператором UNION таблицы должны иметь одинаковую арность*. Остальные ограничения в конкретных реализациях языка либо игнорируются (как, например, несовпадение типов данных в соответствующих столбцах объединяемых таблиц), либо принимаются решения «по умолчанию» (например, имена столбцов объединенной таблицы наследуются от имен столбцов первой из объединяемых).

Листинг 3.5 иллюстрирует использования оператора UNION для формирования объединенной таблицы, включающей перечень всех контрагентов — как клиентов, так и поставщиков товаров. Дополнительный (вычисляемый) столбец таблицы содержит наименование статуса контрагента.

```
SELECT Код_Клиента AS Код, Клиент AS Контрагент,
Город, АдресГлавногоОфиса AS Адрес,
Телефон, «Клиент» AS [Статус]
FROM Города INNER JOIN Клиенты ON
Города.Код_Города = Клиенты.КодГорода
UNION
SELECT Код_Поставщика, Поставщик, Город,
Адрес,Телефон, «Поставщик»
FROM Города INNER JOIN Поставщики ON
Города.Код_Города = Поставщики.КодГорода;
```

Листинг 3.5
Пример использования оператора UNION

6.3.4. Модифицирующие SQL-запросы

Оператор INSERT вставляет строки в существующую таблицу, при этом структура и значения столбцов вставляемых строк должны соответствовать схеме таблицы. Оператор DELETE удаляет из таблицы строки, соответствующие условию ограничения WHERE. Оператор UPDATE изменяет значения указанных столбцов таблицы в строках, соответствующих ограничению WHERE.

```
а) INSERT INTO Поставщики (Поставщик, Адрес, Телефон, КодГорода)
VALUES («Horns and Hoofs, Limited», «666666», «+7 777777», 106);
б) INSERT INTO Поставщики (Поставщик, Адрес, Телефон, КодГорода)
SELECT Клиент,АдресГлавногоОфиса, Телефон, КодГорода
FROM Клиенты WHERE Клиенты.КодГорода=20;
в) SELECT Заказы.Код_Заказа, ДатаИсполнения,
Товар, Заказано.Количество,Клиент, Сотрудник
   INTO Продажи_2018
   FROM Сотрудники INNER JOIN (Клиенты
INNER JOIN (Склад INNER JOIN (Заказы INNER JOIN Заказано
ON Заказы.Код_Заказа = Заказано.Код_Заказа)
ON Склад.КодТовара = Заказано.Код_Товара)
ON Клиенты.Код_Клиента = Заказы.Код_Клиента)
ON Сотрудники.Код_Сотрудника = Заказы.Код_Сотрудника
  WHERE Year(ДатаИсполнения)=2018;
г) UPDATE Склад INNER JOIN (Заказы INNER JOIN Заказано
ON Заказы.Код_Заказа = Заказано.Код_Заказа)
ON Склад.КодТовара = Заказано.Код_Товара
SET Склад.Количество =
    Склад.Количество — Заказано.Количество
WHERE Заказы.ДатаИсполнения =Date();
д) DELETE * FROM Поставщики WHERE КодГорода=106;
```

Листинг 3.6
Примеры использования операторов
INSERT, DELETE и UPDATE

Синтаксические правила использования этих операторов интуитивно понятны и иллюстрируются приведенными ниже примерами (листинг 3.6):

а) в таблицу *Поставщики* вставляется одна строка, значения полей которой заданы соответствующими константами;

б) все клиенты, находящиеся в городе с кодом 20, становятся поставщиками (по-прежнему оставаясь клиентами);

в) этот пример иллюстрирует еще один способ вставки строк в таблицу без использования оператора INSERT: инструкция SELECT … INTO создает в базе данных *новую таблицу Продажи_2018*, в которую помещается результат выборки информации о товарах, включенных в заказы 2018 г.;

г) во всех строках таблицы *Склад*, связанных со строками таблицы *Заказано*, которые, в свою очередь, связаны со строками таблицы *Заказы*, датированными «сегодняшним» числом, изменяется значение поля *Количество* путем его уменьшения на количество проданных товаров (по завершении торгового дня корректируется складской запас товаров с учетом объемов проданных товаров);

д) из таблицы *Поставщики* удаляются все строки, представляющие поставщиков из города № 106:

– этот пример иллюстрирует простейшую ситуацию, когда условие выборки удаляемых из таблицы строк ссылается только на поля этой таблицы;

– в более сложных случаях (например, для удаления поставщиков, поставки товаров которых прекращены) потребуются ссылки на другие таблицы, связанные с модифицируемой таблицей;

– SQL дает несколько альтернативных способов решения такой задачи, один из них связан с использованием подчиненных запросов, рассматриваемых в п. 6.3.6.

6.3.5. Хранимые представления

Представлением (***view***) называется именованный логический объект, представляющий собой SQL-запрос, хранимый в базе данных в виде исходного SQL-кода или в некотором прекомпилированном формате. При выполнении представления формируется виртуальная таблица, схема и состав кортежей которой определены структурой оператора CREATE VIEW, а также текущим состоянием используемых в представлении базовых таблиц, в котором они находились в момент выполнения представления.

Ссылки на имена представлений и имена их полей могут использоваться в операторе SELECT точно так же, как для реальных таблиц базы данных, что делает хранимые представления полезным и эффективным инструментом при разработке пользовательских запросов.

Листинг 3.7 содержит пример использования хранимых представлений.

Операторами а) и б) создаются два хранимых представления: *Представители_Клиентов* (соединением пяти базовых таблиц: Клиент, Представитель, Город, Страна и Регион) и, аналогично, представление *Представители_Клиентов*.

Результатом автономного выполнения каждого из этих представлений будет список представителей (соответственно клиентов или поставщиков), для каждого из которых будет указано имя представителя, город, страна и регион его нахождения, а также наименование его работодателя (клиента или поставщика).

Далее оператором в) производится выборка строк из виртуальной таблицы, полученной путем соединения двух представлений (также виртуальных таблиц), в результате формируется список городов, в которых находятся и представители клиентов, и представители поставщиков, с указанием имен таких представителей.

```
а) CREATE VIEW Представители_Клиентов
    (Клиент, Представитель, Город, Страна, Регион)
    AS
     SELECT Представитель, Клиент, Город, Страна, Регион
     FROM (Регионы INNER JOIN (Страны INNER JOIN  Города
        ON Страны.Код_Страны = Города.КодСтраны)
        ON Регионы.Код_Региона = Страны.КодРегиона)
        INNER JOIN (Клиенты INNER JOIN Представители
        ON Клиенты.Код_Клиента = Представители.Код_Клиента)
        ON Города.Код_Города = Представители.Код_Города;
б) CREATE VIEW Представители_Поставщиков
    (Поставщик, Представитель, Город, Страна, Регион)
    AS
    SELECT Представитель, Поставщик, Город, Страна, Регион
     FROM (Регионы INNER JOIN (Страны INNER JOIN  Города
        ON Страны.Код_Страны = Города.КодСтраны)
        ON Регионы.Код_Региона = Страны.КодРегиона)
        INNER JOIN (Поставщики INNER JOIN Представители
        ON Поставщики.Код_Поставщика = Представители.Код_Поставщика)
        ON Города.Код_Города = Представители.Код_Города;
в) SELECT Представители_Клиентов.Представитель,
         Представители_Поставщиков.Представитель,
         Представители_Клиентов.Город
    FROM Представители_Клиентов INNER JOIN
         Представители_Поставщиков
         ON Представители_Клиентов.Город =
              Представители_Поставщиков.Город
   ORDER BY Представители_Клиентов.Город;
```

Листинг 3.7
Пример использования представлений в SQL-запросах

6.3.6. Подчиненные SQL-запросы

Подчиненный запрос или, более кратко, *подзапрос* — это SQL-запрос, вложенный в другой SQL-запрос и компилируемый совместно с основным запросом. Синтаксически подзапросом будет считаться любой корректный оператор SELECT, заключенный в круглые скобки. Подзапросы могут находиться в любых разделах основного запроса, допускающих использование выражений, в том числе они могут быть вложены в предикаты условий выборки раздела WHERE.

Если подзапрос возвращает скалярное значение (унарную таблицу мощностью в одну строку), его допускается использовать в качестве операнда в простых предикатах сравнения (**=, <, <=, >, >=, <>**), предикатах **between** или **LIKE** в зависимости от типа данных возвращаемого подзапросом значения. Такое использование подзапроса иллюстрирует листинг 3.8 — результирующий SQL-запрос

возвращает список имен торговых представителей, находящихся в том же городе, что и сотрудник по имени «Новиков».

```
SELECT Представитель, Город
FROM Города INNER JOIN Представители
ON Города.Код_Города = Представители.Код_Города
WHERE Город = (
SELECT Город
FROM (Города INNER JOIN Филиалы
ON Города.Код_Города = Филиалы.Код_Города)
INNER JOIN Сотрудники
ON Филиалы.Код_Филиала = Сотрудники.Код_Филиала
WHERE Сотрудник Like «Новиков»);
```

Листинг 3.8
Пример использования подчиненного запроса
в простом предикате сравнения раздела WHERE

Если подзапрос возвращает множество скалярных значений (унарную таблицу из множества строк), в предикатах условий выборки раздела WHERE основного запроса могут использоваться ключевые слова **ALL**, **ANY**|**SOME**, **IN**, **EXISTS** и **NOT EXISTS**.

Предикат **EXISTS** (*подзапрос*) получит значение «истина», если *подзапрос* возвращает непустое множество строк.

Предикат *выражение* **IN** (*подзапрос*) получит значение «истина», если вычисляемое значение *выражения* входит во множество значений, возвращаемое *подзапросом*.

```
а) SELECT Сотрудник FROM Сотрудники
    WHERE NOT EXISTS
(SELECT Код_Заказа FROM Заказы
WHERE Заказы.Код_Сотрудника =
    Сотрудники.Код_Сотрудника
AND YEAR(Заказы.ДатаРазмещения)=2017);
б)  SELECT Сотрудник FROM Сотрудники
    WHERE Код_Сотрудника NOT IN
(SELECT Код_Сотрудника FROM Заказы
WHERE YEAR(Заказы.ДатаРазмещения)=2017);
в)  DELETE  FROM Поставщики
WHERE Код_Поставщика IN (
SELECT Код_Поставщика FROM Склад
WHERE ПоставкиПрекращены=True);
```

Листинг 3.9
Примеры использования подчиненных запросов
в предикатах сравнения EXISTS и IN

Примеры а) и б) в листинге 3.9 представляют два синтаксически различных, но семантически эквивалентных SQL-запроса, производящих выборку из таблицы *Сотрудники* имен всех сотрудников, не оформивших ни одного заказа в 2017 г.

Первый из двух этих примеров иллюстрирует еще одну важную особенность использования подчиненных запросов, связанную с известным в программировании понятием «области видимости переменных». Автономное выполнение подзапроса из примера а) было бы невозможным, так как в этом случае считался бы неопределенным параметр *Сотрудники.Код_Сотрудника*, однако в приведенном примере такая конструкция вполне корректна, так как *в подзапросе доступны имена всех столбцов всех таблиц, используемые в разделах FROM всех внешних запросов более высоких уровней*.

Пример в) демонстрирует запрос на удаление поставщиков, прекративших поставку хотя бы одного из своих товаров.

```
а)  SELECT Поставщик FROM Поставщики
      WHERE Код_Поставщика=ANY(
            SELECT Код_Поставщика FROM Склад
   WHERE Количество>0 AND
               ПоставкиПрекращены=True);
б)   SELECT Поставщик  FROM Поставщики
     WHERE Код_Поставщика=ALL(
     SELECT Код_Поставщика  FROM Склад
     WHERE Количество<=МинимальныйЗапас);
в)   SELECT Сотрудник FROM Сотрудники
      WHERE Код_Сотрудника=SOME(
      SELECT  DISTINCT Код_Сотрудника FROM Заказы
      WHERE ДатаИсполнения >
      DateAdd(«m»,1,ДатаРазмещения));
г)   DELETE   FROM Поставщики
     WHERE Код_Поставщика = ALL (
          SELECT Код_Поставщика FROM Склад
              WHERE ПоставкиПрекращены=True);
```

Листинг 3.10
Примеры использования подчиненных запросов
в предикатах сравнения ANY|SOME и ALL

Предикат *выражение* **ANY|SOME** (*подзапрос*) получит значение «истина», если вычисляемое значение *выражения* совпадает *хотя бы с одним значением* из множества значений, возвращаемых *подзапросом*, а предикат *выражение* **ALL** (*подзапрос*) — если вычисляемое значение *выражения* совпадает *со всеми* возвращаемыми *подзапросом значениями*.

Листинг 3.10 содержит примеры использования этих предикатов.

Запрос а) формирует список поставщиков, прекративших поставку *хотя бы одного из своих товаров*, имеющихся на складе.

Запрос б) формирует список поставщиков, складские запасы *всех товаров которых* не превышают установленного минимального запаса.

Запрос в) формирует список сотрудников, *хотя бы один заказ которых* исполнялся более одного месяца.

Запрос г) удаляет поставщиков, прекративших поставку всех своих товаров.

6.3.7. SQL-средства статистической обработки данных

Рассмотренные выше примеры использования DML-операторов демонстрируют возможности построчной обработки табличных данных: соединение строк таблиц, выборка отдельных строк по заданным критериям и их последующая обработка.

Язык SQL также содержит средства статистической обработки данных, обеспечивающие возможности группировки строк таблиц по различным критериям, выборки групп строк (а не отдельных строк, как это делает раздел WHERE оператора SELECT) и вычисления статистических характеристик сформированных групп строк по различным столбцам и их комбинациям.

Для формирования групп строк используются разделы GROUP BY и HAVING оператора SELECT, а для вычисления статистических характеристик сформированных групп — *агрегатные функции* (*set-functions*).

GROUP BY

Раздел GROUP BY <*список параметров группировки*> производит группировку строк таблицы, сформированной разделами SELECT, FROM и WHERE оператора SELECT: в одну группу попадают те строки сформированной таблицы, для которых все значения *параметров* из заданного *списка* одинаковы.

В качестве *параметров группировки* допускается использовать любые имена столбцов таблиц или представлений, присутствующих в разделе FROM, а также любые (нестатистические) выражения, составленные из констант и имен этих столбцов.

При наличии в операторе SELECT раздела GROUP BY раздел SELECT этого оператора не может включать имен столбцов или выражений, отсутствующих в *списке параметров группировки* (за исключением агрегатных функций, формирующих значения вычисляемых столбцов результирующей таблицы).

Наличие раздела GROUP BY в операторе SELECT имеет смысл лишь в том случае, если группировка строк производится с целью последующей «фильтрации» групп разделом HAVING и/или вычисления статистических характеристик сформированных групп с помощью агрегатных функций. Мощность (количество строк) результирующей таблицы, формируемой оператором SELECT с разделом GROUP BY, равна количеству сформированных групп.

HAVING

Раздел HAVING <*условие выборки групп*> имеет смысл только при наличии в операторе SELECT раздела GROUP BY: он вычисляет *условие выборки* для каждой группы и сохраняет в результирующей таблице только те группы, для которых это условие принимает значение «*истина*».

Условие выборки групп — это логическое выражение, операндами которого могут быть *параметры группировки* из списка параметров раздела GROUP BY и/или агрегатные функции, вычисляющие значения статистических характеристик групп, сформированных в разделе GROUP BY.

Агрегатные функции

Стандартом SQL-89 определен набор из пяти агрегатных функций, каждая из которых вычисляет значение определенной числовой характеристики для каждой группы строк, сформированных разделом GROUP BY:

– COUNT(*|*p*) — вычисляет количество строк в группе;

– MAX(*p*) — вычисляет максимальное значение параметра *p*;

– MIN(*p*) — вычисляет минимальное значение параметра *p*;

– SUM(*p*) — вычисляет суммарное значение параметра *p*;

– AVG(*p*) — вычисляет среднее арифметическое значение параметра *p*.

В качестве параметра ***p*** допускается использовать любое корректное выражение числового типа, составленное из констант и имен столбцов таблиц, указанных в разделе FROM. Исключение составляет функция COUNT, допускающая использование параметра любого типа, — при подсчете количества строк функция не будет учитывать строки, в которых этот параметр имеет неопределенное значение (NULL). Функция COUNT допускает также использование символа «*****» в качестве своего параметра — в этом случае она будет учитывать все строки групп.

При отсутствии раздела GROUP BY в операторе SELECT все строки таблицы, сформированной этим оператором, будут считаться ее единственной группой и агрегатная функция (при ее наличии) вычислит соответствующую характеристику для всей таблицы.

Агрегатные функции могут использоваться в разделах SELECT и/или HAVING оператора SELECT, а также в подчиненных запросах, как это показано в примерах листинга 3.11.

Два семантически эквивалентных, но синтаксически различных запроса а) и б) рассчитывают количество заказов, оформленных каждым из сотрудников в 2017 г. В каждом из этих запросов сначала формируется исходная таблица путем соединения таблиц Сотрудники и Заказы.

В запросе а) из исходной таблицы производится *выборка строк* (WHERE), соответствующих заказам 2017 г., затем строки полученной выборки группируются по полю Сотрудник и для каждой из полученных групп функция COUNT(*) вычисляет количество строк.

В запросе б) выборка строк не производится, а исходная таблица сразу группируется по двум параметрам — имени сотрудника и году исполнения заказа, после чего производится *выборка групп* (HAVING), соответствующих заказам 2017 г., и вычисляется количество строк для каждой из этих групп.

```
а) SELECT Сотрудник, COUNT(*) AS [Количество заказов]
     FROM Сотрудники INNER JOIN Заказы
          ON Сотрудники.Код_Сотрудника = Заказы.Код_Сотрудника
     WHERE YEAR(ДатаИсполнения)=2017
     GROUP BY Сотрудник;
б) SELECT Сотрудник,YEAR(ДатаИсполнения),
          COUNT(*) AS [Количество заказов]
    FROM Сотрудники INNER JOIN Заказы
          ON Сотрудники.Код_Сотрудника = Заказы.Код_Сотрудника
    GROUP BY Сотрудник,YEAR(ДатаИсполнения)
    HAVING  YEAR(ДатаИсполнения)=2017;
в) SELECT  Категория, SUM(ОптоваяЦена*Количество)
                              AS [Стоимость складского запаса]
    FROM Категории AS К INNER JOIN Склад AS С
 ON К.Код_Категории = С.Код_Категории
    GROUP BY Категория  HAVING  AVG(Количество)>50;
г) SELECT  Город  FROM Города WHERE
       (SELECT  COUNT(*) FROM Клиенты
          WHERE Города.Код_Города=Клиенты.КодГорода)
          +  (SELECT  COUNT(*) FROM Поставщики
             WHERE Города.Код_Города=Поставщики.КодГорода) >3;
```

Листинг 3.11
Примеры групповой обработки данных

Запрос в) рассчитывает суммарную стоимость складского запаса товаров каждой из категорий, для которых среднее количество складского запаса превышает 50 единиц. В этом примере используются две агрегатные функции: SUM() в разделе SELECT и AVG() в разделе HAVING — обе эти функции производят обработку одних и тех же групп строк, сформированных в соответствии с условием выборки групп, заданным в разделе GROUP BY.

В запросе г) формируется список городов, в которых суммарное количество клиентов и поставщиков превышает 5. Агрегатная функция COUNT(*) использована в двух подчиненных запросах, не содержащих раздела GROUP BY и вложенных в раздел WHERE основного запроса. Каждый из подчиненных запросов вычисляется для каждой строки таблицы Города и возвращает скалярное значение — количество строк в таблице Клиенты (для первого подчиненного запроса) и количество строк в таблице Поставщики (для второго подчиненного запроса), соответствующих значению поля Код_Города в соответствующей строке таблицы Города.

Заметим, что автономное выполнение каждого из этих подчиненных запросов было бы невозможным, так как в условиях выборки строк этих запросов используется ссылка на столбец Код_Города таблицы Города, доступной в основном запросе.

6.4. Стандарты и диалекты языка SQL

6.4.1. История стандартизации языка SQL

Язык SQL (под названием SEQUEL — *Structured English QUEry Language*) был разработан корпорацией IBM в середине 1970-х гг. в рамках проекта экспериментальной реляционной СУБД System R [30].

Название («язык запросов») только частично отражает суть этого языка, который уже тогда являлся полноценным языком реляционных баз данных, содержащим не только операторы формулирования запросов выборки и модификации данных, но также и средства определения схемы БД и ограничений целостности, триггеров и хранимых представлений; поддержку структур физического уровня, обеспечивающих эффективное выполнение запросов, средств управления транзакциями и разграничения доступа пользователей к таблицам базы данных и отдельным их полям.

К началу 1980-х гг. уже существовали различные коммерческие версии этого языка, существенно отличающиеся от языка SQL System R, так как полная реализация всех идей System R была для того времени слишком сложной. В 1983 г. Международная организация по стандартизации (ISO) и Американский национальный институт стандартов (ANSI) приступили к разработке стандарта языка SQL.

Первый этап стандартизации языка SQL завершился к 1989 г., когда был принят международный стандарт **SQL89**, в котором многие аспекты языка не были детально прописаны — их предполагалось определять в реализации. Достижением SQL89 являлась стандартизация синтаксиса и семантики операторов выборки и манипулирования данными (SELECT, INSERT, DELETE, UPDATE) и средств ограничения целостности БД: первичного и внешних ключей (PRIMARY KEY и FOREIGN KEY), проверяемых (CHECK CONSTRAINTS) ограничений целостности.

В 1992 г. был введен существенно более полный стандарт языка SQL, получивший название **SQL92** и охватывающий практически все необходимые для реализации аспекты:

– манипулирование схемой БД;

– полноценное управление изолированностью транзакций;

– каскадное удаление и изменение данных в связанных отношениях;

– динамический SQL и встроенный SQL для использования в семи различных языках программирования;

– использование временных таблиц;

– использование подчиненных запросов в проверяемых ограничениях;

– расширенный набор типов данных и средства преобразования типов.

В последующих версиях были расширены возможности стандарта SQL92 и добавлены некоторые объектно-ориентированные возможности.

SQL99:

– использование UDT-типов (User-Defined Datatypes), определяемых пользователем с помощью SQL-оператора CREATE TYPE;

– использование определяемых пользователем нескалярных типов данных (массивов из скалярных элементов допустимых SQL-типов) и объектных типов данных, в том числе с поддержкой множественного наследования;

– работа с бинарными и символьными LOB-объектами (Large Object);

– поддержка конструкторов типов данных и значений строк таблиц;

– поддержка дополнительных возможностей ссылочной целостности, например использование подчиненных запросов в ограничениях целостности CHECK оператора CREATE TABLE;

– поддержка рекурсивных запросов и средств описания сложных запросов, востребованных в системах аналитической обработки данных (OLAP), например использование в операторе SELECT операции INTERSECT для формирования пересечения множеств, выданных несколькими SQL-запросами, и операции FULL OUTER JOIN для создания *полных внешних соединений* таблиц, содержащих все строки из соединяемых таблиц, с NULL-значениями в несовпадающих столбцах;

– использование PSM-модулей (*Persistent Stored* — постоянно хранимые), поддерживающих процедурные расширения языка: переменные, операторы управления CASE, IF, WHILE, REPEAT, LOOP и FOR, процедуры и функции, создаваемые операторами CREATE PROCEDURE и CREATE FUNCTION;

– вызов из SQL внешних программ, написанных на других языках программирования; при этом внешняя программа может создаваться так же, как и внутренние объекты базы данных — SQL-операторами CREATE PROCEDURE или CREATE FUNCTION с обязательным указанием параметров EXTERNAL и LANGUAGE.

SQL2003

В стандарте 2003 г. специфицирован ряд новых свойств языка:

– обновлен состав используемых *типов данных*:

- специфицирован новый тип данных, значениями которого являются XML-документы; для XML-типа определен набор операций, обеспечивающих доступ и преобразования значений типа XML;
- расширены возможности использования нескалярных типов данных:
  - во-первых, введен новый *конструктор типов мультимножеств* MULTISET;
  - во-вторых, в качестве элементов любого нескалярного типа допускается использование любого допустимого в SQL, в том числе и нескалярного, типа данных (кроме самого конструируемого нескалярного типа), тем самым полностью снято ограничение «плоских таблиц», исторически присущее реляционным (в том числе и SQL-ориентированным) базам данных;

– введено понятие *табличной функции* (*Table Value Function*), возвращающей значение типа мультимножества, элементы которого — строки таблицы; к результату, возвращаемому табличной функцией, можно адресовать SQL-запросы точно так же, как и к таблице или представлению, хранимым в базе данных;

– появились три новых возможности определения автоматически заполняемых столбцов таблиц:

- использование объектов базы данных нового типа — *генераторов последовательностей* (*sequence generators*), производящих последовательности изменяемых во времени уникальных автоинкрементных числовых данных; для работы с генераторами последовательностей предусмотрены операторы CREATE SEQUENCE, ALTER SEQUENCE, DROP SEQUENCE и функция NEXT VALUE FOR <имя генератора>;
- использование типа IDENTITY для определения столбцов с уникальными автоинкрементными целочисленными значениями;
- использование *генерируемых столбцов* (*generated columns*), значения которых при вставке строк таблицы формируются в результате вычисления заданного выражения, в котором допустимо использование констант и ссылок на основные (не генерируемые) столбцы этой таблицы.

В стандарте языка **SQL2006/2008** значительно расширены средства работы с XML-данными, появилась возможность совместного использования в запросах SQL и XQuery, а также устранены неоднозначности стандарта SQL2003.

SQL2011

Основным достижением стандарта 2011 г. является улучшение и развитие средств работы с временными (*temporal*) базами данных, в которых хранится информация, актуальная для определенных временных периодов.

В *битемпоральных* базах данных могут быть определены два периода: период *valid time* (или *application time*), в течение которого некоторый факт действителен в реальном мире, и период *transaction time* (или *system time*), в течение которого был известен факт, хранящийся в базе данных.

В *многотемпоральных* базах данных может быть определено более двух временных интервалов.

SQL2011 содержит языковые средства определения и манипулирования временными интервалами:

– для определения именованного временного интервала используются два стандартных столбца таблиц;

– для *application time*-таблиц:

- определение периода *valid time* (PERIOD FOR);
- обновление и удаление строк с автоматическим расщеплением временного интервала;
- определение временных первичных ключей (*temporal primary keys*) с контролем перекрытия интервалов (WITHOUT OVERLAPS);
- определения временных ограничений ссылочной целостности (*temporal referential integrity*);
- использование новых предикатов для обработки временных интервалов: CONTAINS, OVERLAPS, EQUALS, PRECEDES, SUCCEEDS, IMMEDIATELY PRECEDES и IMMEDIATELY SUCCEEDS;

– для *system time*-таблиц:

- определение таблиц с использованием PERIOD FOR SYSTEM_TIME и модификатора WITH SYSTEM VERSIONING;
- автоматическое сохранение интервалов *system time*;
- использование языковых конструкций AS OF SYSTEM TIME и VERSIONS BETWEEN SYSTEM TIME ... AND ... для упорядоченных во времени (*time-sliced* и *sequenced*) запросов.

6.4.2. Диалекты языка SQL

Язык SQL в его исходном виде являлся информационно-логическим языком декларативного типа, однако более поздние версии этого языка предусматривают возможность его процедурных и частично объектно-ориентированных расширений, что делает современный SQL полноценным языком программирования, на котором реализуются серверные компоненты пользовательских приложений.

Коммерческие СУБД реализуют собственные диалекты SQL, базирующиеся на различных его стандартах и, как правило, расширяющие возможности стандартного языка. В таблице 3.2 приведены фирменные наименования некоторых из широко распространенных диалектов SQL.

Таблица 3.2

Диалекты языка SQL

СУБД	Диалект языка SQL
IBM DB2	SQL PL (SQL Procedural Language)
MySQL	SQL/PSM (SQL/Persistent Stored Module)
InterBase/Firebird	PSQL (Procedural SQL)
Oracle	PL/SQL (Procedural Language/SQL)
Postgres, PostgreSQL	PL/pgSQL (Procedural Language/PostgreSQL)
Microsoft SQL-Server	Transact-SQL
Microsoft Jet/Access	Microsoft Jet SQL

Правила использования диалектов языка SQL размещены на официальных ресурсах разработчиков соответствующих СУБД.

Синтаксические конструкции языков программирования формулируются с использованием специальной нотации — так называемых форм Бэкуса — Наура (BNF). Стандарт BNF и пример BNF-формулы одного из SQL-операторов приведены в приложении А.

Microsoft Jet SQL

Основная причина рассмотрения в учебном пособии особенностей именно этого диалекта языка SQL — его относительная простота, что немаловажно на начальном этапе освоения такого специфического языка программирования, как SQL. По этой же причине практикум по SQL-программированию, предлагаемый в следующей главе учебного пособия, ориентирован именно на использование Microsoft Jet SQL и поддерживающей его СУБД MS ACCESS при выполнении практических заданий.

Microsoft Jet SQL разработан на основе стандарта SQL-89, однако в этом диалекте реализованы не все средства стандартного языка, имеются также до-

полнительные возможности, не поддерживаемые стандартным языком, например Microsoft Jet SQL позволяет использовать более мощные выражения, разрешает группировку и сортировку по выражениям, допускает реализацию перекрестных запросов и т. д.

Ниже приведена краткая сводка основных различий между этими языками.

1. Оператор ***Between...And***

Оператор имеет следующий синтаксис:

выражение [**NOT**] ***Between*** *значение_1* ***And*** *значение_2*

В языке Microsoft Jet SQL *значение_1* может превышать *значение_2*, а в SQL-89 *значение_1* должно быть меньше или равно *значение_2*.

2. ***Подстановочные символы*** предиката **LIKE** (табл. 3.3)

Таблица 3.3

Подстановочные символы предиката сравнения LIKE

Заменяемые символы	SQL-89	Microsoft Jet SQL
Любой одиночный символ	_ (подчеркивание)	**?**
Любое количество символов	**%**	*

3. Различные наборы ***типов данных***

В таблице 3.4 перечислены типы данных SQL-89, эквивалентные им типы данных языка Microsoft Jet SQL и допустимые синонимы для их именования.

Таблица 3.4

Сравнительная характеристика типов данных

SQL-89	Microsoft Jet SQL	Синонимы
BIT, BIT VARYING	BINARY	VARBINARY
Не поддерживается	BIT	BOOLEAN, LOGICAL, YES/NO
Не поддерживается	BYTE	INTEGER1
Не поддерживается	COUNTER	AUTOINCREMENT
Не поддерживается	CURRENCY	MONEY
DATE, TIME, TIMESTAMP	DATETIME	DATE, TIME, TIMESTAMP
Не поддерживается	GUID	
DECIMAL	Не поддерживается	
REAL	SINGLE	FLOAT4, IEEESINGLE, REAL
DOUBLE PRECISION, FLOAT	DOUBLE	FLOAT, FLOAT8, NUMBER, NUMERIC
SMALLINT	SHORT	INTEGER2, SMALLINT
INTEGER	LONG	INT, INTEGER, INTEGER4
INTERVAL	Не поддерживается	
Не поддерживается	LONGBINARY	GENERAL, OLEOBJECT
Не поддерживается	LONGTEXT	LONGCHAR, MEMO, NOTE
CHARACTER, CHARACTER VARYING	TEXT	ALPHANUMERIC, CHAR, CHARACTER, STRING, VARCHAR

Дополнительные возможности языка Microsoft Jet SQL

4. Оператор ***TRANSFORM***

Оператор TRANSFORM предназначен для создания так называемых ***перекрестных запросов***, результат выполнения которых представляется пользова-

телю в стиле электронной таблицы — в более компактной форме по сравнению со стандартным запросом выборки данных.

Ниже приведены формат этого оператора и описания его аргументов.

TRANSFORM *агрегатная_функция*
SELECT …
PIVOT *поле* [**IN** (*значение_1*[, *значение_2*[, ...]])]

Аргументы оператора:

– инструкция **TRANSFORM** — должна быть записана первой;

– *агрегатная_функция* — одна из агрегатных функций из числа поддерживаемых СУБД;

– инструкция **SELECT** может содержать:

- список полей, имена которых образуют заголовки строк перекрестной таблицы, записываемые в ее левом столбце;
- раздел **GROUP BY**, задающий параметры группировки по строкам;
- раздел **WHERE**, задающий условия выборки строк;
- подчиненные запросы в предложении **WHERE**;

– *поле* — имя столбца или выражение, которое содержит заголовки столбцов для результирующего набора;

– *значение_1, значение_2* и т. д. — фиксированные значения, используемые при создании заголовков столбцов (верхняя строка результирующей перекрестной таблицы).

Листинг 3.12 содержит пример перекрестного запроса, представляющего информацию о суммах выручки, полученных от реализации товаров, в координатах СОТРУДНИК — ТОВАР, а рисунок 3.1 — результат выполнения этого запроса.

```
TRANSFORM Sum(Заказано.Количество*
БазоваяЦенаРеализации*(1-Скидка)) AS [Сумма выручки]
SELECT Товар
FROM Склад INNER JOIN (Сотрудники INNER JOIN
(Заказы INNER JOIN Заказано
ON Заказы.Код_Заказа = Заказано.Код_Заказа)
ON Сотрудники.Код_Сотрудника =
    Заказы.Код_Сотрудника)
ON Склад.КодТовара = Заказано.Код_Товара
GROUP BY Товар
PIVOT  Сотрудник;
```

Листинг 3.12
Пример перекрестного запроса

5. Агрегатные функции ***StDev*** и ***StDevP***

Дополнительно к пяти стандартным агрегатным функциям язык Microsoft Jet SQL включает функции StDev(*выражение*) и StDevP(*выражение*), возвраща-

ющие соответственно смещенное и несмещенное значения среднеквадратичного отклонения, вычисляемого по набору значений, содержащихся в *выражении*.

Товар	Акбаев	Бабкина	Белова	Воронова	Кралев	Кротов	Крылова	Новиков	Ясенева
Alice Mutton	47385	210600	179010	63180	81608		78975		194805
Aniseed Syrup		43200	675	39825	16875			20250	34425
Boston Crab Meat	138669	232875	98615	144755	230577	52164	53406	179345	31919
Camembert Pierrot	110880	169785	11550	329175	140910	8778	99330	346731	11550
Carnarvon Tigers	25312	84375	25312	855562	262406	45562	255234	209883	
Chai	97200	137295	50957	293422	71685	17314	87723	66217	36450
Chang	25650	253614	69576	319471	79002	37770	6412	64638	52582
Chartreuse verte	52245	458055	96592	335340	73629	77760	64395	58927	42525
Chef Anton's Cajun S		22275		75364	32967		29700	95040	
Chef Anton's Gumbo		139789	212422	175673	244271	57645	203991	36028	
Chocolade	9811	34425	127372	79177	32532		69366	58953	24098
Cote de Blaye	2361125	1938701	2107671	1960045	42687	1775068	1458472	1351755	177862
Escargots de Bourgo	118236	97487	84697	198328	160674	123334	107325	113228	243672
Filo Mix	35957	21664	52589	8505	43942	72647	59417	31185	

Запись: 17 из 77

Рис. 3.1
Результат выполнения перекрестного запроса,
представленного на листинге 3.12

Аргумент *выражение* может быть именем столбца, содержащего обрабатываемые данные числового типа, или выражением, операндами которого могут быть имена столбцов, числовые константы или (нестатистические) функции, возвращающие числовые значения.

Если результат запроса содержит меньше двух строк (или не содержит строк для функции StDevP), эти функции возвращают значение Null (что означает невозможность вычисления среднеквадратичного отклонения).

6. ***Запросы с параметрами***

Запрос с параметрами помогает автоматизировать процесс изменения условий отбора запроса. При выполнении такого запроса значения параметров запрашиваются у пользователя и после ввода значений подставляются вместо имен параметров в текст запроса.

Раздел параметров SQL-запроса записывается перед разделом SELECT в соответствии со следующей BNF-формулой (приложение А):

PARAMETERS ИмяПараметра ТипДанных [,ИмяПараметра ТипДанных [, ...]];

ИмяПараметра — текст, который будет отображаться в окне диалога при выполнении запроса. При наличии пробелов или знаков препинания в имени параметра его следует заключить в квадратные скобки. Имена параметров допускается использовать в качестве переменных в разделах WHERE или HAVING запроса.

ТипДанных — один из базовых типов данных Microsoft Jet SQL или его синоним (табл. 4.2).

В листинге 3.13 рассчитывается сумма выручки от заказов, исполненных сотрудниками филиалов, при этом у пользователя запрашиваются значения параметров: имя сотрудника и годы исполнения заказов.

```
PARAMETERS
[Сотр] Text (16),
[Год_ОТ] Long,
[Год_ДО] Long;
SELECT Филиал, Город, Сотрудник,
SUM(Количество*БазоваяЦенаРеализации*(1-Скидка))
AS [Сумма выручки]
FROM Города INNER JOIN (Филиалы
INNER JOIN (Сотрудники
INNER JOIN (Заказы
INNER JOIN Заказано
ON Заказы.Код_Заказа = Заказано.Код_Заказа)
ON Сотрудники.Код_Сотрудника = Заказы.Код_Сотрудника)
ON Филиалы.Код_Филиала = Сотрудники.Код_Филиала)
ON Города.Код_Города = Филиалы.Код_Города
WHERE Сотрудник LIKE [Сотр] AND
    YEAR(ДатаРазмещения) BETWEEN [Год_ОТ] AND [Год_ДО]
GROUP BY Филиал, Город, Сотрудник;
```

Листинг 3.13
Пример использования параметризованного SQL-запроса

ГЛАВА 7. ПРАКТИКУМ ПО SQL-ПРОГРАММИРОВАНИЮ

7.1. Общие методические указания

Практические занятия нацелены на изучение базовых элементов языка SQL, освоение инструментальных средств, используемых при программировании и отладке SQL-запросов, и получение практических навыков работы в среде СУБД в процессе выполнения практических заданий, каждое из которых связано с написанием и отладкой некоторого SQL-запроса.

Все задания выполняются в учебной базе данных, схема которой приведена на рисунке 3.2. В качестве базовой СУБД рекомендуется использовать MS Access со встроенным языком Microsoft Jet SQL, что обосновывается следующими соображениями:

1) язык Microsoft Jet SQL является диалектом стандартного SQL-89, и его вполне достаточно для демонстрации типовых возможностей SQL;

2) MS Access поддерживает технологию визуального конструирования запросов с автоматической генерацией их SQL-кода, что представляется весьма полезным на начальной стадии освоения языка;

3) MS Access является стандартным Windows-приложением, входящим в комплект поставки популярного пакета MS Office, и не предъявляет особых требований к аппаратуре и системному ПО.

Перед выполнением практических заданий рекомендуется самостоятельно проанализировать и программно реализовать SQL-запросы, примеры которых приведены на листингах 3.3–3.13.

Защита практических заданий производится в форме демонстрации текстов подготовленных SQL-запросов и результатов их выполнения.

7.2. Учебная база данных

Для выполнения практических заданий предоставляется учебная база данных (файл mdb-формата), обеспечивающая процессы торгово-складского учета и анализа в торговой компании. Схема базы данных (в стиле MS Access) приведена на рисунке 3.2.

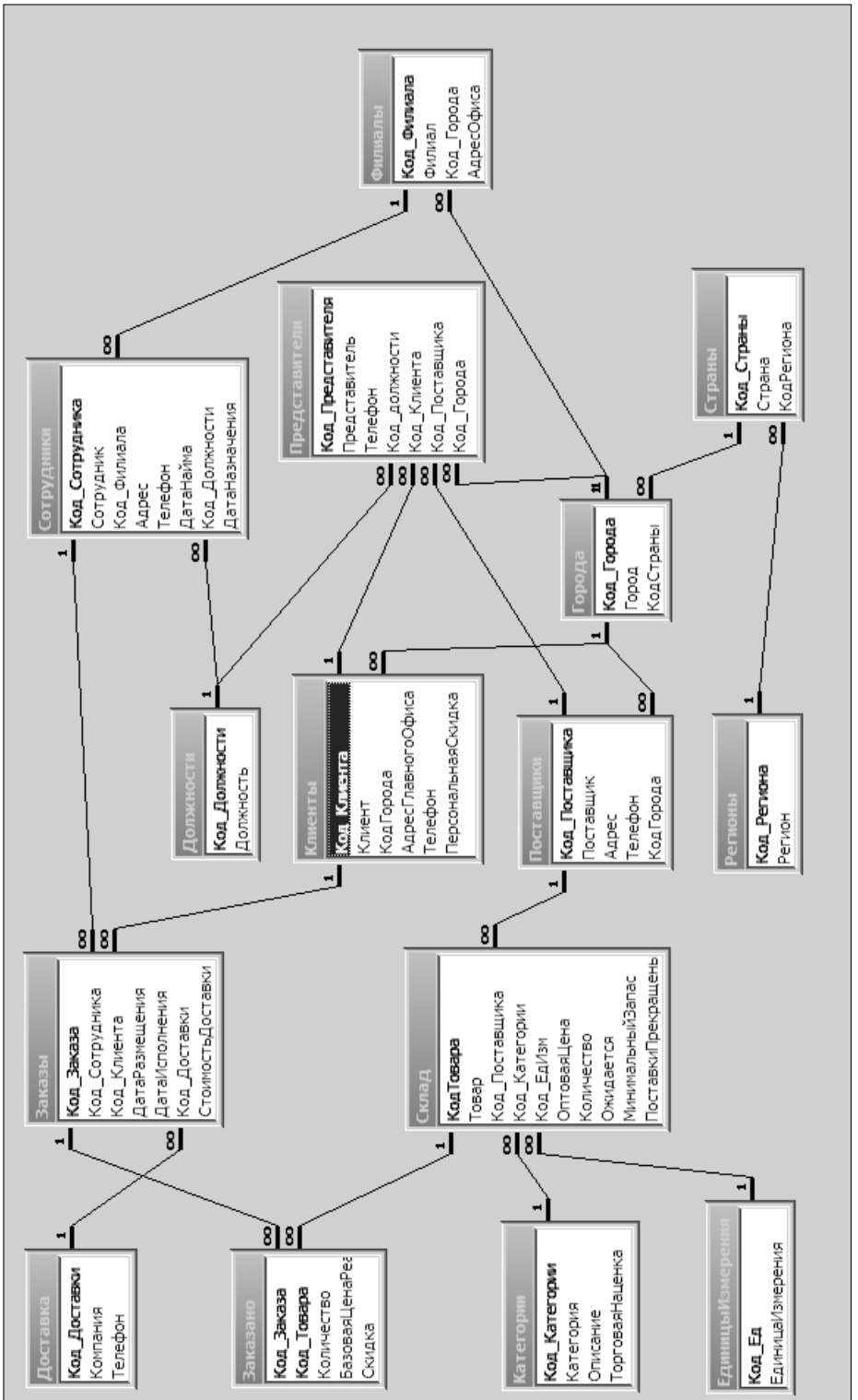


Рис. 3.2
Схема учебной базы данных

7.3. Практические задания

Задание № 1. Простейшие запросы выборки данных

1.1. Выбрать товары, складской запас которых превышает минимально допустимый запас не менее чем на 50%. Определить количество единиц и стоимость складского запаса по каждому такому товару.

1.2. Из числа товаров, имеющихся на складе, выбрать такие, поставки которых прекращены. Определить количество единиц и стоимость складского запаса по каждому такому товару.

1.3. Выбрать заказы, время исполнения которых превысило 1 месяц.

1.4. Выбрать сотрудников, стаж работы которых превышает ***n*** лет (***n*** задается параметром, значение которого запрашивается у пользователя).

1.5. Выбрать заказы, исполненные в ***k***-м квартале ***p***-го года (год ***p*** и номер квартала ***k*** задаются параметрами запроса).

1.6. Получить список городов для страны, код которой задается параметром.

Задание № 2. Запросы с соединением таблиц

2.1. Прокомментировать SQL-запросы, представленные в листинге 3.14, и оценить результаты их выполнения.

```
а) SELECT Город, Страна FROM Страны, Города;
б) SELECT Город, Страна FROM
       Страны AS С LEFT JOIN Города AS Г
       ON С.КодСтраны = Г.КодСтраны;
в)  SELECT Город, Страна
    FROM    Страны AS С, Города AS Г  WHERE  Г.КодСтраны=С.КодСтраны;
г)  SELECT Город, Страна  FROM  Страны AS С RIGHT JOIN Города AS Г
    ON С.КодСтраны = Г.КодСтраны;
д)   SELECT Город, Страна   FROM
     Страны AS С INNER JOIN Города AS Г
     ON С.КодСтраны = Г.КодСтраны;
```

Листинг 3.14

Примеры SQL-запросов с соединением таблиц

2.2. Сформировать список сотрудников по филиалам, городам, странам и регионам.

2.3. Выбрать товары, заказанные в первом квартале 2017 г. клиентами из Европы, которым товары были доставлены по почте. Отсортировать по названиям стран и городов.

2.4. Выбрать товары, заказанные во втором квартале 2001 г. клиентами из Америки, которым товары были доставлены по почте.

2.5. Выбрать товары, заказанные в первом квартале 2001 г. клиентами из России и Белоруссии, которым товары были доставлены по почте.

2.6. Выбрать зарубежных поставщиков, остаток товаров которых на складе не превышает установленного минимального запаса.

2.7. Выбрать поставщиков из Северной Америки, поставки товаров которых прекращены.

2.8. Выбрать рыбные и мясные товары (категория и марка товара, поставщик, оптовая цена, единица измерения, складской запас и ожидаемое количество), поставки которых продолжаются.

2.9. Для каждого клиента, разместившего заказ, выбрать поставщиков заказанных товаров, находящихся в том же городе, что и клиент.

2.10. Для каждого клиента, разместившего заказ, выбрать поставщиков заказанных товаров, находящихся в той же стране, что и клиент.

2.11. Для каждого клиента, разместившего заказ, выбрать поставщиков заказанных товаров, находящихся в том же регионе, что и клиент.

2.12. Выбрать товары, заказчики которых (клиенты) находятся в том же городе, что и поставщики заказанных товаров.

2.13. Выбрать товары, заказчики которых (клиенты) находятся в той же стране, что и поставщики заказанных товаров.

2.14. Выбрать товары, заказчики которых (клиенты) находятся в том же регионе, что и поставщики заказанных товаров.

Задание № 3. Статистическая обработка данных

3.1. Прокомментировать SQL-запросы, представленные в листинге 3.15, и оценить результаты их выполнения.

3.2. Определить общее количество заказов, оформленных каждым из сотрудников компании в первом квартале 2017 г.

3.3. Определить общее количество заказов, оформленных каждым из сотрудников компании в каждом квартале 2017 г.

3.4. Определить суммарную стоимость товаров в каждом из исполненных заказов.

3.5. Определить средний срок исполнения заказа по каждому региону.

3.6. Определить суммарную стоимость каждой категории товаров, включенных в заказы 2017 г.

3.7. Определить среднюю оптовую цену имеющихся на складе товаров по категориям.

3.8. Определить количество и общую стоимость имеющихся на складе товаров для каждого из поставщиков.

3.9. Определить суммарную стоимость доставки заказов каждым из способов в каждом квартале 2017 г.

3.10. Определить суммарную стоимость доставки заказов каждым из способов в каждую из стран клиентов, заказавших товары.

3.11. Определить заказы, стоимость которых превышает среднюю.

3.12. Выбрать города, количество клиентов из которых превышает заданное значение.

3.13. Выбрать заказы, при выполнении которых товары, поставляемые из той же страны, из которой поступил заказ, составляли не менее половины стоимости заказа.

3.14. Выбрать заказы, поступившие из США, в которых заказано рыбопродуктов на сумму большую, чем средняя стоимость аналогичной продукции в заказах клиентов из других стран (указать код заказа, наименование клиента и сумму, уплаченную за рыбопродукты).

```
а)  SELECT COUNT(*) FROM Города;
б) SELECT COUNT(*), Города.КодСтраны, Страны.КодРегиона
   FROM Регионы INNER JOIN (Страны INNER JOIN Города
        ON Страны.КодСтраны = Города.КодСтраны)
        ON Регионы.КодРегиона =  Страны.КодРегиона
   GROUP BY  Города.КодСтраны,  Страны.КодРегиона;
в)   SELECT COUNT(КодСтраны) FROM Города;
г)    SELECT COUNT(*), КодСтраны FROM Города;
д)   SELECT COUNT(*), КодСтраны FROM Города  GROUP BY КодСтраны ;
е)   SELECT AVG(Товары.ЦенаПоставщика),Категории.Категория
     FROM Категории INNER JOIN Склад ON  Категории.Код_Категории =
            Склад.КодКатегории
     GROUP BY  Категории.Категория  HAVING  AVG(Склад.Количество)>30;
ж)  SELECT COUNT(*), КодСтраны  FROM Города
    GROUP BY КодСтраны  HAVING Count(*)>10;
и)  SELECT COUNT(*), Города.КодСтраны, Страны.КодРегиона
   FROM   Регионы INNER JOIN (Страны INNER JOIN Города ON
    Страны.КодСтраны = Города.КодСтраны)
    ON Регионы.Код_Региона = Страны.КодРегиона
  GROUP BY Страны.КодРегиона, Города.КодСтраны;
к) SELECT  Город,  COUNT(*)  AS [Количество клиентов]
  FROM Города INNER JOIN Клиенты
          ON Города.Код_Города =  Клиенты.КодГорода
  GROUP BY Город;
л)   SELECT  Город  FROM Города
    WHERE (SELECT  COUNT(*) FROM Клиенты
              WHERE Города.Код_Города = Клиенты.КодГорода)>2;
```

Листинг 3.15
Примеры SQL-запросов с групповой обработкой данных

Задание № 4. Модифицирующие SQL-запросы

4.1. Запросом к таблице Заказы создать новую таблицу T1, содержащую всю информацию о заказах, размещенных клиентами в 2017 г.

4.2. Исключить из таблицы T1 все записи о заказах, полученных от клиентов из Северной Америки.

4.3. Создать новую таблицу T2, содержащую следующую информацию о заказах, размещенных в первом квартале 2016 г.:

– код заказа и дата его размещения;

– сведения о клиенте (клиент, город, страна, регион);

– сведения о заказанных товарах (категория, товар, количество в заказе, сумма, уплаченная за товар);

4.4. На базе таблицы T2 создать таблицу для заказчиков из Европы.

4.5. На базе таблицы T2 создать таблицу для заказчиков из Америки.

4.6. В таблице Заказы продлить на 30 дней срок исполнения заказов, полученных от клиентов из Франции и Испании. Отменить все эти изменения.

4.7. Увеличить на 10 лет все даты в таблице Заказы.

4.8. Для всех заказанных товаров обновить базовую цену их реализации в соответствии с торговой наценкой, заданной для категории товара.

4.9. Для всех заказанных товаров обновить величину скидки в соответствии со значением персональной скидки клиентов, заказавших эти товары.

Задание № 5. Запросы с объединением таблиц

5.1. На базе таблиц Клиенты и Поставщики составить запрос для получения объединенного списка контрагентов.

5.2. На базе таблиц Регионы, Страны и Города составить запрос для получения объединенного списка, содержащего поля *Наименование*, *Код* и дополнительное поле *Тип*, содержащее значения: «Город» — для городов, «Страна» — для стран и «Регион» — для регионов.

5.3. На базе запросов, созданных при выполнении заданий 5.1 и 5.2, создать две новые базовые таблицы.

5.4. Объединить таблицы Сотрудники и Представители в одну новую базовую таблицу.

Задание № 6. Перекрестные запросы

6.1. Составить перекрестный запрос, позволяющий представить информацию о суммах, полученных от клиентов за исполненные заказы (стоимость товаров + доставка), в координатах СОТРУДНИК — СТРАНА ПОЛУЧАТЕЛЯ.

6.2. Составить перекрестный запрос, позволяющий представить информацию о суммах выручки, полученных от реализации товаров, в координатах ПОСТАВЩИК — КЛИЕНТ.

6.3. Составить перекрестный запрос, позволяющий представить информацию о суммах выручки, полученных от реализации товаров, в координатах **ПОСТАВЩИК — СОТРУДНИК**.

6.4. Составить перекрестный запрос, позволяющий представить информацию о суммах выручки, полученных от реализации товаров, в координатах **КЛИЕНТ — ТОВАР**.

6.5. Составить перекрестный запрос, позволяющий представить информацию о суммах выручки, полученных от реализации товаров, по кварталам каждого года.

6.6. Составить перекрестный запрос, позволяющий сопоставить информацию о стоимости доставки товаров каждым из способов доставки по странам с суммами соответствующих заказов.

ЧАСТЬ 4.
УПРАВЛЕНИЕ
И АДМИНИСТРИРОВАНИЕ

ГЛАВА 8. ОБЗОР ФУНКЦИЙ СУБД

Как уже отмечалось, существенной особенностью АИС, отличающей их от многих других программных систем, является обеспечение автономности подсистемы хранения данных, в основе которой — принцип независимости данных от прикладных программ, обрабатывающих хранимые данные в интересах конечных пользователей системы.

Реализация принципа автономности баз данных потребовала совместного файлового хранения метаданных с основными данными, представления модели данных на трех иерархических уровнях — концептуальном, логическом и физическом, запрета непосредственного доступа к базе данных со стороны прикладных программ и лишения их возможности (и необходимости) интерпретации низкоуровневых структур данных. В результате была сформирована концепция СУБД (рис. 1.1) как специализированной программной системы, обеспечивающей решение всего комплекса задач управления базами данных.

Обсуждение всех функций СУБД и методов их реализации выходит за рамки данного издания, ниже приведен лишь их обзор с краткими комментариями и более детально рассмотрены две важнейшие и взаимосвязанные функции СУБД: *управление транзакциями* и *управление блокировками*, обеспечивающие эффективную работу АИС в условиях интенсивного многопользовательского доступа к базам данных.

Реализация функций поддержки физической модели данных, трансляции SQL-запросов и управления производительностью, а также функций защиты от несанкционированного доступа к данным рассмотрена применительно к СУБД MS SQL-Server.

Поддержка физической модели данных. Физическая модель представлена множеством взаимосвязанных низкоуровневых структур данных, обеспечивающих их внутреннее (как межфайловое, так и внутрифайловое) представление.

Физическая модель недоступна пользовательским программным приложениям АИС, на основе этой модели в СУБД реализованы низкоуровневые алгоритмы исполнения запросов к базе данных — так называемые *процедурные планы*. СУБД обеспечивает взаимодействие компонентов логической и физической моделей базы данных, а также отображение структур физической модели на файловые структуры, поддерживаемые операционной системой, управляющей аппаратным комплексом АИС.

Поддержка системного каталога базы данных. Системный каталог (называемый также *словарем данных*) обеспечивает хранение в самой базе данных *метаданных* — описаний объектов логической и физической моделей данных, что позволяет прикладным программам оперировать исключительно объектами логической модели, не имея информации об их низкоуровневом представлении.

Наличие системного каталога обеспечивает независимость данных от обрабатывающих их прикладных программ и относительную автономность функ-

ционирования базы данных в составе прочих компонентов программного обеспечения АИС.

Системный каталог пользовательской базы данных представлен множеством системных таблиц, а также хранимых в базе данных представлений, функций и процедур, используемых как самой СУБД в процессе трансляции и исполнения SQL-запросов, так и администраторами баз данных для анализа, настройки и оптимизации работы системы.

Трансляция SQL-запросов. Процесс трансляции SQL-запроса включает отображение затрагиваемых запросом объектов логической модели данных на соответствующие объекты физической модели и формирование оптимального процедурного плана его исполнения. Оптимизация процедурных планов производится с целью повышения производительности работы системы на основании информации о наличии индексных структур данных и в соответствии с результатами автоматически выполняемого «сбора статистики» о текущем состоянии объектов базы данных.

Управление производительностью. Основным критерием оценки производительности системы является среднее время ее отклика на пользовательские SQL-запросы. Для минимизации времени отклика СУБД использует различные индексные структуры данных и предоставляет администраторам средства мониторинга пользовательской активности, обновления статистической информации о состоянии объектов базы данных, а также средства анализа процедурных планов исполнения запросов.

Производительность СУБД в режиме многопользовательского доступа к данным во многом определяется настройкой и реализацией *подсистемы управления транзакциями и блокировками*.

Управление надежностью. Надежное функционирование базы данных предполагает сохранение ее целостности и работоспособности в течение длительного периода времени и обеспечивается различными средствами на всех стадиях ее жизненного цикла. СУБД выполняет следующие функции управления надежностью на стадии эксплуатации базы данных:

– автоматический контроль заданных разработчиком ограничений целостности данных, в том числе ограничений типа и домена, проверяемых (check constraints) и ссылочных (foreign key) ограничений;

– контроль взаимовлияния транзакций, конкурирующих за доступ к объектам базы данных в многопользовательских системах, которое может приводить к искажениям результатов выполнения SQL-запросов (с точки зрения клиентских приложений, инициировавших транзакции);

– создание и надежное хранение резервных копий баз данных (в том числе и разностных копий) с возможностью восстановления по ним баз данных после «жестких сбоев», связанных с потерей данных на внешних запоминающих устройствах;

– поддержку журналов транзакций (LOG-файлов), обеспечивающих хранение «исторической» информации об операциях, модифицировавших базу данных;

– эффективное управление журналом транзакций в соответствии с протоколом WAL (Write Ahead LOG), гарантирующим сохранение журнальной информации во внешней памяти прежде, чем там будут сохранены результаты соответствующих модифицирующих операций;

– восстановление согласованного состояния базы данных по журналу транзакций после «мягких сбоев», которые не приводят к потере информации, размещенной на внешних запоминающих устройствах;

– автоматическое восстановление согласованного состояния базы данных по журналу транзакций после неудачного завершения (отката) транзакций.

Защита от несанкционированного доступа. Коммерческие СУБД общего применения предоставляют стандартный набор возможностей дискреционной (логической) защиты информации: идентификация и аутентификация пользователей (субъектов доступа), разграничение прав доступа субъектов к логическим объектам базы данных с возможностью группировки как субъектов, так и объектов доступа, шифрование хранимых данных. Специальные СУБД обеспечивают более высокий уровень информационной безопасности, используя методы мандатной (физической) защиты данных.

Инструментальная поддержка. СУБД предоставляют программистам и администраторам баз данных разнообразные программные и визуальные средства поддержки процессов разработки и управления:

– визуализация схем баз данных;

– визуальное конструирование и написание SQL-запросов, хранимых представлений, пользовательских функций и процедур;

– просмотр и обновление различных статистических характеристик объектов базы данных;

– анализ эффективности процедурных планов исполнения SQL-запросов;

– просмотр объектов системного каталога;

– сознание, корректировка и группирование субъектов доступа;

– настройка системы разграничения доступа к данным и др.

ГЛАВА 9. ЗАДАЧИ АДМИНИСТРИРОВАНИЯ БАЗ ДАННЫХ

Эксплуатационные характеристики хранилища данных, интегрированного в состав автоматизированной информационной системы, во многом определяют эффективность работы всего программного комплекса. Грамотное администрирование базы данных может существенно повысить производительность информационной системы, обеспечить высокую надежность хранения и требуемый уровень защиты информации.

Администрирование как вид профессиональной деятельности направлено на поддержание эффективной и бесперебойной работы баз данных, обеспечивающих функционирование информационных систем, и связано с выполнением следующих основных функций:

– эксплуатация серверов баз данных;

– поддержание баз данных в актуальном состоянии и обеспечение их эффективного функционирования;

– обеспечение доступности данных для легальных пользователей и защита от несанкционированного доступа к данным;

– мониторинг и идентификация потребностей пользователей.

Приведенный набор типовых функций администраторов баз данных закреплен как российскими, так и зарубежными профессиональными стандартами и классификационными системами, в которых явно разделяется деятельность по созданию баз данных (разработка концепций, проектирование и программирование) и деятельность по их сопровождению и поддержанию работоспособности (администрирование).

Так, в стандарте США SOC (Standard Occupational Classification) (приложение Б [1]) работы по сопровождению баз данных включены в подкатегорию «Database Administrators» в составе категории «Database and Systems Administrators and Network Architects», а функции по созданию баз данных определены в подкатегории «Software Developers», входящей в состав категории «Software and Web Developers, Programmers and Testers».

Соответствующий европейский стандарт European ICT Professional Profiles (приложение Б [2]) предусматривает профиль «Developer» для деятельности по проектированию баз данных и отдельный профиль «Database Administrator», в котором определены работы по обеспечению их функционирования в процессе эксплуатации.

Российские профессиональные стандарты «Архитектор программного обеспечения», «Руководитель разработки программного обеспечения» и «Программист» определяют функции проектирования и программирования баз данных, а комплекс работ по их сопровождению регламентирован стандартом «Администратор баз данных» (приложение Б [3]), в котором приведен деталь-

ный перечень трудовых функций с указанием для каждой из них квалификационного уровня, требований к профессиональной компетентности, базовому образованию и опыту работы. Стандарты «Специалист по информационным системам» (приложение Б [4]), «Системный администратор информационно-коммуникационных систем» (приложение Б [5]) и «Специалист по защите информации в автоматизированных системах» (приложение Б [6]) также регламентируют отдельные аспекты администрирования баз данных.

ГЛАВА 10. УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ И БЛОКИРОВКАМИ

Важнейшими компонентами СУБД являются менеджеры транзакций и блокировок, обеспечивающие конкурентный многопользовательский доступ к объектам базы данных: наложение и снятие блокировок объектов базы данных, обрабатываемых транзакциями, эффективный откат транзакций при невозможности их штатного завершения, управление распределенными транзакциями.

10.1. Понятие и базовые свойства транзакций

Стандартом SQL/92 определено понятие транзакции как последовательности SQL-операторов, рассматриваемых как единое целое в контексте их выполнения и возможной отмены результатов произведенных ими модификаций базы данных. Формально транзакцией будет считаться любая последовательность SQL-операторов, заключенных в «операторные скобки» BEGIN TRANSACTION и COMMIT (при успешном завершении транзакции) или ROLL BACK (при невозможности ее успешного завершения). При отсутствии таких «скобок» СУБД будет считать транзакцией каждый отдельный SQL-оператор.

Стандартом определены 4 базовых свойства транзакций, обозначаемых англоязычной аббревиатурой **ACID**: **A**tomicity (атомарность), **C**onsistency (согласованность), **I**solation (изолированность) и **D**urability (долговременность).

Свойство *атомарности* (**A**) реализует принцип «или все, или ничего» и предписывает рассматривать транзакцию как единое целое: если какой-либо из операторов, включенных в транзакцию, не может быть выполнен, СУБД обязана сделать откат (ROLL BACK) к началу транзакции, отменив все изменения объектов базы данных, произведенные предшествующими операторами этой транзакции.

Свойство атомарности применимо и к транзакциям, содержащим единственный оператор: например, если оператором UPDATE производится модификация 1000 строк таблицы, удовлетворяющих некоторому условию, то либо все эти строки будут модифицированы, либо, при невозможности успешного завершения этой операции, будет выполнен откат транзакции и модифицируемая таблица останется в исходном состоянии.

Согласованность (**C**) транзакции требует от СУБД гарантий того, что к моменту начала любой транзакции и в момент ее завершения база данных будет находиться в согласованном состоянии. Фактически это допускает рассогласованное состояние базы данных «внутри» транзакции, что весьма существенно при необходимости последовательного внесения изменений во множество взаимосвязанных таблиц.

Изолированность (**I**) — одно из важнейших свойств транзакций, которое далее будет рассмотрено более детально.

Транзакция всегда выполняется от имени определенного пользователя, идентификатор которого присутствует (явно или неявно) в качестве префикса

имени транзакции в операторе BEGIN TRANSACTION, поэтому, изолируя транзакции, СУБД обеспечивает и определенный уровень изолированности пользователей в многопользовательской системе.

Реализация изолированности транзакций должна исключить взаимовлияние параллельно выполняемых транзакций, конкурирующих в доступе к одному и тому же объекту базы данных, и при этом для каждого пользователя, инициировавшего выполнение транзакции, должна быть создана достоверная иллюзия того, что он в системе один. Достоверность такой иллюзии подтверждается (для пользователя) двумя основными факторами:

– каждый пользователь, анализируя результаты выполнения инициированных им запросов, должен быть уверен, что никакие другие пользователи не модифицируют данные, обрабатываемые транзакцией этого пользователя;

– пользователь не должен ощущать существенного увеличения *времени отклика* на его запросы к базе данных и в целом снижения производительности системы из-за ожидания освобождения ресурсов, временно заблокированных конкурирующими транзакциями.

Обеспечение свойства *долговременности* (**D**) транзакции должно гарантировать сохранение в файлах базы данных всех ее изменений, произведенных в буферных областях оперативной памяти каждой успешно завершенной (COMMIT) транзакцией.

Заметим, что в соответствии с протоколом WAL (Write Ahead LOG), непосредственно перед записью изменений во внешнюю память информация об этих изменениях также будет сохранена в файле журнала транзакций.

10.2. Конфликты между транзакциями

Поддержка требования *согласованности* транзакций допускает рассогласованное состояние базы данных в процессе их выполнения, а реализация требований *атомарности* и *долговременности*, с одной стороны, должна обеспечить возможность отката (ROLLBACK) частично выполненных транзакций, а с другой — гарантировать фиксацию в базе данных всех изменений, произведенных успешно завершенными (COMMIT) транзакциями.

В многопользовательских системах все это может создавать различные проблемы как у транзакций-читателей (R), осуществляющих пассивный доступ к данным (SELECT), так и у транзакций-писателей (W), модифицирующих данные (DELETE, INSERT, UPDATE).

Проблема *потерянного изменения* отражает конфликт типа «W–W» между параллельно выполняемыми транзакциями-писателями, конкурирующими в доступе к одному объекту базы данных. Если, например, каждая из транзакций по-своему модифицировала таблицу, то в базе данных будут сохранены только те изменения, которые были сделаны транзакцией, последней модифицировавшей объект, а все более ранние изменения, сделаны другой транзакцией, будут потеряны («затерты» второй транзакцией). При этом согласованное состояние базы данных в целом не будет нарушено, но пользователь, инициировавший первую (успешно завершившуюся раньше) транзакцию, так и

не увидит результатов ее завершения. Очевидно, что для решения рассмотренной проблемы СУБД должна обеспечить изолированное выполнение двух конкурирующих транзакций-писателей.

Проблема *чтения грязных данных* отражает конфликт типа «W–R» и может проявляться в результате конкуренции транзакции, модифицирующей объект, с другой транзакцией, параллельно читающей данные этого же объекта и принимающей некоторые решения в соответствии с прочитанной информацией.

Если СУБД производит откат (ROLLBACK) первой транзакции, то все решения второй транзакции оказываются некорректными, так как они были приняты на основании ложной (еще не зафиксированной в БД) информации. Если первая транзакция все же завершается успешно (COMMIT), то и в этом случае у второй транзакции могут быть проблемы с корректностью принятых решений, так как в соответствии с *требованием согласованности* допускается рассогласованное состояние БД в процессе выполнения первой транзакции. Для решения рассмотренной проблемы СУБД должна изолировать транзакцию-писателя, запретив другим транзакциям читать соответствующие объекты БД до момента завершения первой транзакции.

Проблема *неповторяемого чтения* отражает конфликт типа «R–W»: первая транзакция многократно читает один и тот же объект базы данных и при этом каждый раз «видит» его в различных состояниях, так как в промежутках между чтениями этот объект изменяет другая транзакция. Обе транзакции могут завершиться успешно (COMMIT), и проблем с согласованностью базы данных не будет, однако у транзакции-читателя остается проблема несоответствия полученных результатов, решением которых должна заниматься СУБД, обеспечивающая изолированность первой транзакции и запрещающая другим транзакциям модифицировать объект до ее завершения.

Последняя из стандартных проблем, связанных с недостаточной изолированностью транзакций, получила название ***проблемы чтения кортежей-фантомов***. Эта проблема также отражает конфликт типа «R–W» и проявляется в ситуациях, когда одна транзакция многократно сканирует таблицу, производя при каждом сканировании обработку множества ее строк, соответствующих одному и тому же логическому условию, а другая транзакция, независимо от первой, производит вставку в эту таблицу или удаление из нее строк, соответствующих этому же условию.

Классический пример:

1) *первая транзакция* при первом сканировании таблицы производит выборку ее строк по некоторому условию и по результатам выборки вычисляет какую-либо статистическую характеристику (например, среднее значение одного из атрибутов);

2) *вторая транзакция* производит вставку или удаление строк, соответствующих этому же условию;

3) *первая транзакция*

– при повторном сканировании таблицы выбирает ее строки по тому же условию, и в эту выборку попадут кортежи-фантомы, вставленные в таблицу

второй транзакцией (или, наоборот, в выборке не окажется кортежей, фантомно присутствовавших при первом сканировании и затем удаленных второй транзакцией);

– в выбранных строках модифицирует значение другого атрибута в соответствии со значением статистической характеристики, вычисленной по результатам первого сканирования;

4) *вторая транзакция*

– завершается успешно (COMMIT), или производится ее откат (ROLLBACK); в любом случае она уже причинила вред первой транзакции, выполнившей некорректную обработку данных.

10.3. Уровни изолированности транзакций

Стандарт SQL-92 определяет 4 уровня изолированности транзакций — от самого слабого нулевого уровня до самого сильного — третьего (табл. 4.1).

Таблица 4.1

Уровни изолированности транзакций

Уровни изолированности		Решаемые проблемы (конфликты между транзакциями)			
		Потерянные изменения	Чтение грязных данных	Неповторяющееся чтение	Кортежи-фантомы
0	READ UNCOMMITTED	Да	Нет	Нет	Нет
1	READ COMMITTED	Да	Да	Нет	Нет
2	REPEATABLE READ	Да	Да	Да	Нет
3	SERIALIZABLE	Да	Да	Да	Да

На каждом уровне изолированности СУБД обеспечивает разрешение определенных конфликтов между конкурирующими транзакциями, при этом на каждом следующем (более сильном) уровне решаются и проблемы всех предыдущих (более слабых) уровней.

Минимальный (нулевой) уровень **READ UNCOMMITTED** решает только проблему потерянного изменения, запрещая двум любым транзакциям параллельно модифицировать один и тот же объект базы данных. Транзакции, требующие модификации объекта, будут ожидать успешного завершения или отката транзакции-конкурента, первой начавшей модифицировать этот объект. При этом доступ к объекту со стороны транзакций-читателей не запрещается и сохраняется вероятность чтения грязных данных, сформированных еще не завершенными транзакциями-писателями.

1-й уровень **READ COMMITTED** дополнительно блокирует доступ транзакций-читателей к объектам, находящимся в стадии обработки транзакциями-писателями, что обеспечивает решение проблемы чтения грязных данных, но допускает неповторяющееся чтение, так как не запрещает модифицировать объекты, обрабатываемые транзакциями-читателями.

На 2-м уровне изолированности **REPEATABLE READ** дополнительно решается проблема неповторяющегося чтения, так как СУБД блокирует возможность модификации (UPDATE) объекта транзакциями-писателями в течение всего периода обработки этого объекта транзакцией-читателем.

Максимальный 3-й уровень изолированности SERIALIZABLE обеспечивает полную независимость транзакций друг от друга и гарантирует, что никакие транзакции-писатели не смогут вставить в таблицу или удалить из нее строки, соответствующие условию выборки данных из этой таблицы транзакцией-читателем.

На уровне **SERIALIZABLE** дополнительно решается *проблема чтения кортежей-фантомов*, и результат выполнения всех конкурирующих параллельных транзакций будет точно таким же, как и в случае их реально *последовательного* выполнения (откуда и название этого уровня), когда каждая очередная транзакция начинается только после завершения предыдущей.

Чем выше уровень изолированности транзакций, тем более надежно будет работать система, однако платой за это будет увеличение объема системных ресурсов, требуемых для управления транзакциями, и снижение производительности за счет увеличения интервалов ожидания одними транзакциями освобождения объектов БД, обрабатываемых другими транзакциями.

Как правило, СУБД по умолчанию поддерживает некоторый уровень изолированности транзакций (например, для MS SQL-Server это уровень READ COMMITTED), однако разработчик вправе назначить требуемый уровень изолированности индивидуально для каждой транзакции, определив тем самым степень влияния на ее операции других параллельно выполняемых транзакций, а также степень влияния данной транзакции на операции транзакций-конкурентов. Соответствующие примеры будут рассмотрены в п. 10.5.1.

10.4. Управление блокировками

Блокировка — это механизм, с помощью которого СУБД синхронизирует параллельный доступ нескольких транзакций к одному и тому же объекту базы данных. Перед тем как транзакция получит доступ к объекту для его чтения или модификации, она должна убедиться в том, что объект не заблокирован другими транзакциями, и, если он свободен, транзакция запрашивает у СУБД блокировку этого объекта, чтобы защитить его от изменений другими транзакциями во время своего выполнения.

Временная блокировка объектов базы данных — основной (хотя и не единственный) метод обеспечения требуемого уровня изолированности транзакций, реализацией которого занимается специальный компонент СУБД — менеджер блокировок, работающий совместно с другим ее важнейшим компонентом — менеджером транзакций. Функции двух этих менеджеров весьма разнообразны, различаются также и способы реализации этих функций в разных СУБД, ниже приведена упрощенная схема, поясняющая алгоритм их взаимодействия, обеспечивающий требуемый уровень изолированности транзакций.

Менеджер транзакций:

– получает от транслятора SQL-кода информацию о транзакциях, требуемых уровнях их изолированности, а также о составе операций каждой транзакции и объектах базы данных, затрагиваемых этими операциями;

– сохраняет полученную информацию в системном каталоге БД;

– формирует очереди транзакций, конкурирующих в доступе к объектам БД;

– фиксирует (COMMIT) результаты успешно завершенных транзакций или производит откат (ROLLBACK) транзакций в случае невозможности их успешного завершения;

– удаляет из очереди завершенные транзакции;

– разрушает *тупиковые блокировки* (п. 10.4.3) в случае их обнаружения менеджером блокировок:

- ранжирует транзакции, участвующие в тупиковой блокировке, используя поддерживаемую СУБД модель стоимости транзакции;
- выбирает транзакцию, имеющую минимальную стоимость;
- выполняет принудительный откат (ROLLBACK) этой транзакции;
- циклически повторяет процесс принудительного отката транзакций до тех пор, пока тупиковая блокировка не будет разрушена.

Менеджер блокировок:

– производит мониторинг очередей транзакций в соответствии с информацией, сохраненной менеджером транзакций в системном каталоге базы данных;

– принимает решения о наложении блокировок на объекты, требующие обработки очередными транзакциями:

- выбирает *режим блокирования* (п. 10.4.2) объекта в соответствии с требуемым уровнем изолированности транзакции;
- выбирает оптимальный *уровень блокируемого объекта* (п. 10.4.1) по критерию минимизации затрат ресурсов на поддержание выбранного режима блокирования и в соответствии с требуемым уровнем изолированности транзакции;

– принимает решения о снятии блокировок с объектов, освобождаемых завершенными транзакциями;

– сохраняет в системном каталоге информацию о временно заблокированных объектах БД;

– идентифицирует (прогнозирует) ситуации с *тупиковыми блокировками* (п. 10.4.3), которые не могут быть разрешены естественным путем при освобождении завершенными транзакциями заблокированных ими объектов.

10.4.1. Уровни блокирования ресурсов

СУБД может блокировать объекты как логической, так и физической[5] моделей данных различных иерархических уровней, типичный набор которых приведен в таблице 4.2.

Высокоуровневые блокировки таких объектов, как *база данных*, *файл* или *таблица*, очень экономичны — их реализация не требует больших затрат системных ресурсов хотя бы потому, что количество таких «крупных» объектов относительно невелико.

[5] Объекты физической модели данных (файлы, экстенты, страницы, строки), а также индексы различных типов детально рассмотрены в 12-й главе учебника.

Таблица 4.2

Уровни блокирования объектов базы данных

Блокируемый ресурс	Комментарии
DATABASE	База данных — ресурс блокируется транзакциями, модифицирующими схему базы данных или создающими ее резервные копии
FILE	Один из множества файлов базы данных
TABLE	Таблица, включая все ее строки и все созданные в ней индексы
EXTENT	Группа страниц файла базы данных
PAGE	Одна из страниц файла базы данных, содержащая множество строк таблицы или индекса
RID	Идентификатор строки — используется для блокировки одной строки таблицы
KEY	Ключ индекса — используется транзакциями 3-го уровня изолированности для блокировки диапазонов значений атрибутов, включенных в условия выборки строк таблиц

С другой стороны, наложение высокоуровневых блокировок уменьшает степень параллелизма выполнения конкурирующих транзакций, что в результате снижает производительность системы.

Блокирование объектов низких уровней позволяет гибко управлять конкурирующими транзакциями (например, снимать блокировки строк таблицы, уже обработанных транзакцией, не дожидаясь ее завершения), повышая производительность системы, однако поддержка низкоуровневых блокировок может оказаться недопустимо ресурсоемкой из-за большого их количества.

Менеджер блокировок автоматически выбирает оптимальный уровень блокирования ресурсов, выполняя в необходимых случаях так называемую *эскалацию блокировок* — повышение уровня блокирования путем замены множества низкоуровневых блокировок одной или несколькими блокировками более высокого уровня, или их *деэскалацию*, то есть понижение уровня блокирования объектов. В любом случае критерием оптимальности работы менеджера блокировок является максимум производительности системы при условии сохранения приемлемой ресурсоемкости поддержки блокировок.

Приведем два примера эскалации блокировок.

1. Транзакция заблокировала множество строк таблицы, и при этом все эти строки физически оказались размещенными в одном экстенте. В такой ситуации блокировка множества строк может быть заменена гораздо более экономичной блокировкой всего этого экстента или нескольких его страниц.

2. Транзакции требуется выборка и блокировка строк таблицы для выполнения их обработки, при этом степень селективности предиката выборки составляет 70% от мощности таблицы, содержащей порядка 1 000 000 строк. В такой ситуации менеджер блокировок может принять решение об эскалации блокировки до уровня таблицы, заменив тем самым 700 000 блокировок строк единственной блокировкой всей таблицы.

Примеры выполнения деэскалации блокировок приведены в п. 9.4.2 при рассмотрении специальных режимов блокирования — так называемых *блокировок с намерениями*.

10.4.2. Режимы блокирования

Транзакция запрашивает блокировку объекта, необходимую ей для ограничения доступа к этому объекту других транзакций, в соответствии с типом операции, выполняемой в рамках транзакции, и требуемого уровня ее изолированности. СУБД принимает решение о выборе необходимого *режима блокирования* объекта и, если блокировка в этом режиме не может быть реализована по причине ее несовместимости с ранее установленными блокировками этого объекта, ставит транзакцию в очередь для ожидания освобождения объекта от блокировок.

СУБД поддерживают два основных режима блокирования объектов — ***монопольная блокировка*** и ***совмещаемая блокировка***, а также ряд вспомогательных режимов (***блокировка обновления*** и различные ***блокировки с намерениями***), используемых для повышения эффективности реализации основных режимов блокирования.

Совмещаемую блокировку (**S**hared lock, **S**) объекта может запрашивать транзакция-читатель, и этот запрос будет выполнен при условии, если объект не заблокирован в монопольном режиме. Наличие совмещаемой блокировки объекта *не препятствует* другим транзакциям *читать заблокированный объект* и, соответственно, накладывать на него свои совмещаемые блокировки, что повышает степень параллелизма конкурирующих транзакций и позитивно сказывается на производительности системы.

При этом *транзакции-писатели не получат доступа* к этому объекту до снятия с него совмещаемой блокировки, что позволит решить проблемы «грязного чтения», «неповторяющегося чтения» и «кортежей-фантомов» на соответствующих уровнях изолированности транзакции, по запросу которой была установлена совмещаемая блокировка объекта.

Если для транзакции установлен 1-й уровень изолированности ***READ COMMITTED****, совмещаемые блокировки снимаются сразу после завершения операции чтения, для более высоких уровней изолированности такие блокировки снимаются только после успешного завершения всей транзакции или ее отката*.

Монопольная блокировка (e**X**clusive lock, **X**) решает проблему «последнего изменения» и устанавливается по запросу транзакции-писателя, модифицирующей объект, независимо от уровня изолированности, установленного для этой транзакции. Этот режим блокирования несовместим с любыми режимами — монопольная блокировка не может быть установлена на объект, заблокированный другими транзакциями как в монопольном, так и в совмещаемом режимах, и при этом совмещаемые блокировки не могут быть установлены на монопольно заблокированный объект.

Реализация операций INSERT, UPDATE и DELETE, как правило, требует предварительного чтения данных из таблиц, связанных с модифицируемой таблицей (например, для анализа условий выборки модифицируемых строк), поэтому транзакция-писатель, кроме монопольной блокировки обновляемого объ-

екта, часто запрашивает также и совмещаемые блокировки связанных с ним объектов.

Блокирование объектов в монопольном режиме негативно сказывается на производительности системы как за счет длительного времени ожидания установки самих монопольных блокировок, так и за счет снижения степени параллелизма выполнения конкурирующих транзакций, требующих совмещаемых блокировок объектов, заблокированных в монопольном режиме.

Для повышения эффективности управления монопольными блокировками СУБД используют различные вспомогательные режимы блокирования, обсуждаемые ниже.

Блокировка обновления (**U**pdate, **U**) используется при необходимости обновления объекта и рассматривается как подготовительный этап перед установкой его монопольной блокировки. В отличие от монопольной блокировки, блокировка обновления не конфликтует с совмещаемыми блокировками — она может быть установлена до их снятия и не будет препятствовать завершению установивших их транзакций.

При этом блокировка обновления объекта запретит установку на этот объект любых других блокировок и будет ожидать снятия с него ранее установленных совмещаемых блокировок. Как только последняя из них будет снята, статус блокировки обновления будет повышен до монопольной блокировки, после чего транзакция выполнит необходимые обновления объекта.

Не допускается одновременная установка нескольких блокировок обновления на один объект, что в ряде случаев позволяет предотвратить возникновение некоторых форм взаимоблокировок, рассматриваемых в п. 10.4.3.

Блокировки с намерениями (**I**ntent lock, **I**) позволяют установить блокировку объекта высокого уровня (например, таблицы) с намерением впоследствии провести деэскалацию этой блокировки с понижением уровня блокируемого объекта (например, до уровня нескольких строк этой таблицы).

Блокировки с намерениями повышают степень параллелизма конкурирующих транзакций, а также позволяют значительно снизить затраты (как временные, так и ресурсные) на установку, снятие и проверку конфликтности блокировок. Блокировки с намерениями, так же как и блокировки обновления, могут быть установлены на ранее заблокированные объекты, но, будучи установленными, они препятствуют установке на объект новых блокировок.

Если транзакция запрашивает блокировку объекта низкого уровня (например, множества строк таблицы), менеджер блокировок в первую очередь проверяет наличие соответствующей *блокировки с намерениями* у объекта более высокого уровня (например, таблицы, экстента или файловой страницы): если такая *блокировка с намерениями* установлена и если она конфликтует (см. табл. 4.3) с запрашиваемой блокировкой, запрашиваемая блокировка сразу отклоняется, и только в противном случае запускается существенно более длительная процедура проверки конфликтности блокировок на более низких уровнях блокируемого объекта.

СУБД использует несколько разновидностей блокировок с намерениями, основные из которых — это *совмещаемая*, *монопольная* и *совмещаемая с намерением монопольного блокирования*.

Блокировка с намерением совмещаемого доступа (**I**ntent **S**hared lock, **IS**) устанавливается на всю таблицу в начале транзакции, которая намерена читать отдельные строки этой таблицы соответствующими операциями, и заменяет множество низкоуровневых блокировок строк, устанавливаемых непосредственно перед выполнением операций чтения. Такой подход позволяет избежать длительного ожидания разблокирования низкоуровневых объектов и сокращает время проверки конфликтности и отклонения монопольных блокировок объекта конкурирующими транзакциями.

Блокировка с намерением монопольного доступа (**I**ntent e**X**clusive lock, **IX**) используется для блокирования объектов верхнего уровня, в которых необходимо выполнить большое количество изменений. Например, если в середине длинной транзакции встречается операция, требующая массового изменения строк таблицы, то установка блокировки типа **IX** на всю эту таблицу в начале транзакции позволит сократить время ожидания установки монопольных блокировок на модифицируемые строки, так как менеджер блокировок запретит их блокирование другими транзакциями.

Совмещаемая блокировка с намерением монопольного доступа (**S**hared with **I**ntent e**X**clusive lock, **SIX**) полезна в ситуациях, когда транзакция выполняет чтение большого объема данных объекта и при этом производит изменение лишь небольшой их части. Менеджер блокировок устанавливает монопольную блокировку намерений типа **IX** на экстент или группу страниц, которые содержат данные, требующие модификации, а остальные данные остаются доступными для чтения другими транзакциями, которым разрешено устанавливать совмещаемые блокировки намерений на уровне всего объекта и читать ту часть данных, которая не изменяется транзакцией, установившей блокировку типа **SIX**.

СУБД, принимая решения об установке или отклонении блокировок, запрашиваемых конкурирующими транзакциями, использует информацию о совместимости режимов блокирования (см. табл. 4.3). Если к моменту поступления от транзакции запроса на блокировку объекта этот объект оказался заблокированным другой транзакцией, новая блокировка будет установлена только в том случае, если режим запрашиваемой блокировки совместим с режимом уже существующей блокировки. В противном случае очередная транзакция будет ожидать снятия с объекта несовместимой блокировки.

Как видно из таблицы, наиболее бесконфликтным является режим **IS** — он совместим со всеми режимами блокирования, кроме монопольного (**X**).

Монопольный режим блокирования (**X**) не совместим ни с одним из режимов: пока транзакция удерживает монопольную блокировку объекта, ни одна из других транзакций не может заблокировать этот объект и произвести его чтение или модификацию, что предотвратит возможность проявления проблем «последнего обновления», «чтения грязных данных» и «неповторяющегося чтения».

Таблица 4.3

Совместимость режимов блокирования объектов

Установленный на объект режим блокирования / Запрашиваемый режим блокирования объекта		IS	S	U	IX	SIX	X
С намерением совмещенного доступа	**IS**	Да	Да	Да	Да	Да	Нет
Совмещенный доступ	**S**	Да	Да	Да	Нет	Нет	Нет
Обновление	**U**	Да	Да	Нет	Нет	Нет	Нет
С намерением монопольного доступа	**IX**	Да	Нет	Нет	Да	Нет	Нет
Совмещенный с намерением монопольного доступа	**SIX**	Да	Нет	Нет	Нет	Нет	Нет
Монопольный доступ	**X**	Нет	Нет	Нет	Нет	Нет	Нет

Если транзакция заблокировала объект в совмещенном (**S**) режиме для его чтения, другие транзакции могут бесконфликтно читать этот объект (**S**), а также могут устанавливать на него блокировку обновления (**U**), не дожидаясь завершения первой транзакции.

При этом другие транзакции не смогут установить на объект блокировку в любом из режимов, требующих его модификации (**IX**, **I**, **SIX**), до момента снятия с объекта совмещенной блокировки. Это надежно защитит первую транзакцию от проявления проблемы «чтения грязных данных», а также (при условии, что совмещенная блокировка не будет снята до момента завершения всей транзакции) и от проблем «неповторяющегося чтения» и «кортежей-фантомов».

10.4.3. Тупиковые блокировки — прогнозирование и разрушение

Тупиковыми блокировками (или просто *тупиками* — *deadlock*) называют ситуации, когда две или более конкурирующие транзакции устанавливают такие взаимные блокировки объекта, при которых ни одна из транзакций не может завершиться, пока другая не снимет с объекта своей блокировки (рис. 4.1).

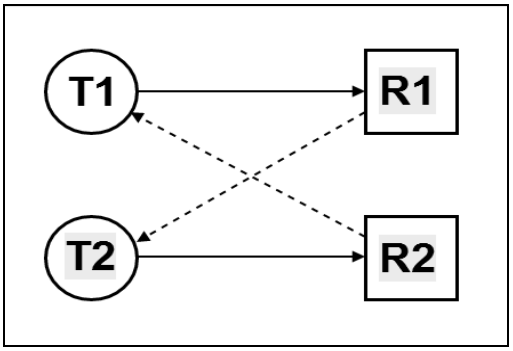


Рис. 4.1
Иллюстрация тупиковой блокировки

Установленные блокировки обозначены на рисунке сплошными линиями — дугами графа, направленными от узлов-транзакций к узлам-объектам, а противоположно направленные (пунктирные) дуги соответствуют блокиров-

кам, запрошенным транзакциями, находящимися в очереди ожидания снятия блокировок с объектов.

Транзакции T1 и T2 содержат операции чтения и модификации объектов R1 и R2, для обеих этих транзакций установлен 2-й уровень изолированности REPEATABLE READ, при котором снятие блокировок производится только в момент полного завершения транзакции (или ее отката).

Транзакции T1 и T2 последовательно запрашивают (и устанавливают) *совмещаемые блокировки* для чтения объектов, соответственно R1 и R2, а затем последовательно запрашивают монопольные блокировки для изменения других объектов, соответственно R2 и R1.

Монопольные блокировки несовместимы с совмещенными (см. табл. 4.3) и не могут быть установлены, пока не будут сняты соответствующие совмещенные блокировки. В результате обе транзакции будут поставлены в очереди ожидания снятия блокировок с заблокированных объектов, но так и не смогут дождаться своей очередности, так как каждая из них препятствует завершению конкурирующей транзакции и, как следствие, снятию соответствующей блокировки.

Для выявления тупиковых блокировок СУБД периодически проводит анализ очередей транзакций, формируемых в системном каталоге базы данных, и в случае обнаружения тупика выполняет его разрушение, жертвуя одной или несколькими транзакциями, участвующими во взаимных блокировках, и выполняя их принудительный откат.

В качестве жертвы, как правило, выбирается самая «дешевая» из транзакций, при этом СУБД ранжирует транзакции с использованием интегральной модели стоимости транзакции, включающей такие параметры, как количество операций и затрагиваемых транзакцией объектов, частота использования транзакции, ее приоритет и др.

Методы распознавания тупиковых блокировок могут быть различными — от простейшего *контроля длительности интервала ожидания снятия блокировок*, предельное значение которого может задаваться специальным параметром LOCK_TIMEOUT, до выполнения моделирующих прогнозов с использованием *алгоритмов редукции графа ожидания транзакций*, иллюстрация одного из которых приведена на рисунке 4.2.

Алгоритм редукции графа ожидания транзакций реализуется циклически следующими последовательными этапами.

Этап 1. Из графа удаляются все дуги, исходящие из тех вершин-транзакций, в которые не входят дуги от вершин объектов — этим моделируется ситуация, в которой все транзакции, не ожидающие снятия блокировок, установленных другими транзакциями, успешно завершились и освободили заблокированные ими объекты.

Этап 2. Направленность дуг, исходящих из тех вершин-объектов, для которых не осталось входящих дуг от вершин-транзакций, изменяется на противоположную — этим моделируется ситуация, в которой все транзакции, ожидавшие снятия блокировок с объектов, установили свои блокировки.

Этап 3. Повторно выполняется 1-й этап алгоритма — и так циклически до тех пор, пока на первом этапе сохраняется возможность удаления дуг, исходящих из вершин-транзакций, в которые не входят дуги от вершин объектов.

Этап 4. Если после завершения алгоритма в графе остаются дуги, значит, имеет место тупиковая блокировка, и следует принимать меры по ее разрушению, выполняя откат одной из транзакций, а затем, уже для новых условий, повторить все этапы алгоритма редукции графа ожидания транзакций.

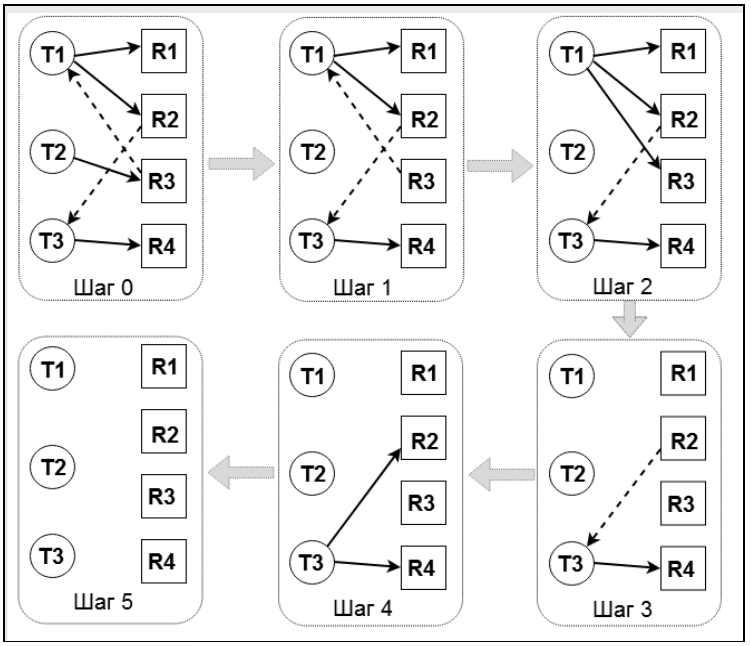


Рис. 4.2
Пример реализации алгоритма редукции графа ожидания транзакций

Исходное состояние графа показано на нулевом шаге алгоритма: транзакция T1 установила блокировки объектов R1 и R2 и ожидает освобождения объекта R3, заблокированного транзакцией T2, которая не претендует на блокировки других объектов. Транзакция T3 установила блокировку объекта R4 и ожидает освобождения объекта R1, заблокированного транзакцией T1.

На первом шаге удаляется дуга T2→R3, на втором — дуга R3→T1 заменяется на противоположную дугу T1→R3, далее удаляются все три дуги, исходящие из вершины-транзакции T1, затем дуга R2→T3 меняет свою направленность и, наконец, удаляются две последние дуги графа, что позволяет сделать вывод об отсутствии тупиковой блокировки.

Попробуем применить этот алгоритм к графу ожидания транзакций, представленному на рисунке 4.1.

На первом этапе алгоритма ни одна из дуг графа не может быть удалена, так как в нем отсутствуют вершины-транзакции, в которые не входят дуги от вершин-объектов — это означает, что ни одна из транзакций не может быть завершена, так как обе они ожидают освобождения заблокированных объектов.

На втором этапе также невозможно переориентировать ни одну из дуг, исходящих из вершин-объектов, так как в графе отсутствуют вершины-объекты, для которых отсутствуют входящие дуги от вершин-транзакций.

Как видим, этот граф ожидания транзакций не поддается редукции, из чего можно сделать вывод о наличии тупиковой блокировки, требующей разрешения путем принудительного отката одной из конкурирующих транзакций. Заметим, что этот вывод можно сделать на основании визуального образа графа, так как его дуги образуют замкнутый цикл.

Иная ситуация представлена на рисунке 4.2 — три транзакции конкурируют в доступе к четырем объектам базы данных.

10.5. SQL-средства управления транзакциями и блокировками

Приведенные ниже синтаксические конструкции и примеры листингов SQL-кода соответствуют требованиям языка Transact SQL, реализованного в MS SQL-Server (начиная с версии 2008).

10.5.1. Уровни изолированности и режимы блокирования

После открытия нового соединения по умолчанию устанавливается определенный уровень изолированности транзакций (для MS SQL-Server это уровень READ COMMITED).

Для установки требуемого уровня изолированности транзакций используется инструкция SET TRANSACTION ISOLATION LEVEL, формат которой иллюстрируется листингом 4.1.

```
SET TRANSACTION ISOLATION LEVEL
  {
    READ UNCOMMITTED
  | READ COMMITTED
  | REPEATABLE READ
  | SNAPSHOT
  | SERIALIZABLE
  }
```

Листинг 4.1
Формат SQL-инструкции, используемой
для установки уровня изолированности транзакций

Единственный параметр этой инструкции — требуемый уровень изолированности (обзор уровней изолированности транзакций, за исключением нестандартного уровня SNAPSHOT, приведен в разделе 9.3).

Замечания:

1) если инструкция SET TRANSACTION ISOLATION LEVEL используется в хранимой процедуре, то при возврате управления будет восстановлен уровень изоляции, действовавший к моменту вызова процедуры;

2) не включенный в стандарт SQL/92 уровень изолированности SNAPSHOT («моментальный снимок») позволяет отказаться от использования традиционных блокировок строк таблиц за счет хранения последних версий измененных строк во временной базе данных;

3) имеется возможность задавать уровни изолированности и режимы блокирования локально для каждого SQL-оператора (SELECT, UPDATE, DELETE, или INSERT), используя для этих целей специальные «подсказки»-*хинты* (hints) — зарезервированные ключевые слова, записываемые в скобках после слова WITH, как это показано в листинге 4.2.

Перечни хинтов и комментарии к их использованию для локального управления уровнями изолированности транзакций и уровнями блокирования объектов БД приведены в таблицах 4.4 и 4.5.

```
SELECT Клиенты.Код_Клиента, Sum(Количество*БазоваяЦенаРеализации)
WITH (REPEATABLEREAD)
FROM (Клиенты INNER JOIN Заказы ON
        Клиенты.Код_Клиента = Заказы.Код_Клиента)
        INNER JOIN Заказано ON Заказы.Код_Заказа = Заказано.Код_Заказа
GROUP BY Клиенты.Код_Клиента;
```

Листинг 4.2
Пример использования *хинтов*

Таблица 4.4

Хинты для управления изолированностью транзакций

Хинт	Условия и результаты использования
READUNCOMMITTED	Устанавливает уровень изолированности READ UNCOMMITTED
READCOMMITTED	Устанавливает уровень изолированности READ COMMITTED
REPEATABLEREAD	Устанавливает уровень изолированности REPEATABLE READ
SERIALIZABLE	Устанавливает уровень изолированности SERIALIZABLE

Таблица 4.5

Хинты для управления уровнями блокирования объектов

Хинт	Условия и результаты использования
ROWLOCK	Устанавливает блокировку на уровне строки таблицы
PAGLOCK	Устанавливает блокировку на уровне файловой страницы
TABLOCK	Устанавливает блокировку на уровне таблицы и удерживает ее только до конца выполнения операции. Если хинт задан в операторе SELECT, другие транзакции могут читать строки таблицы
TABLOCKX	Устанавливает полное блокирование таблицы, запрещающее другим транзакциям чтение данных
HOLDLOCK	Удерживает блокировку до конца транзакции, а не снимает ее после завершения операции
UPDLOCK	Устанавливает блокировку обновления (UPDATE)
NOLOCK	Снимает блокировки на время выполнения операции SELECT
READPAST	При выборке данных оператор SELECT будет пропускать строки, заблокированные другими транзакциями, не ожидая их завершения. Используется при условии, что в соединении установлен уровень изолированности READ COMMITTED

10.5.2. Программирование начала и завершения транзакций

Явные транзакции начинаются с инструкции BEGIN TRANSACTION и заканчиваются инструкциями COMMIT или ROLLBACK. Инструкция SAVE TRANSACTION используется для создания точек сохранения внутри транзакции. Синтаксис этих инструкций поясняется листингами 4.3–4.5.

Начало транзакции

```
BEGIN { TRAN | TRANSACTION }
   [ { transaction_name |
      @tran_name_variable }
    [ WITH MARK [ 'description' ] ] ]
```

Листинг 4.3
Синтаксис инструкции BEGIN TRANSACTION

Замечания:

1) имена транзакций transaction_name или соответствующие переменные @tran_name_variable используются только для внешних транзакций;

2) предложение WITH MARK ['*description*'] позволяет «пометить» транзакцию параметром '*description*' и предписывает сохранять эту пометку в журнале транзакций, что позволит восстанавливать базу данных из резервной копии по журналу транзакций до помеченной транзакции (а не только по дате и времени);

3) если предложение WITH MARK используется без параметра, указание имени транзакции является обязательным — оно будет сохранено в журнале транзакций (вместо отсутствующего параметра '*description*') и может быть использовано при восстановлении базы данных;

4) каждая инструкция BEGIN TRAN производит автоинкремент системной переменной @@TRANCOUNT = @@TRANCOUNT + 1, что позволяет программно контролировать количество активных транзакций (листинг 4.8).

Фиксация транзакции

```
COMMIT [ { TRAN | TRANSACTION }
[ transaction_name | @tran_name_variable ] ]
[ WITH ( DELAYED_DURABILITY = { OFF | ON } ) ]
```

Листинг 4.4
Синтаксис инструкции COMMIT

Замечания:

1) значение параметра DELAYED_DURABILITY = OFF присваивает транзакции статус «устойчивой» и предписывает сообщать об ее успешной фиксации только после того, как соответствующая запись будет сохранена в журнале транзакций;

2) значение параметра DELAYED_DURABILITY = ON присваивает транзакции статус «отложенно-устойчивой» и предписывает сообщать об ее успешной фиксации до того, как соответствующая запись будет сохранена в журнале транзакций;

3) отложенные транзакции получают статус «устойчивы» после сохранения журнала транзакций на диск;

4) каждая инструкция COMMIT производит автодекремент системной переменной @@TRANCOUNT = @@TRANCOUNT – 1 (листинг 4.8).

Точки сохранения

Инструкция SAVE TRANSACTION (листинг 4.5) устанавливает внутри транзакции *точку сохранения* — именованный маркер, к которому можно выполнить частичный откат транзакции инструкцией ROLLBACK TRANSACTION (листинг 4.6). Таких именованных точек внутри транзакции может быть несколько.

```
SAVE { TRAN | TRANSACTION }
{ savepoint_name | @savepoint_variable }
```

Листинг 4.5
Синтаксис инструкции SAVE TRANSACTION

Если произошел откат транзакции к точке сохранения, то выполнение транзакции будет продолжено «вниз» от этой точки до ее фиксации (COMMIT) либо отката (ROLLBACK) к началу транзакции или к одной из точек сохранения.

Откат транзакции

Инструкция ROLLBACK TRANSACTION (листинг 4.6) производит либо полный откат транзакции, либо ее откат до указанной точки сохранения.

```
ROLLBACK { TRAN | TRANSACTION }
   [ transaction_name | @tran_name_variable
     | savepoint_name | @savepoint_variable ]
```

Листинг 4.6
Синтаксис инструкции ROLLBACK TRANSACTION

Замечания:

1) инструкция ROLLBACK TRANSACTION без аргумента *savepoint_name* или *transaction_name* выполняет откат к началу транзакции;

2) при наличии вложенных транзакций такая инструкция выполняет откат всех вложенных транзакций к началу самой внешней транзакции;

3) инструкция ROLLBACK TRANSACTION без аргумента *savepoint_name* уменьшает значение системной переменной @@TRANCOUNT до 0;

4) инструкция ROLLBACK TRANSACTION *savepoint_name* производит частичный откат транзакции до указанной точки сохранения;

5) частичный откат транзакции до точки сохранения не изменяет значения системной переменной @@TRANCOUNT.

10.5.3. Примеры программирования транзакций

Листинг 4.7 иллюстрирует эффект отката именованной транзакции:

– после создания унарной таблицы запускается именованная транзакция;
– в этой транзакции производится вставка трех строк в таблицу;
– выполняется безусловный откат к началу транзакции;
– вне транзакции производится вставка двух строк в эту же таблицу;

– производится безусловная выборка всех строк таблицы, в результате в таблице оказывается только две строки со значениями (4) и (5), вставленными вне транзакции.

```
CREATE TestTran_1 (column_1 int);
BEGIN TRAN TranName
     INSERT INTO TestTran_1 VALUES(1), (2) , (3);
ROLLBACK TRAN TranName;
INSERT INTO TestTran_1 VALUES(4), (5) ;
SELECT column_1 FROM Table_1 ;
```

Листинг 4.7
Пример отката именованной транзакции

```
CREATE TABLE TestTran_2 (a int, b varchar(3));
BEGIN TRANSACTION OuterTran; PRINT @@TRANCOUNT;
    INSERT INTO TestTran_2 VALUES (1, 'aaa');
    BEGIN TRANSACTION InnerTran_1;  PRINT @@TRANCOUNT;
        INSERT INTO TestTran_2 VALUES (2, 'bbb');
        BEGIN TRANSACTION InnerTran_2;  PRINT @@TRANCOUNT;
            INSERT INTO TestTran_2 VALUES (3, 'ccc');
        COMMIT TRANSACTION InnerTran_2;   PRINT @@TRANCOUNT;
    COMMIT TRANSACTION InnerTran_1;  PRINT @@TRANCOUNT;
IF  @@TRANCOUNT=1
    COMMIT TRANSACTION OuterTran;  PRINT @@TRANCOUNT;
ELSE
    ROLLBACK TRANSACTION;
```

Листинг 4.8
Пример фиксации и отката вложенных транзакций
с контролем счетчика открытых транзакций

Листинг 4.8 иллюстрирует фиксацию и откат вложенных транзакций:

– создается бинарная таблица TestTran_2;

– формируется внешняя транзакция OuterTran — системная переменная @@TRANCOUNT получает значение 1;

– формируются две вложенные транзакции: InnerTran_1 первого уровня вложенности и InnerTran_2 второго уровня — системная переменная @@TRANCOUNT последовательно получает значение 2 и затем 3;

– последовательно фиксируются все три транзакции, начиная с InnerTran_2 — системная переменная @@TRANCOUNT последовательно получает значение 2, затем 1 и затем 0;

– если после отката транзакции InnerTran_2 осталась «лишняя» открытая транзакция, производится откат всех транзакций к началу внешней транзакции OuterTran.

Листинг 4.9 иллюстрирует использование транзакций в хранимых процедурах.

```
CREATE PROCEDURE NewPersonalDisconts
     @LastYear int, @LastMonth int
AS
 SET @ID_TMP_Table = OBJECT_ID('ПроданоКлиентам');
 SET TRANSACTION ISOLATION LEVEL REPEATABLE READ ;
 BEGIN TRANSACTION
 IF (@LastYear NOT BETWEEN 2000 AND 2050)
    OR (@LastMonth NOT BETWEEN 1 AND 12)
 BEGIN
   ROLLBACK TRANSACTION
   PRINT('Ошибка входных данных')     RETURN
 END ;
 IF @ ID_TMP_Table IS NOT NULL
   DROP TABLE 'ПроданоКлиентам';
 ELSE
   BEGIN
   SELECT Клиенты.Код_Клиента,
          Sum(Количество*БазоваяЦенаРеализации) AS Сумма
          INTO ПроданоКлиентам
   FROM (Клиенты INNER JOIN Заказы
          ON Клиенты.Код_Клиента = Заказы.Код_Клиента)
          INNER JOIN Заказано
          ON Заказы.Код_Заказа = Заказано.Код_Заказа
  WHERE Year(ДатаИсполнения) = @LastYear
          AND Month(ДатаИсполнения) = @LastMonth
  GROUP BY Клиенты.Код_Клиента;
  UPDATE Клиенты INNER JOIN ПроданоКлиентам
          ON Клиенты.Код_Клиента =
          ПроданоКлиентам.Код_Клиента
          SET Клиенты.ПерсональнаяСкидка =
               0.00001*ПроданоКлиентам.Сумма;
  END;
DROP TABLE 'ПроданоКлиентам';
COMMIT  TRANSACTION  RETURN
```

Листинг 4.9

Пример использования транзакции в хранимой процедуре

Процедура NewPersonalDisconts обновляет значение персональной скидки клиентам торговой компании (рис. 4.2) пропорционально суммарной стоимости заказанных ими товаров в течение месяца.

Процедура содержит одну транзакцию, для которой установлен уровень изолированности REPEATABLE READ, что гарантирует снятие блокировок, установленных этой транзакцией, только по факту ее полного завершения.

Процедура контролирует значения переданных ей входных параметров (расчетные год и месяц), и если параметры оказываются некорректными, производится откат транзакции и завершение работы процедуры.

В противном случае процедура проверяет наличие в базе данных временной таблицы *ПроданоКлиентам*, удаляет ее (при наличии) и создает обновленную версию этой таблицы.

Далее процедура обновляет значение поля *ПерсональнаяСкидка* в таблице *Клиенты* для тех клиентов, которые заказывали товары в расчетном месяце, после чего удаляет временную таблицу и фиксирует транзакцию.

Контрольные вопросы и задания

1. Какие из свойств транзакций обеспечиваются SQL-инструкциями BEGIN TRAN, ROLLBACK TRAN и COMMIT?
2. Какие проблемы, связанные с конфликтами конкурирующих транзакций, решаются на каждом из четырех уровней их изолированности: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ и SERIALIZABLE? Приведите соответствующие примеры.
3. Для чего используется SQL-инструкция SAVE TRAN и в каких ситуациях она может быть полезной?
4. Поясните понятия «режим блокирования» и «уровень блокирования». Перечислите стандартные режимы блокирования объектов. Какие из них и в каких случаях обеспечивают уровень изолированности транзакций REPEATABLE READ?

ГЛАВА 11. ПОДДЕРЖКА ФИЗИЧЕСКОЙ МОДЕЛИ ДАННЫХ

Модель, или структура данных, — одно из основополагающих понятий технологий хранения и обработки информации, хорошо знакомое каждому программисту. В теории и технологии разработки реляционных баз данных рассматриваются три уровня представления модели данных:

– концептуальный уровень — так называемая ER-модель, описывающая объектную структуру предметной области в понятиях «сущность — атрибут — связь»;

– логический уровень, представляющий реляционную схему базы данных в «табличной» терминологии и обеспечивающий SQL-взаимодействие программных приложений с сервером баз данных;

– физический уровень, на котором объекты логической модели данных отображаются в элементы файловой структуры вычислительной системы.

MS SQL-Server поддерживает двухуровневую структуру физической модели данных:

– верхний уровень физической модели обеспечивает взаимосвязь с файловой системой и представлен объектами «файл» и «группа файлов»;

– нижний уровень обеспечивает взаимосвязь с логической моделью данных, определяет внутреннюю структуру файла базы данных и представлен объектами «страница» и «экстент» (группа страниц).

11.1. Файловая модель базы данных

Файловая структура базы данных представляет верхний уровень физической модели данных, на котором сервер баз данных обеспечивает взаимодействие с файловой системой. Структура базы данных MS SQL-Server представлена объектами двух категорий: ***файлы*** и ***группы файлов***.

11.1.1. Файлы и группы файлов

MS SQL-Server поддерживает файлы двух типов:

– единственный (в базе данных) ***файл журнала транзакций*** (log-file), в котором сервер сохраняет информацию обо всех активных операциях доступа к базе данных, то есть о тех операциях, которые изменили ее состояние (например, об операциях **Update**, **Delete** или **Insert**). Имена файлов этого типа имеют стандартное расширение **.ldf** (**l**og **d**ata **f**ile);

– множество ***файлов данных*** (data-files), в которых хранятся экземпляры всех логических объектов базы данных — пользовательских и системных таблиц, хранимых представлений, процедур, функций, ограничений целостности данных и т. д., а также служебных структур данных, например индексов. Имена файлов этого типа могут иметь одно из двух стандартных расширений: **.mdf** (**m**aster **d**ata **f**ile) или **.ndf** (sec**o**ndary **d**ata **f**ile).

Если в базе данных всего один файл типа data, он имеет статус ***первичного*** (***primary file*** — .mdf). Администратор может создавать и другие файлы этого

типа — все они будут иметь статус вторичных файлов (***secondary file*** **—** .ndf), при этом в базе данных может быть только один первичный файл.

Все данные системного каталога *базы данных* хранятся в первичном файле, а пользовательские данные могут храниться как в первичном, так и во вторичных файлах.

Файл типа data — это, по существу, сегмент дискового пространства определенного размера, зарезервированный сервером баз данных у файловой системы. Начальный размер файла либо явно указывается в момент создания базы данных SQL-инструкцией **CREATE DATABASE**, как это показано в листинге 1.1, либо определяется по умолчанию в соответствии с параметрами системной базы данных **Model**.

При этом предполагается, что файл имеет объем свободного пространства, достаточный для функционирования базы данных в течение некоторого периода времени, а когда свободного места в файле не остается, сервер запрашивает у файловой системы дополнительный сегмент дискового пространства и файл увеличивается в размере.

Каждый файл базы данных характеризуется рядом параметров, значения которых хранятся в системной таблице **SysFiles** этой базы данных:

- **File_ID** — внутренний идентификатор файла, уникальный в пределах базы данных; автоматически присваивается сервером файлу при его создании, используется как элемент адресной ссылки на страницу данных и/или на строку таблицы; выполняет роль первичного ключа таблицы **`SysFiles`**;
- **File_name** — имя файла в файловой системе, задается в стандартном формате (том:\путь\имя.расширение);
- **Name —** логическое имя файла (синоним `File_name`);
- **Size** — текущий размер файла (объем дисковой памяти, зарезервированной сервером баз данных у файловой системы), может автоматически изменяться в процессе эксплуатации базы данных;
- **MaxSize** — максимально допустимый размер файла, может иметь значение `unlimited`, в этом случае размер файла ограничивается ресурсами файловой системы;
- **Growth increment** — шаг приращения размера файла — дополнительная «порция» дискового пространства, запрашиваемая сервером у файловой системы, когда текущий размер файла становится недостаточным для хранения данных, может задаваться в абсолютных размерных единицах или в процентах от текущего размера файла;
- **Group_ID** — внутренний идентификатор файловой группы, к которой принадлежит этот файл, выполняет роль внешнего ключа, обеспечивающего связь (M:1) между системными таблицами **SysFiles** и **SysFileGroups**.

Группы файлов

Для удобства администрирования и повышения эффективности хранения данных файлы типа **data** могут быть (логически!) распределены по ***файловым группам***. В каждой базе данных по умолчанию создается единственная файло-

вая группа, имеющая статус ***первичной*** (*primary file group*), при этом администратор может создавать дополнительные файловые группы, каждая из которых получает статус ***вторичной*** (*secondary file group*). Формируются вторичные файловые группы с помощью ключевого слова **FILEGROUP** в SQL-инструкциях **CREATE DATABASE** и **ALTER DATABASE**.

Первичный файл всегда принадлежит первичной файловой группе, вторичный файл может принадлежать как первичной, так любой из вторичных файловых групп. Файл не может одновременно входить в состав нескольких файловых групп.

Примечание 1. Файловая группа является логической надстройкой над файловой структурой базы данных: принадлежность файла определенной файловой группе не накладывает никаких ограничений на его физическое размещение в дисковом пространстве вычислительной системы.

Примечание 2. Файл журнала транзакций (.ldf) *не принадлежит ни одной из файловых групп базы данных.*

Каждая файловая группа характеризуется рядом параметров, значения которых хранятся в системной таблице **SysFileGroups** базы данных:

- **Group_ID** — внутренний идентификатор файловой группы, уникальный в пределах БД, автоматически присваивается сервером при создании группы, выполняет роль первичного ключа системной таблицы **SysFileGroups**;
- **Group_name** — имя файловой группы, используется в операторах **Create Table** и **Create Index** для явного указания файловой группы, в файлах которой должны сохраняться данные таблицы или индекса. Первичная файловая группа всегда имеет имя `PRIMARY`;
- **Default** — присваивает группе статус «*группа по умолчанию*»:

– из множества групп только одна группа может иметь такой статус;

– если этот параметр явно не указан ни для одной из групп, группой по умолчанию будет назначена первичная группа;

– при создании нового файла в базе данных он будет ассоциирован с группой по умолчанию, если иное явно не указано;

– при создании новой таблицы в базе данных она будет ассоциирована с группой по умолчанию, если иное явно не указано, и все данные этой таблицы будут физически размещены в файлах этой группы;

- **Read Only** — присваивает группе статус «*только для чтения*»; если таблица базы данных ассоциирована с такой группой, для этой таблицы будет заблокирована возможность модификации данных.

Приведенный ниже листинг 4.10 иллюстрирует языковые средства (TransactSQL) управления файловой структурой баз данных и содержит пакет из 4 последовательных SQL-инструкций.

Инструкция **CREATE DATABASE** создает пользовательскую базу данных **MyDB**, содержащую первичный файл данных в первичной файловой группе, пользовательскую файловую группу **FG1**, содержащую два вторичных файла, и файл журнала (вне файловых групп). Инструкция не устанавливает свойство

DEFAULT ни одной из файловых групп, в результате чего статус «группа по умолчанию» получает первичная файловая группа.

Инструкция **CREATE TABLE** создает в базе данных таблицу **MyTable**, явно ассоциированную с пользовательской файловой группой **FG1** (а не с первичной группой, имеющей к этому моменту статус «группа по умолчанию»).

Инструкция **ALTER DATABASE** модифицирует структуру базы данных **MyDB** и придает файловой группе **FG1** статус «группа по умолчанию».

Последняя инструкция **CREATE TABLE** создает в базе данных еще одну таблицу **MyTable1**, также (неявно) ассоциированную с пользовательской файловой группой **FG1**, имеющей к этому моменту статус «группа по умолчанию».

```
USE master;
CREATE DATABASE MyDB
 ON PRIMARY
    (NAME='Primary_F',FILENAME='C:\data\Prm.mdf',
     SIZE=4MB, MAXSIZE=10MB,FILEGROWTH=1MB),
  FILEGROUP FG1
    (NAME = 'Dat1',FILENAME = 'D:\data\FG1_1.ndf',
     SIZE = 1MB, MAXSIZE=5MB, FILEGROWTH=1MB),
    (NAME = 'Dat2',FILENAME = 'E:\data\FG1_2.ndf',
     SIZE = 15MB, MAXSIZE=50MB, FILEGROWTH=1MB)
 LOG ON (NAME='MyDB_log',FILENAME = 'C:\data\MyDB.ldf',
     SIZE=1MB,MAXSIZE=10MB, FILEGROWTH=1MB);
USE MyDB;
CREATE TABLE MyTable(ID int PRIMARY KEY, Data char(8))
 ON FG1;
ALTER DATABASE MyDB MODIFY FILEGROUP FG1 DEFAULT;
CREATE TABLE MyTable1 (ID int, Name char(16));
```

Листинг 4.10
Создание и модификация файловой структуры базы данных

11.1.2. Файловые страницы и экстенты

Основным объектом файловой модели базы данных является файл типа DATA, ассоциированный с некоторой файловой группой. База данных может содержать несколько таких файлов, главное назначение которых — хранение экземпляров объектов логической модели данных (например, строк таблиц) для их последующего извлечения в соответствии с поступившими серверу SQL-запросами.

При создании новой таблицы она будет связана с одной из файловых групп: или явной ссылкой на имя файловой группы в SQL-инструкции **CREATE TABLE**, или неявно — с группой, имеющей в этот момент статус «группа по умолчанию». В любом случае при вставке строк в таблицу серверу баз данных потребуется решать задачу поиска свободного пространства в файлах, включенных в соответствующую файловую группу, и задачу эффективного распре-

деления строк таблицы между этими файлами с сохранением соответствующей адресной информации, необходимой для последующего поиска строк таблицы.

Листинг 4.11 иллюстрирует задачу поиска свободного пространства в файлах базы данных и задачу поиска строк заданной таблицы, удовлетворяющих определенному условию.

```
USE Master
CREATE DATABASE MyDB
 ON PRIMARY (NAME='MyDB_Primary',FILENAME='C:\Prm.mdf'),
   FILEGROUP FG1
   (NAME = 'MyDB_Dat1',FILENAME='D:\F1.ndf',
    SIZE = 2MB, MAXSIZE=5MB, FILEGROWTH=1MB),
   (NAME = 'Dat2',FILENAME='E:\F2.ndf',
    SIZE = 15MB, MAXSIZE=50MB, FILEGROWTH=5MB);
USE MyDB
CREATE TABLE MyTable (Key_0 INT IDENTITY,Key_1 INT,Key_2 INT,
 FullName CHAR(60))
 ON FG1;
DECLARE @key0 INT, @key1 INT, @key2 INT, @name CHAR(60)
SET @key1=1000*RAND(),@key2=1000*RAND()
SET @name=STR(@key1)+STR(@key2)
INSERT INTO MyTable values(@key1,@key2,@name)
Go 10000
SELECT key_1, FullName INTO NewTable
FROM MyTable WHERE Key_2>500;
```

Листинг 4.11
Пример выборки и вставки строк в таблицы

Пакет содержит четыре последовательных SQL-инструкции.

Инструкция **CREATE DATABASE** создает пользовательскую базу данных, содержащую три файла: один файл в первичной файловой группе и два файла — во вторичной группе FG1. Инструкция не содержит явного указания на группу со статусом DEFAULT, следовательно, группой по умолчанию будет считаться первичная файловая группа.

Инструкция **CREATE TABLE** создает таблицу MyTable, явно ассоциированную со вторичной группой FG1.

Следующая инструкция **INSERT INTO** вставляет в таблицу MyTable 10 000 строк, заполняя их случайными данными.

Последняя инструкция **SELECT … INTO** производит выборку из таблицы MyTable тех ее строк, которые удовлетворяют заданному условию, и вставку выбранных строк в новую таблицу NewTable.

Вставка строк в таблицу MyTable потребует поиска свободного пространства в двух вторичных файлах группы FG1, а строки таблицы NewTable будут

записаны в единственный первичный файл первичной файловой группы, так как именно эта группа имеет статус DEFAULT.

Очевидно, что для реализации алгоритмов поиска свободного пространства в файле и алгоритмов поиска строк таблиц, сохраненных в файле, необходимо определить информационную структуру самого файла и предусмотреть наличие служебных структур данных, обеспечивающих хранение адресной информации.

Файловые страницы

Базовым элементом файла типа DATA является ***файловая страница***. Сервер баз данных представляет файл типа DATA как линейный список из страниц фиксированного размера (по 8 Кб), которые последовательно (позиционно) пронумерованы, начиная с нулевой страницы[6].

Таким образом, номер страницы **PageNum** однозначно определяет ее позицию внутри файла: смещение (в килобайтах) от начала файла до начала страницы вычисляется как **8*PageNum**.

Учитывая тот факт, что файлы базы данных тоже пронумерованы и каждый из них при регистрации получил свой уникальный идентификатор FileID, сохраненный в системной таблице SysFiles, агрегат SysFiles.PageNum однозначно определяет адрес страницы в файловой структуре базы данных.

Страница является минимальным объектом физической модели данных, который может быть выделен сервером логическому объекту, при этом один логический объект (например, таблица) может быть владельцем одной или нескольких страниц одного или нескольких файлов, а одна страница не может иметь более чем одного владельца.

Из сказанного, в частности, следует, что номера страниц, владельцем которых является таблица базы данных, могут использоваться в качестве адресных ссылок, используемых для поиска и выборки всех строк этой таблицы.

Экстенты

Если страница является «единицей занятости» файла данными логических объектов, то для хранения информации о свободном пространстве файлов используется другая, более крупная единица, называемая ***экстентом***. Экстент имеет фиксированный размер 64 Кб и представляет собой блок из восьми соседних страниц одного файла.

Экстенты, так же как и страницы, последовательно пронумерованы в пределах файла, начиная с нулевого экстента. При этом номер экстента **ExtNum** и номера входящих в него страниц **PageNum** связаны простой арифметической зависимостью. Например, страница 18 принадлежит экстенту № 2, а экстент № 5 включает страницы с 40-й по 47-ю.

Сервер баз данных может присвоить экстенту статус ***однородного*** (uniform) или ***смешанного*** (mixed, shared): смешанный экстент может совмест-

[6] На уровне файловой системы файл может быть фрагментирован, однако на уровне файловой модели этот факт игнорируется, и с точки зрения сервера баз данных файл — это неразрывная цепочка последовательно расположенных и позиционно пронумерованных страниц.

но использоваться разными владельцами страниц, а владельцем всех восьми страниц однородного экстента является какой-либо один логический объект базы данных (рис. 4.3).

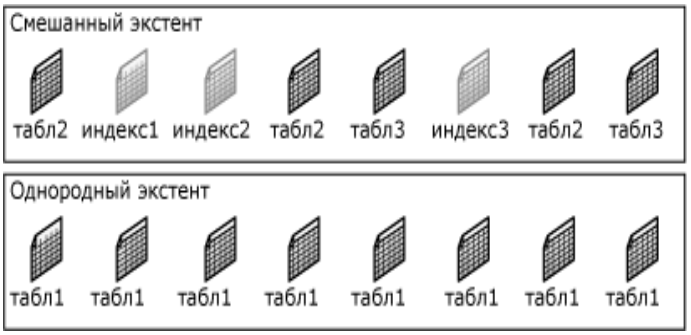


Рис. 4.3
Схема размещения файловых страниц в смешанных и однородных экстентах

Очевидно, что для небольших таблиц эффективнее оперировать смешанными экстентами, иначе некоторые страницы однородных экстентов всегда будут оставаться незанятыми. С другой стороны, что также очевидно, наличие однородных экстентов ускоряет работу алгоритмов поиска данных в таблицах большого объема (от 64 Кб), так как в этом случае адрес (порядковый номер) какой-либо одной страницы, принадлежащей логическому объекту, фактически определяет и адреса еще семи страниц этого же объекта.

Совсем не очевидна логика сервера баз данных, принимающего решение о выделении логическому объекту экстентов того или иного типа — исследованию этого вопроса посвящено одно из заданий практикума по администрированию (п. 14.2.2).

Типы файловых страниц

Перечень типов страниц, поддерживаемых MS SQL-Server, приведен в таблице 4.6.

Страницы типа **Data/Index** — основной тип файловых страниц, предназначенных для хранения экземпляров логических объектов — строк таблиц базы данных, а также строк индексных таблиц, подобных по своей структуре классическим реляционным таблицам. Упрощенная схема хранения данных в странице типа Data/Index иллюстрируется рисунком 4.4.

Все пространство страницы (8 Кб) разделено на три области:

– *заголовок страницы* — начальная область страницы, имеет фиксированный размер 96 байтов, содержит служебную информацию, специфическую для страниц различных типов;

– *тело страницы* — основная область страницы (максимум 8060 байт), содержит множество контейнеров («слотов»), каждый из которых предназначен для хранения одной строки таблицы;

– *хвостовик страницы* — конечная область страницы переменной длины (минимум 36 байт), содержит массив указателей на слоты тела страницы (т. е., по существу, на строки таблицы).

Таблица 4.6

Типы файловых страниц

Тип страницы	Тип хранимой информации
Data	Данные логических объектов, кроме данных типов LOB (LargeOBjects)
Index	Данные индексов
Text/Image	• Данные «длинных» типов, экземпляры которых превышают размер файловой страницы: **text**, **ntext**, **image**, **xml**, **nvarchar(max)**, **varchar(max)**, **varbinary(max)**. • Данные типов переменной длины **varchar()**, **nvarchar()**, **varbinary()**, **sql_variant** в условиях, когда размер строки на основной странице типа Data превышает размер 8060 байт
IAM	Index Allocation Map — битовая карта размещения страниц логического объекта по экстентам файла
GAM, SGAM	Global Allocation Map, Shared Global Allocation Map — глобальные битовые карты, содержащие информацию о свободных экстентах, их типах и степени заполнения
PFS	Page Free Space — информация о степени заполнения страниц
BCM	Bulk Changed Map (карта массовых изменений) — глобальная битовая карта, содержащая информацию об экстентах, измененных с момента последнего выполнения операции резервного копирования журнала транзакций
DCM	Differential Changed Map (карта разностных изменений) — глобальная битовая карта, содержащая информацию об экстентах, измененных с момента последнего выполнения операции резервного копирования базы данных

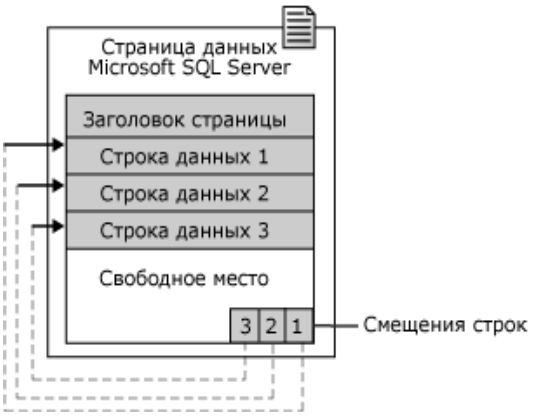


Рис. 4.4
Структура страницы типа DATA/INDEX

Согласно канонам реляционной модели данных, все кортежи одного отношения (т. е. все строки одной таблицы) имеют одинаковую арность (количество атрибутов/столбцов), при этом соответствующие атрибуты кортежей содержат данные одного типа. Именно в этом смысле верно утверждение о «прямоугольности» реляционных таблиц: действительно, все строки таблицы имеют (логически) одинаковую длину, равную сумме длин типов данных всех ее столбцов.

Например, все строки таблицы MyTable (листинг 4.3) имеют одинаковую длину, равную 72 байтам (3*4 + 60), и эта таблица действительно является «пря-

моугольной» как на логическом уровне, так и на уровне физической модели данных, так как для описания всех ее атрибутов были использованы типы данных постоянной длины, в том числе и строковый тип CHAR(60) для атрибута FullName.

Скорее всего, использование типа CHAR(60) в данном случае было ошибочным, так как независимо от реальной длины текста атрибут FullName будет занимать ровно 60 байт в каждой строке таблицы. Было бы эффективнее (с точки зрения экономии памяти) использовать вместо CHAR(60) тип данных переменной длины VARCHAR(60), тогда затраты памяти на хранение экземпляров атрибута FullName будут минимальными, но таблица при этом потеряет свою «прямоугольность», что явно усложнит систему хранения адресов ее строк в файловых страницах типа DATA.

Приведенная на рисунке 4.4 структура файловой страницы обеспечивает эффективную адресацию строк переменной длины.

Массив указателей на слоты, расположенный в хвостовике страницы, содержит целые числа, трактуемые как смещения от начала страницы до начала соответствующего слота. Первый элемент массива указателей всегда содержит число 96 — это указатель на первый слот, начинающийся по смещению 96 байт от начала страницы (сразу после ее заголовка). Когда страница пуста, все остальные указатели в этом массиве отсутствуют.

При записи на страницу первой строки (длина которой, естественно, известна) она помещается в первый слот, а в массив указателей заносится ссылка на начало второго слота (равное сумме длины записанной строки и числа 96).

При записи второй и всех последующих строк сканируется (справа налево) массив указателей, определяется номер очередного еще не заполненного слота, этот слот заполняется, а в массив указателей заносится ссылка на начало следующего слота. Процесс заполнения страницы может продолжаться до тех пор, пока обе эти структуры (массив слотов и массив указателей на них) не «встретятся» — в этот момент страница получит статус заполненной на 100%.

Если задан номер слота SlotNum, в котором размещена искомая строка таблицы, то адрес этого слота легко определяется прямым доступом к соответствующему элементу массива указателей.

Таким образом, адрес строки таблицы (RID — Row IDentifier) — это агрегат SysFiles.PageNum, используемый для адресации страницы в файловой структуре базы данных, дополненный номером соответствующего слота страницы: RID = SysFiles.PageNum.SlotNum. Именно такие RID-адреса используются в индексах для обеспечения прямого доступа к строкам таблиц, содержащих искомые значения ключевых полей таблицы.

Страницы типа **Text/Image** используются для хранения «больших объектов» (LOB — Large Object) — значений столбцов таблиц, превышающих объем файловой страницы (длинных текстов, графических объектов с хорошим разрешением и пр.). Если, например, столбец таблицы имеет один из таких типов данных и при этом в какой-либо строке таблицы экземпляр данных этого типа имеет действительно большой объем, для хранения этого экземпляра выделяется отдельная страница типа **Text/Image**, а в соответствующем слоте основной страницы

типа **Data** сохраняется указатель на эту страницу. Если для хранения экземпляра оказывается недостаточно одной страницы типа **Text/Image**, сервер дополнительно выделяет необходимое количество таких страниц, связывая их в линейный список специальными указателями (NextPage и PrevPage в заголовках страниц).

Страницы типа Text/Image используются также для хранения данных типов переменной длины в условиях, когда размер строки на основной странице типа **Data** превышает максимально допустимый размер 8060 байт. Если в результате вставки или обновления данных в таблице размер строки выходит за указанный предел, происходит перемещение данных столбца на страницу типа **Text/Image** с сохранением указателя на эту страницу в соответствующем слоте основной страницы типа **Data**. Если в дальнейшем размер строки уменьшается, данные перемещаются обратно на исходную страницу типа **Data**.

Страницы типа IAM (Index Allocation Map) формируются для каждого логического объекта базы данных (таблицы или индекса) в момент вставки в этот объект первого экземпляра данных. **IAM**-страница содержит информацию о номерах всех экстентов, содержащих страницы логического объекта, и представляет собой битовую карту размером около 8 Кб, в которой номер позиции каждого бита ассоциируется с номером экстента: если **i**-й экстент файла содержит хотя бы одну страницу, владельцем которой является логический объект, то в **IAM**-странице этого объекта **IAM[i] = 1**.

Нетрудно подсчитать, что одной **IAM**-страницы будет достаточно для представления таблицы объемом около 4 Гб. Если таблица имеет больший размер, она получит дополнительные **IAM**-страницы, связанные в линейный список указателями NextPage и PrevPage в заголовках этих страниц.

Если известен номер первой **IAM**-страницы, владельцем которой является таблица базы данных, легко определить номера всех используемых таблицей экстентов и получить доступ к страницам и строкам этой таблицы — именно такой подход и реализуется в низкоуровневом методе TableScan(), используемом для выборки строк из некластеризованных таблиц.

Страницы типов GAM (Global Allocation Map) и **SGAM** (Shared Global Allocation Map) содержат глобальные битовые карты (объемом около 8000 байт), в которых каждый бит несет определенную информацию о соответствующем экстенте файла.

Битовая карта **GAM** содержит информацию о свободных или занятых экстентах: если **GAM[i]** = 1, то **i**-й экстент свободен, в противном случае хотя бы одна страница этого экстента занята.

Битовая карта **SGAM** содержит информацию о типах экстентов и степени их заполнения: если **SGAM[i]** = 1, то **i**-й экстент используется как смешанный и при этом имеет хотя бы одну свободную страницу; в противном случае этот экстент либо является однородным, либо смешанным, но полностью занятым.

Таким образом, двухбитовый код **i**-го экстента (**GAM[i]**,**SGAM[i]**) несет информацию о его типе и степени заполнения (табл. 4.7), что позволяет серверу реализовывать несложные алгоритмы поиска свободного пространства при вставке строк в таблицы базы данных:

– если **GAM[i]** = 1, то **i**-й экстент свободен, его тип еще не определен и значение бита **SGAM[i]** может быть любым;

– если серверу требуется свободный однородный экстент (например, для массового заполнения строк большой таблицы), производится поиск **GAM[i]** = 1 и после заполнения этого экстента для него устанавливается **GAM[i]** = 0 и **SGAM[i]** = 0;

– для поиска смешанного экстента со свободными страницами сканируются обе битовые карты, и выбирается экстент с кодом (0;1). После 100%-ного заполнения выбранного экстента для него устанавливается **SGAM[i]** = 0;

– при отсутствии смешанного экстента со свободными страницами выбирается свободный экстент (**GAM[i]** = 1), после его частичного заполнения устанавливаются **GAM[i]** = 0 и **SGAM[i]** = 1. В результате этот экстент получит статус смешанного экстента, имеющего свободные страницы;

– при освобождении **i**-го экстента (например, в результате массового удаления строк соответствующих таблиц) для него устанавливается **GAM[i]** = 0, и он получает статус свободного экстента.

Пара страниц **GAM/SGAM** описывает файл размером около 4 Гб. Для файлов большего размера формируются дополнительные пары **GAM/SGAM**-страниц, связанные в линейные списки указателями NextPage и PrevPage в заголовках соответствующих страниц.

Таблица 4.7

Кодирование состояний экстентов

Состояние i-го экстента	GAM [i]	SGAM [i]
Свободен, в текущий момент не используется	1	0
Однородный или заполненный смешанный	0	0
Смешанный со свободными страницами	0	1

Страницы типа PFS (Page Free Space) предназначены для хранения информации о степени заполнения страниц файла базы данных. Тело PFS-страницы состоит из единственного слота размером 8092 байта, содержимое которого трактуется как числовой массив байтового типа, каждый элемент которого **PFS[i]** представляет 8-битовый код **i**-й страницы файла. Младшие 4 бита этого кода определяют степень заполнения соответствующей страницы (табл. 4.8), а старшие биты кода несут дополнительную информацию о странице (например, о ее типе и принадлежности смешанному или однородному экстенту).

Таблица 4.8

Кодирование степени заполнения страниц

Состояние i-й страницы	PFS[i]
Страница не занята	0
Страница заполнена от 1 до 50%	1
Страница заполнена от 51 до 80%	2
Страница заполнена от 81 до 95%	3
Страница заполнена от 96 до 100%	4

Одна **PFS**-страница описывает файл размером чуть меньше 64 Мб. Для файлов большего размера выделяется необходимое количество дополнительных **PFS**-страниц, образующих линейный список с помощью указателей NextPage и PrevPage в заголовках этих страниц.

На рисунке 4.5 представлена типовая структура нулевого экстента файла: нулевая страница — это заголовок файла, первая из **PFS**-страниц файла всегда имеет порядковый номер «1», затем расположены другие служебные страницы, и далее — все остальные страницы файла.

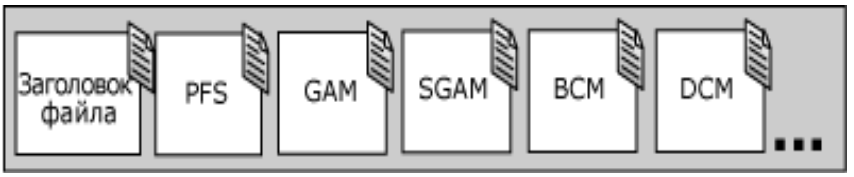


Рис. 4.5
Стандартное расположение служебных страниц в начальной области файла

11.2. Средства управления физической моделью данных

Информация о принадлежности файловых страниц логическим объектам базы данных и об адресах этих страниц содержится в системной таблице SysIndexes, входящей в состав системного каталога базы данных. Каждая строка этой таблицы представляет один логический объект (таблицу или индекс), поля **First**, **Root** и **FirstIAM** содержат идентификаторы («адреса») соответственно первой страницы типа **Data** (для объектов-таблиц), корневой индексной страницы (для объектов-индексов) и первой **IAM**-страницы (для любых логических объектов).

Идентификатор страницы **ID_Page** хранится в формате **BINARY(6)** и представляет собой агрегат из двух чисел, в котором младшие два байта представляют идентификатор файла (**ID_File**), а старшие четыре байта — порядковый номер страницы (**PageNum**) в этом файле: **ID_Page = ID_File.PageNum**.

Хранимая процедура **sp_spaceused** принимает имя таблицы и возвращает пять ее параметров: количество строк (rows), общий объем (в килобайтах) зарезервированного дискового пространства (reserved), в том числе занятого страницами данных (data), индексными (включая IAM) страницами (index_size) и неиспользуемыми (unused) страницами, расположенными в однородных частично заполненных экстентах.

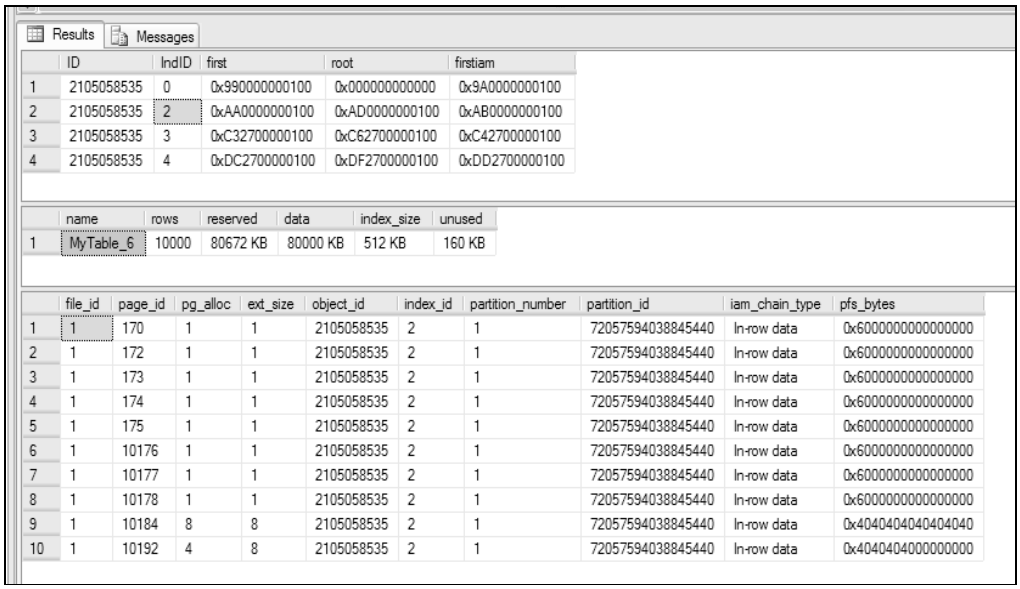
Команда **EXTENTINFO** системной утилиты DBCC позволяет получить более детальную информацию о страницах, занятых данными таблицы и всеми ее индексами: номера страниц, степень их заполнения и количество страниц в экстентах.

Команда DBCC **PAGE** позволяет получить информацию о содержимом страницы и отображает данные ее заголовка, содержимое всех слотов и массив указателей на строки таблицы (рис. 4.4).

Листинг 4.12 содержит пакет SQL-инструкций, иллюстрирующих использование средств анализа структуры файла базы данных, а на рисунке 4.6 приведен результат выполнения первых трех инструкций этого пакета.

```
USE MyDB_2
SELECT ID,IndID,root,first,firstiam
FROM sysindexes WHERE ID=Object_ID('MyTable_6')
GO
EXEC sp_spaceused MyTable_6
GO
DBCC EXTENTINFO(MyDB_2,MyTable_6,-1)
GO
DBCC TRACEON (3604)
DBCC PAGE ('MyDB_2'1,1,2)
GO
```

Листинг 4.12
Пример использования средств анализа структуры файла базы данных



Results | Messages

	ID	IndID	first	root	firstiam
1	2105058535	0	0x990000000100	0x000000000000	0x9A0000000100
2	2105058535	2	0xAA0000000100	0xAD0000000100	0xAB0000000100
3	2105058535	3	0xC32700000100	0xC62700000100	0xC42700000100
4	2105058535	4	0xDC2700000100	0xDF2700000100	0xDD2700000100

	name	rows	reserved	data	index_size	unused
1	MyTable_6	10000	80672 KB	80000 KB	512 KB	160 KB

	file_id	page_id	pg_alloc	ext_size	object_id	index_id	partition_number	partition_id	iam_chain_type	pfs_bytes
1	1	170	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
2	1	172	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
3	1	173	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
4	1	174	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
5	1	175	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
6	1	10176	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
7	1	10177	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
8	1	10178	1	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
9	1	10184	8	8	2105058535	2	1	72057594038845440	In-row data	0x4040404040404040
10	1	10192	4	8	2105058535	2	1	72057594038845440	In-row data	0x4040404000000000

Рис. 4.6
Результат выполнения листинга 4.12

11.3. Алгоритм доступа к неупорядоченным данным

Вспомним, что реляционная модель данных не гарантирует упорядоченности расположения строк в таблице, и если пользователю необходимо представить выборку данных, отсортированную по значениям ее столбцов, то программист должен явно указать на это в разделе Order By соответствующего SQL-запроса.

На уровне физической модели данных строки реляционной таблицы хранятся в файловых страницах типа DATA, объединенных (по умолчанию) в структуры типа куча (heap), при этом файловые страницы, принадлежащие одной таблице, могут располагаться в произвольных местах файла (или нескольких файлов) базы данных.

Информация о принадлежности групп файловых страниц (экстентов) определенной таблице хранится в специальной IAM-странице, также принадлежащей таблице, а номер IAM-страницы хранится в системном каталоге базы данных (рис. 4.7).

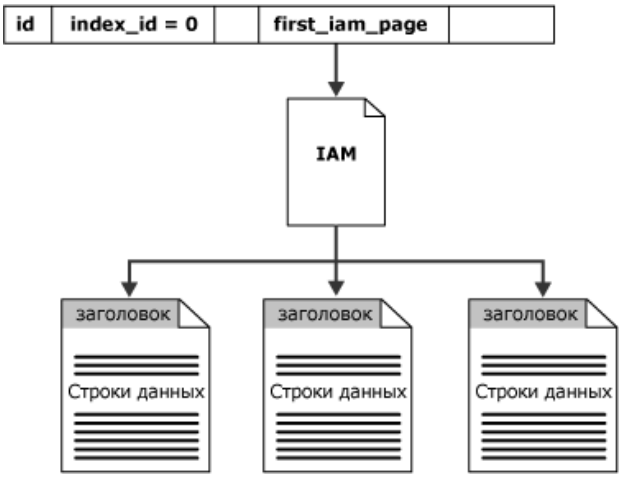


Рис. 4.7
Упрощенная схема доступа к данным типа куча
методом последовательного сканирования

В условиях отсутствия дополнительной информации адресного характера единственным методом поиска в куче строк таблицы, удовлетворяющих заданному критерию отбора (диапазону значений указанных в запросе столбцов), является метод последовательного сканирования кучи (TableScan), работающий по следующему алгоритму:

– чтение системного каталога, определение номера IAM-страницы;

– загрузка IAM-страницы, определение номеров экстентов, включающих файловые страницы, принадлежащие таблице;

– последовательная загрузка каждой страницы выбранных экстентов в оперативную память;

– выполнение циклической процедуры сравнения значений столбцов с критерием отбора в каждой загруженной странице;

– формирование результирующей выборки.

Стоимость такого метода весьма высока, она пропорциональна количеству занятых таблицей страниц и не зависит от степени селективности предиката выборки. Например, единственная строка таблицы, удовлетворяющая критерию поиска, может оказаться как в первой, так и в последней странице, но в любом случае придется просканировать все страницы, принадлежащие таблице.

ГЛАВА 12. УПРАВЛЕНИЕ ИНДЕКСАМИ

12.1. Линейный индекс

Естественным способом снижения стоимости выборки данных из кучи является отказ от технологии последовательного сканирования и применение методов прямого доступа, предусматривающих наличие специальных указателей (адресных ссылок) на строки таблиц, соответствующие критерию выборки. Применение таких методов требует наличия служебных структур данных (индексов), формируемых для основных таблиц базы данных.

Простейшей индексной структурой является линейный индекс, представляющий собой классический двунаправленный список, реализованный в виде бинарной таблицы, один из столбцов которой содержит все значения индексируемого поля (называемого «ключом индекса»), а второй — указатели на соответствующие строки индексируемой таблицы.

Отметим основные особенности устройства индексов:

1) если актуальна задача ускорения поиска по нескольким полям таблицы (ключам), то для одной таблицы может потребоваться несколько индексов — по одному для каждого поискового ключа;

2) мощность (количество строк) индексной таблицы соизмерима с мощностью основной (индексируемой) таблицы базы данных, а в предельном случае обе эти таблицы имеют одинаковую мощность;

3) физический размер индексной таблицы существенно (возможно, в сотни раз) меньше размера основной индексируемой таблицы, соответственно индекс занимает существенно меньше файловых страниц по сравнению с индексируемой таблицей;

4) строки индексной таблицы отсортированы по значению ключа индекса, что позволяет существенно ускорить поиск указателя по значению ключа непосредственно в самой индексной странице, применяя, например, метод дихотомии;

5) если индексируемое поле таблицы не является уникальным, одинаковые его значения могут многократно встречаться в различных строках таблицы и, как следствие, могут оказаться в нескольких файловых страницах. В этом случае поле ключа индекса в индексной таблице будет содержать множества значений-дубликатов, причем пара значений «ключ — ссылка» будет оставаться уникальной в пределах всей индексной таблицы;

6) индексные страницы не хранятся в структуре типа кучи — все страницы одного индекса организованы (логически) в линейный список (в порядке возрастания или убывания значения ключа индекса), реализуемый с помощью специальных указателей, присутствующих в заголовках всех индексных страниц;

7) операции вставки/удаления строк таблицы, а также операции модификации индексированных столбцов требуют оперативной перестройки соответствующих индексов этой таблицы, что может негативно сказаться на показателях производительности.

Ниже приведено пошаговое описание алгоритма поиска строк таблицы по значению линейно-индексированного столбца в структуре данных типа куча. Этот алгоритм актуален как для случая индексирования таблицы по уникальному ключу, так и для случая, когда значение индексируемого столбца многократно дублируется в разных строках таблицы.

– Последовательная загрузка всех индексных страниц (начиная с первой и далее до последней по указателям на следующий элемент списка), в каждой из которых:

• выполняется циклическая процедура сравнения значений ключей индекса с критерием отбора; при их совпадении формируется массив указателей на страницы и строки таблицы;

– последовательная загрузка страниц кучи, номера которых попали в массив указателей, в каждой из которых:

• выбираются строки таблицы в соответствии с массивом указателей;

• формируется результирующая выборка.

Пример

Размер файловой страницы — 8 kb (2^{13} b)

Индексируемая таблица базы данных:

мощность — около 16 млн (~2^{24}) строк;

средняя длина одной строки — 2 k;

длина индексируемого поля — 8 b;

всего строк таблицы в одной странице — 4;

всего страниц, занятых таблицей, — около 4 млн ($2^{24}/2^2$).

Индексная таблица:

мощность — около 16 млн (~2^{24}) строк;

длина строки — 16 b (ключ индекса 8 b + указатель 8 b);

строк индекса в одной индексной странице — 512 ($2^{13}/2^4$);

всего индексных страниц — порядка 32 тыс. ($2^{24}/2^9$).

Сравним два рассмотренных выше метода поиска данных в куче по критерию стоимости реализации простого SQL-запроса:

```
Select * From T1 Where T1.Col = const,
```

где T1 — таблица из рассмотренного выше примера; const — произвольное числовое значение.

В модели стоимости будем учитывать только операции доступа к внешней памяти для загрузки файловых страниц.

Метод полного прямого сканирования дает оценку стоимости в четыре миллиона единиц, независимо от степени селективности запроса Sel, равной количеству строк таблицы, в которых T1.Col = const.

Метод доступа с использованием линейного индекса потребует последовательной загрузки некоторого количества $\mathbf{K_I}$ индексных страниц до тех пор, пока значение ключа индекса не выйдет за пределы диапазона поиска, и дополнительной загрузки тех $\mathbf{K_T}$ страниц индексированной таблицы T1, ссылки на которые были найдены (Ind.Col = const) при обработке индексных страниц. Количество $\mathbf{K_T}$(Sel) таких «дополнительных» страниц зависит от степени се-

лективности запроса Sel, а общая оценка стоимости для этого метода составит $K_I + K_T(Sel)$.

Пессимистическая оценка количества загружаемых индексных страниц $K_I = 32\,768$ (когда совпадение будет найдено только в последней из них), оптимистическая оценка $K_I = 1$, а усредненная оценка $K_I \cong 16\,000$.

Пессимистическая оценка количества загружаемых страниц индексируемой таблицы $K_T \cong 4\,000\,000$, что соответствует ситуации, когда в таблице **T1** достаточно много (больше, чем страниц в куче) одинаковых и равных **const** значений столбца **Col**, и при этом они равномерно распределены по всем файловым страницам этой таблицы.

Ясно, что вероятность второго пессимистичного случая крайне мала, а вероятность одновременного проявления двух рассмотренных выше пессимистичных ситуаций равна нулю (что в целом должно нам добавить оптимизма в оценке эффективности линейных индексов).

Завершая краткий обзор линейных индексных структур, следует отметить, что линейные индексы не нашли практического применения в системах управления реляционными базами данных по причине своей низкой эффективности.

В рассмотренном примере усредненная оценка количества операций чтения индексных страниц, необходимого для выборки данных из таблицы мощностью в 16 млн строк, составила 16 тыс. единиц. Это, конечно, существенно лучше, чем оценка в 4 млн единиц для метода полного сканирования, но все-таки весьма посредственно по сравнению с многоуровневыми индексными структурами, построенными на базе сбалансированных деревьев.

12.2. Многоуровневый иерархический индекс

Основная причина низкой эффективности линейного индекса заключается в его «линейности». Обеспечив возможность прямого (адресного) доступа к файловым страницам кучи, сам индекс остался классической структурой последовательного доступа к данным — линейным списком, на котором, как известно, заданы два основных «поисковых» метода: переход к следующему и переход к предыдущему элементам списка.

В отличие от линейных индексов, иерархические (древовидные) структуры данных обеспечивают прямой доступ не только к страницам кучи, но и к самим индексным страницам, организованным на каждом уровне индекса в последовательную списковую структуру.

Порядком индекса (**P**) будем называть количество строк индекса, помещающихся в одной индексной странице. Для нашего примера $P = 512$ (2^9), и при этом весь линейный индекс занимает **32 768** (2^{15}) файловых страниц.

Рассмотрим процесс построения многоуровневого иерархического индекса на базе существующего линейного индекса, который теперь будем считать конечным («листовым») уровнем многоуровневого индекса.

Все индексные страницы и все строки внутри индексных страниц упорядочены по значению ключа индекса, и каждая индексная строка листового уровня

содержит указатель на соответствующую строку таблицы базы данных, включающий номер файловой страницы из кучи и номер строки этой страницы.

Построим еще один уровень иерархического индекса — линейный индекс, страницы которого будут содержать указатели на индексные страницы листового уровня. Потребуются **64** индексных страницы для этого «предлистового» уровня (**$2^{15}/2^9$**), так как каждая индексная страница может содержать **P = 512** (**2^9**) ссылок на страницы листового уровня, а всего на листовом уровне **32 768** (**2^{15}**) страниц.

Продолжим построение многоуровневого индекса — создадим еще один уровень, страницы которого будут содержать указатели на индексные страницы предлистового уровня. Здесь нам будет достаточно одной индексной страницы (с восьмикратным запасом, так как указателей надо всего **64**, а порядок индекса **P = 512**). Эта страница называется «корневой страницей» многоуровневого индекса или, более кратко, «корнем дерева».

В результате построен многоуровневый индекс, страницы которого на каждом уровне связаны в двунаправленные списки, а переходы от страниц верхних (родительских) уровней на страницы нижних (дочерних) уровней выполняются по прямым ссылкам в соответствии со значениями ключей индекса. Построенный индекс содержит **32 833** страницы (**1 + 64 + 32768**), распределенные по трем уровням индекса.

Количество уровней индекса (**H**) называют «глубиной индекса», которая зависит от количества страниц кучи **N** и порядка индекса **P**: **$H = \mathrm{Log}_P(N) + 1$**. Для нашего примера **$H = \mathrm{Log}_{512}(32768) + 1 \cong 2{,}665 \rightarrow 3$**.

Уровни индекса принято последовательно нумеровать в порядке возрастания, начиная от корневого (**Level = 0**) и заканчивая листовым (**Level = H – 1**) уровнем. На корневом (нулевом) уровне индекса всегда находится единственная индексная страница **N[0] = 1**, а количество страниц **N[i]** на каждом дочернем уровне зависит от количества значений ключа индекса на всех страницах родительского уровня и, в частности, от степени заполнения индексных страниц. Максимально возможное количество страниц на **i**-м[7] уровне индекса (при 100%-ном заполнении всех индексных страниц) **$N[i] = P^i$**.

Ниже приведено описание алгоритма реализации метода поиска строки в куче по значению индексированного столбца, включающего «спуск» от корня индекса до листового уровня с последующей загрузкой страниц кучи, указатели на которые были получены на листовом уровне индекса (рис. 4.8):

– получение адреса корневой страницы индекса запросом к системному каталогу базы данных (*адрес страницы* = file_ID.page_Num);

– загрузка корневой индексной страницы, поиск индексной строки, удовлетворяющей критерию отбора по ключу индекса, определение адреса индексной страницы нижележащего уровня индекса;

– загрузка индексной страницы промежуточного уровня индекса;

[7] В системе MS SQL-Server принята иная нумерация уровней индекса: нулевым считается листовой уровень, а корневой уровень индекса имеет наибольший номер.

– поиск индексной строки, удовлетворяющей критерию отбора по ключу индекса, определение адреса индексной страницы нижележащего уровня;

– и т. д. по всем промежуточным уровням индекса;

– загрузка индексной строки *листового* уровня:

• выполнение циклической процедуры сравнения значений ключей индекса с критерием отбора и формирование массива указателей на строки таблицы в куче (*указатель на строку* = file_ID.page_Num.slot_Num);

• последовательная загрузка в оперативную память страниц кучи, номера которых попали в массив указателей;

• в каждой из загруженных страниц: выборка строк таблицы в соответствии с массивом указателей;

• формирование результирующей выборки.

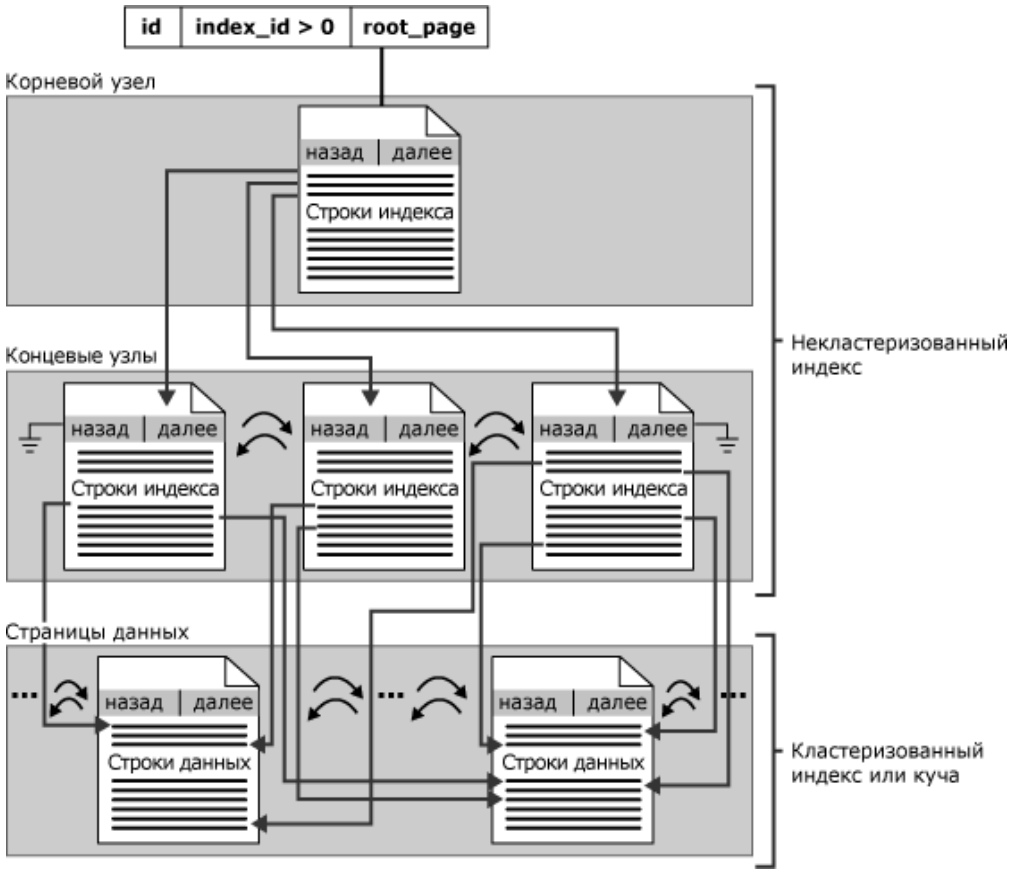


Рис. 4.8

Схема доступа к данным типа куча методом спуска по многоуровневому индексу

Как видно из рисунка 4.8 и приведенного выше описания алгоритма «спуска по дереву», для получения ссылки на страницу кучи, содержащую уникальное значение искомого ключа, потребуется загрузка одной индексной страницы на каждом уровне индекса (в нашем примере **H = 3**) и дополнительно за-

грузка одной страницы кучи. Общая оценка стоимости запроса в этом случае составит 4 единицы (сравните с оценкой 16 000 единиц для линейного индекса).

Если многоуровневый индекс построен по неуникальному столбцу таблицы, его преимущество по сравнению с линейным индексом будет не настолько радикальным: потребуется загрузка одной индексной страницы на каждом нелистовом уровне индекса (в нашем примере 2 страницы) и дополнительно загрузка K_I страниц листового уровня индекса, количество которых зависит от степени селективности запроса **Sel** и порядка индекса **P**: $K_I = Sel/P$.

Если, например, **P = 512** (как в нашем примере), а степень селективности предиката выборки по индексируемому ключу составляет 2%, то при мощности таблицы в 16 млн строк получим значение оценки стоимости $K_I = 320000/512 \cong 640$, что существенно меньше 16 000 единиц для линейного индекса.

12.3. Кластеризованный уникальный индекс

Рассмотренный выше многоуровневый индекс — автономный объект базы данных, существующий отдельно от структуры кучи, на базе которой он был создан. Такие индексы (в терминологии MS SQL-Server) принято называть «некластеризованными» (nonclustered), подчеркивая тот факт, что данные индексированной таблицы и индексы, связанные с этой таблицей, хранятся отдельно друг от друга, то есть в разных «кластерах».

Некластеризованные индексы существенно ускоряют поиск данных по уникальным ключам, но при высокой степени селективности предиката выборки их эффективность резко падает.

Некластеризованные индексы хорошо работают в запросах с точечными условиями выборки (Select … Where Col = Const) и гораздо хуже, если условие выборки содержит диапазон значений ключевого поля (Select…Where Col Between Const1 And Const2).

Некластеризованные индексы недостаточно эффективны при выполнении запросов, требующих соединения таблиц или группировки их строк, реализация которых основана на алгоритмах сортировки и слияния данных.

Кластеризованный (clustered) индекс — это структура данных, объединяющая в единый «кластер» и строки индексируемой таблицы, и собственно индекс. В кластеризованном индексе файловые страницы, содержащие строки индексируемой таблицы, перестают быть кучей — они упорядочиваются по значению ключа (как внутри страниц, так и между страницами) и в таком виде занимают место листового уровня многоуровневого индекса, страницы которого образуют двунаправленный список, обеспечивающий возможность быстрого перехода на последующую или предыдущую страницу. Остальные (верхние) уровни кластеризованного индекса устроены так же, как и в некластеризованных индексах (рис. 4.9).

Так как данные таблиц в кластеризованном индексе хранятся в уже упорядоченном виде, такие индексы оказываются эффективными в запросах с диапазонными условиями выборки, а также при реализации методов, требующих сортировки данных.

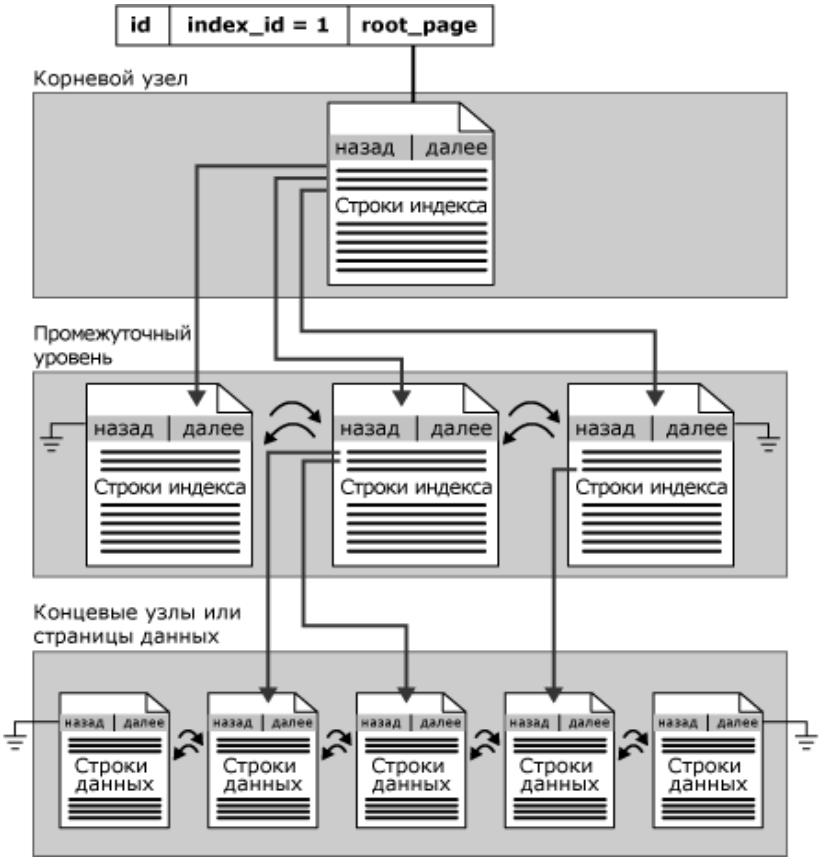


Рис. 4.9
Схема доступа к данным
типа кластеризованный многоуровневый индекс

Физическая упорядоченность строк таблицы в кластеризованном индексе накладывает существенное ограничение на его использование: в одной таблице может быть создано не более одного такого индекса. Как правило, кластеризованный индекс создается по одному из уникальных столбцов таблицы, а при наличии в таблице первичного ключа сервер автоматически (по умолчанию) создает по нему кластеризованный индекс.

Если кластеризованный индекс создается до заполнения таблицы, то при последующей вставке строк они сразу упорядочиваются по значению ключа индекса, формируя листовой уровень индекса как двусвязный список файловых страниц.

Если кластеризованный индекс создается на базе уже заполненной таблицы, сервер автоматически перестраивает кучу в список листового уровня индекса и затем достраивает все остальные его уровни, вплоть до корневого.

Если к этому моменту в таблице уже существовали некластеризованные индексы, то все они будут автоматически перестроены, так как в этом случае изменяется формат указателя листового уровня некластеризованного индекса:

вместо ссылки на строку таблицы (номер слота файловой страницы) этот указатель теперь будет содержать значение ключа кластеризованного индекса, соответствующего этой строке (которое однозначно определяет номер слота).

Такой подход несущественно увеличивает стоимость поиска по значению ключа индекса, так как требует дополнительного «спуска» по кластеризованному индексу, однако он дает существенную экономию при реализации процедур реиндексации — перестройки индексов таблицы после модификации ее данных в условиях, когда в таблице, кроме кластеризованного индекса, создано большое количество некластеризованных индексов.

Если в таблице модифицируется значение неиндексированного столбца и это приводит к перераспределению данных между страницами кластеризованного индекса, то это никак не скажется на некластеризованных индексах, так как в них отсутствуют указатели на физическое размещение страниц. Если же в аналогичной ситуации изменение коснулось одного из индексированных столбцов, то это потребует перестройки только этого индекса.

12.4. Фактор заполнения индексных станиц

Как было показано выше, порядок индекса **P**, определяющий количество индексных строк на одной индексной странице, влияет на глубину индекса $\mathbf{H = Log_P(N) + 1}$ и, следовательно, на стоимость алгоритма спуска от корневого до листового уровня индекса.

Если оптимизировать индексные структуры по критерию производительности выполнения операций поиска данных, следует стремиться к увеличению порядка индекса **P** как за счет минимизации длины индексных записей (ключ + ссылка), так и за счет максимального использования пространства индексных страниц, то есть 100%-ного заполнения этих страниц индексными записями.

Однако, 100%-ное заполнение индексных страниц приведет к резкому снижению производительности выполнения операций модификации данных, так как вставка и удаление строк таблицы потребуют оперативной перестройки всех ее индексов, связанной с массовым проведением весьма дорогостоящих операций расщепления и слияния индексных страниц на всех уровнях индексов.

В этих условиях администратор базы данных, учитывая специфику структуры базы данных, характер типовых запросов и результаты мониторинга работы пользователей, может принять компромиссное решение и установить требуемое значение *фактора заполнения* индексных страниц в процентах от значения порядка индекса **P** (например, 20% при интенсивном выполнении операций модификации данных и 80% — в противном случае).

Сервер баз данных будет учитывать установленное значение фактора заполнения при начальном формировании индексов, то есть, фактически, будет занижать допустимый порядок индекса, что приведет к увеличению глубины индекса и, как следствие, к снижению производительности выполнения операций выборки данных.

В процессе эксплуатации базы данных реальное значение фактора заполнения будет изменяться (вспомним про расщепление и слияние страниц при вставке и удалении строк таблицы), однако при каждом выполнении *операции реиндексации* сервер автоматически перестроит индексы в соответствии с установленным начальным значением фактора заполнения.

По умолчанию сервер устанавливает фактор заполнения индексных страниц на уровне 50%.

12.5. Рекомендации по использованию индексов

Прежде, чем принимать решение об индексировании таблиц, следует провести детальный анализ и попытаться понять, индексирование каких их столбцов позволит повысить производительность работы базы данных. Для этого рекомендуется проанализировать активность пользователей, собрать и обработать статистику выполняемых запросов к базе данных, обращая внимание на интенсивность выполнения поисковых и модифицирующих запросов, наборы соединяемых в запросах таблиц, критерии поиска данных в таблицах, условия группировки и сортировки данных и т. д.

Кластеризованные индексы

– По умолчанию сервер автоматически создаст кластеризованный индекс для первичного ключа таблицы, и с таким его решением следует согласиться в подавляющем большинстве случаев;

– если создается подчиненная таблица, будет полезно создать кластеризованный индекс для составного ключа, включающего пару «первичный ключ — внешний ключ», так как в процессе эксплуатации базы данных, как правило, часто будут выполняться запросы с эквисоединением таблиц, требующие сортировки данных;

– если планируется создать единственный индекс в таблице, пусть он будет кластеризованным;

– если не планируется создавать в таблице первичный ключ (и такое тоже случается), следует создать кластеризованный индекс по столбцу, который часто используется в условиях группировки и операторах сравнения **between, >, >=, <=** или **<** ;

– не рекомендуется создавать кластеризованные индексы для столбцов с «длинными» типами данных: во-первых, это приведет к уменьшению порядка самого этого индекса и, как следствие, к увеличению его глубины, а во-вторых, что гораздо важнее, этот «длинный тип» будет многократно дублироваться в качестве конечной ссылки на страницах листовых уровней всех некластеризованных индексов этой таблицы, что также негативно скажется на их структуре.

Некластеризованные индексы

– Некластеризованные индексы целесообразно создавать для тех столбцов таблицы, которые участвуют в SQL-запросах в разделах **Where**, **Having**, **Group BY**, а также в условиях соединения таблиц **Join**;

– не следует применять такие индексы для часто изменяемых столбцов таблиц: ускорение поиска данных станет незаметным на фоне существенных временных потерь на реиндексацию после каждого такого изменения;

– не следует также применять некластеризованные индексы для таблиц малой мощности: при небольшом количестве файловых страниц метод полного сканирования кучи может оказаться более эффективным по сравнению с методами поиска по ключу индекса;

– наибольший эффект от применения некластеризованного индекса достигается, когда индексируемый столбец содержит относительно много различных (уникальных) значений, а также при небольшой степени селективности предиката выборки (т. е. когда количество строк, соответствующих критерию отбора по ключу индекса, значительно меньше общего количества строк в таблице).

Серверы баз данных поддерживают различные специальные типы индексов, например *композитные индексы* или *индексы с включаемыми неиндексируемыми столбцами*, позволяющие повысить производительность выполнения запросов определенных типов. Структуры некоторых из таких индексов предстоит исследовать при выполнении заданий практикума по администрированию (глава 14).

ГЛАВА 13. ОПТИМИЗАЦИЯ ПРОЦЕДУРНЫХ ПЛАНОВ ИСПОЛНЕНИЯ SQL-ЗАПРОСОВ

Повышение производительности информационных систем (ИС), активно использующих операции доступа к данным, — комплексная задача, решение которой не ограничивается оптимизацией исключительно базы данных и может затрагивать различные аспекты функционирования ИС — организационные, технические, архитектурные, программные, эксплуатационные.

Важнейшей эксплуатационной характеристикой ИС, влияющей на ее производительность, является *время отклика на запрос к базе данных*, которое во многом зависит от реализации физической модели данных и, в частности, от правильности построения индексных структур данных.

- *«Почему запрос выполняется так долго?»*
- *«Почему длительности выполнения двух практически одинаковых запросов так сильно отличаются?»*
- *«Почему сегодня этот запрос выполняется гораздо дольше, чем он выполнялся вчера?»*
- *«Почему все так плохо даже при наличии индексов?»*

Для получения ответов на такие вопросы, часто задаваемые администратору пользователями и программистами баз данных, необходимо рассмотреть процесс формирования *процедурного плана выполнения SQL-запроса*, генерируемого сервером баз данных в результате трансляции соответствующего SQL-кода.

13.1. SQL — язык программирования декларативного типа

Программируя на классическом языке высокого уровня, таком, например, как Basic, Pascal или C#, мы, по существу, описываем на этом языке алгоритм решения некоторой задачи в виде последовательности операторов, то есть явно определяем процедуру обработки данных, реализация которой должна привести к получению желаемого результата. Такие высокоуровневые языки относятся к категории *процедурных* языков программирования.

Процедурными являются и низкоуровневые (машинные) языки, так как машинная программа — это детальное описание алгоритма обработки данных в виде последовательности команд процессора. Из этого, в частности, следует, что компиляция исходного кода программы, написанной на процедурном языке высокого уровня, в ее машинный код — это (всего лишь!) преобразование одного описания процедуры обработки данных в другое описание этой же процедуры.

В отличие от процедурных языков программирования, язык SQL является языком *декларативного* типа, и текст SQL-запроса к базе данных не содержит описания алгоритма получения результата, а только *декларирует требования* к этому *результату*.

Например, SQL-запрос вида Select * From T Where T.x>T.y требует выборки из таблицы **T** только тех ее строк, в которых значение поля **x** больше значения поля **y**, не описывая при этом алгоритма выполнения такой операции.

13.2. Типовая схема трансляции SQL-запроса

Если SQL-запрос не содержит описания процедуры поиска данных, а в результате компиляции SQL-кода все же получается процедурный машинный код, то, очевидно, задачу выбора необходимой процедуры транслятор решает самостоятельно, без прямого участия программиста.

В состав транслятора с языка SQL входит специализированный компонент — *оптимизатор запросов*, задача которого — сгенерировать *оптимальный процедурный план выполнения запроса,* то есть, фактически, «переписать» исходный непроцедурный SQL-код в эквивалентный ему код на некотором промежуточном процедурном языке. На завершающем этапе трансляции этот процедурный план будет скомпилирован в исполнимый машинный код.

Процесс трансляции SQL-запроса (рис. 4.10) включает несколько последовательных фаз его обработки. Перед тем как попасть на вход оптимизатора, исходный SQL-код подвергается предварительной обработке (на рисунке не показано), включающей его синтаксический анализ, лексическое и логическое преобразования.

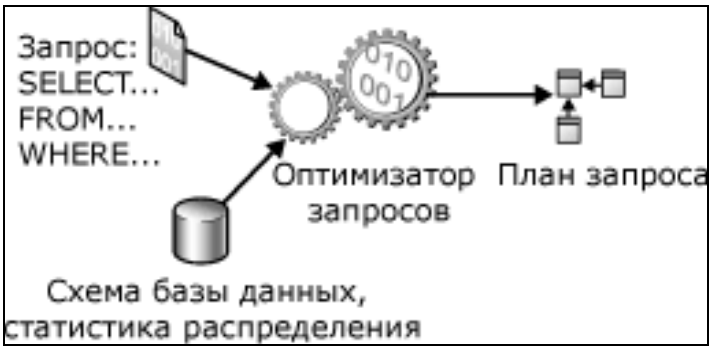


Рис. 4.10
Упрощенная схема трансляции SQL-запроса

Фаза 1. Синтаксический анализ

Стандартная для любых трансляторов фаза — синтаксический разбор (parsing) исходного кода. Синтаксический анализатор просматривает инструкцию SELECT, разбивает ее на логические единицы (ключевые слова, выражения, операторы и идентификаторы), контролирует правильность написания языковых конструкций.

Фаза 2. Лексические преобразования

На этой фазе проверяется корректность использования имен логических объектов (таблиц, индексов, полей таблиц), а сами эти имена заменяются на со-

ответствующие им внутренние идентификаторы (вспомним системный каталог базы данных и, в частности, таблицы SysObjects, SysColumns, SysIndexes).

В результате формируется новое (все еще непроцедурное) представление запроса, синтаксически эквивалентное исходному SQL-коду.

Фаза 3. Логические преобразования

Дальнейшая обработка запроса связана с его логическими преобразованиями с целью упрощения процесса дальнейшего анализа и принятия решений, связанных с генерацией процедурных планов. При этом применяются как ***синтаксические***, так и ***семантические*** преобразования.

Синтаксические преобразования запроса выполняет специализированный компонент транслятора — *algebrizer* (*алгебраизатор*), в котором входное представление запроса приводится к виду, удобному для его последующей трансформации в последовательность операций реляционной алгебры.

В результате синтаксических преобразований:

– производится упрощение предикатов ограничения выборки в предикаты соединений (логические выражения разделов Where и Join) путем приведения их к некоторой канонической форме;

– исключается вложенность запросов, например запрос вида

Select R1.a From R1 Where R1.b IN
(Select R2.d From R2 Where R2.e = R1.c)

приводится к виду

Select R1.a From R1 Join R2 ON (R1.b = R2.d AND R1.c = R2.e);

– запросы, заданные на представлениях, преобразуются путем объединения кода запроса с кодом представления, например пусть создано представление **V(C)** :

Create View V(C) AS Select T.C1 From T Where T.C1>6

и на базе этого представления написан следующий запрос:

Select * From V Where V.C<6

План выполнения такого запроса (без предварительного логического преобразования) состоял бы из двух фаз:

1) «материализация» представления **V** путем выборки из таблицы **T** строк, соответствующих ограничению **T.C1>6**, с сохранением результатов во временной таблице, например в таблице **Tmp.V**;

2) выборка из временной таблицы **Tmp.V** тех строк, которые удовлетворяют ограничению **Tmp.V.C<6**.

После объединения кодов запроса и представления получится запрос вида **Select T.C1 From T Where T.C1>6 AND T.C1<6**, в котором раздел **Where** содержит тождественно ложное логическое выражение, из чего явно следует, что строить и, тем более, оптимизировать процедурный план вообще не потребуется — достаточно возвратить в результат запроса пустое множество строк.

В процессе ***семантических преобразований*** формируется новый запрос, синтаксически не эквивалентный исходному запросу, но дающий точно такой же результат.

Пусть, например, в базе данных, обслуживающей систему кадрового учета компании, имеются две связанные таблицы: таблица **Employees**, содержащая данные обо всех сотрудниках компании, в том числе размер должностного оклада сотрудника (поле **Salary**), и таблица **Posts** — справочник наименований должностей компании (поле **Title**).

Следующий исходный SQL-запрос производит выборку всех сотрудников компании, занимающих должности начальников («**Head»**):

```
Select Employees.Name
From Employees Inner Join Posts
ON Employees.Post_ID = Posts.Post_ID
Where Posts.Title Like «Head»;
```

План реализации такого запроса будет включать процедуру внутреннего соединения двух таблиц, например методом вложенных циклов, и процедуру фильтрации строк временной таблицы, сформированной первой процедурой. Заметим, что процедура соединения таблиц — одна из наиболее дорогостоящих процедур, даже при наличии соответствующих индексов.

Пусть для поля **Salary** таблицы **Employees** задано следующее ограничение целостности CONSTRAINT, соответствующее утверждению, что зарплата начальника не может быть меньше $1000:

```
ALTER TABLE Employees
ADD CONSTRAINT MaxHeadSalary
CHECK(
  If (Select Posts.Title WHERE Posts.Post_ID = Employees.Post_ID) Like «Head»
  Then Employees.Salary> = $1000);
```

С помощью такого ограничения можно, например, контролировать правильность заполнения данных о заработной плате сотрудников.

В этих условиях исходный запрос может быть семантически преобразован в другой запрос, синтаксически отличающийся от исходного, но эквивалентный ему по результату выполнения, и при этом гораздо более простой и «удобный» для оптимизатора, генерирующего процедурный план:

```
Select Employees.Name From Employees
Where Employees.Salary>=$1000;
```

К тому же план исполнения такого запроса не содержит процедуры соединения таблиц, следовательно, он будет существенно менее дорогостоящим по сравнению с процедурным планом выполнения исходного запроса.

Четвертая и пятая фазы трансляции SQL-запроса реализуются непосредственно ***оптимизатором запросов***, который получает непроцедурное представление запроса, прошедшее предварительную обработку на предшествующих фазах трансляции, и генерирует процедурный план выполнения запроса. В своей работе оптимизатор использует дополнительную статистическую информацию о текущем состоянии базы данных.

Фаза 4. Генерация альтернативных планов выполнения запроса

Генератор процедурных планов получает внутреннее непроцедурное представление запроса (результат его предшествующих преобразований) и запрашивает дополнительную информацию о текущем состоянии объектов базы данных, указанных в запросе: например, данные о мощности таблиц, наличии и типах индексов, созданных в этой таблице по столбцам, затрагиваемым запросом.

В распоряжении генератора имеется набор типовых стратегий реализации запросов по существу, набор правил, применяемых в определенных условиях. Генератор анализирует информацию о текущем состоянии объектов базы данных и принимает решение о возможности использования определенных стратегий при формировании планов.

Результатом данной фазы трансляции запроса является множество альтернативных планов, каждый из которых представлен в виде соответствующего ***дерева логических операторов***, описывающего на концептуальном уровне последовательность выполнения операций реляционной алгебры (вот где оказывается полезной проведенная ранее алгебраизация непроцедурного представления запроса).

Примеры логических операторов:

– **Cross Join** — соединяет каждую строку из первого (верхнего) входного параметра с каждой строкой второго (нижнего) входного параметра;

– **Distinct** — удаляет дубликаты из набора строк;

– **Lazy Spool** — сохраняет все строки входных данных в скрытом временном объекте, который хранится в системной базе данных **`TempDB`**.

Фаза 5. Оценка стоимости и выбор оптимального плана

На этой фазе оптимизатор решает задачу эффективной реализации логических операторов, описывающих альтернативные планы выполнения запроса. Для каждого логического оператора выбирается подходящий физический оператор (или несколько физических операторов) и определяется «стоимость» их выполнения. В результате альтернативный план представляется ***деревом физических операторов*** и вычисляется его суммарная стоимость.

Каждый физический оператор является объектом или процедурой, выполняющей соответствующую логическую операцию, например сканирование таблицы или индекса, соединение таблиц, вычисление, статистическую обработку, проверку целостности данных и др.

Примеры физических операторов:

– **Filter** — просматривает входные данные и возвращает только строки, удовлетворяющие критерию фильтрации;

– **Nested Loops** — выполняет логические операции внутреннего соединения методом вложенных циклов;

– **Clustered Index Delete** — удаляет строки из кластеризованного индекса;

– **Index Scan** — получает все записи некластеризованного индекса, которые удовлетворяют условию, указанному в предикате.

Некоторые низкоуровневые операторы (например, **Index Scan**) являются и логическими, и физическими. С полным перечнем операторов, используемых для представления процедурных планов, можно ознакомиться на соответствующем официальном ресурсе корпорации Microsoft.

На рисунке 4.11 приведен пример графического представления процедурного плана выполнения следующего SQL-запроса:

```
Select * From MyTable_4 Where Key0 = 4 AND Key1>50
```

Таблица MyTable_4 содержит около 10 000 строк, не имеет кластеризованного индекса, а по полям Key0 и Key1 этой таблицы созданы некластеризованные неуникальные индексы. Результирующая выборка составила 11 строк.

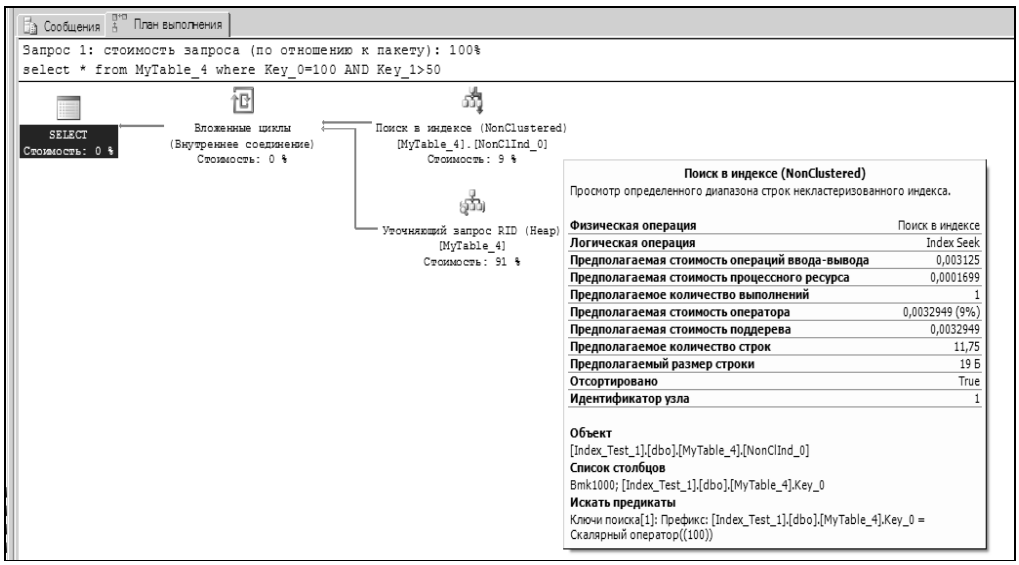


Рис. 4.11
Пример графического представления процедурного плана

Для оценки стоимости выполнения каждого физического оператора оптимизатор запроса использует соответствующую этому оператору модель стоимости.

Например, для оператора **Index Seek** эта модель определена как сумма стоимостей операций ввода-вывода (**IO_Cost** = 0,003125) и обработки данных (**ProcessingCost** = 0,0001699), умноженная на количество операций (= 1).

Оценка стоимости операторов процедурного плана выполняется на основании статистических данных, характеризующих состояние объектов, затрагиваемых SQL-запросом. Статистические данные содержат усредненную информацию о таблицах и индексах (например, количество занимаемых страниц, степень селективности предикатов выборки данных, гистограммы распределения значений полей таблиц и т. д.).

MS SQL-сервер собирает статистику в фоновом режиме в соответствии со сценарием, заданным администратором базы данных. Например, процедура сбора статистики может запускаться в фиксированное время суток или после каждой модификации данных, при этом данные таблиц могут собираться как

полностью (для первых 200 строк), так и случайной выборкой части строк больших таблиц.

Соответствующий инструмент среды SQL-Server Management Studio (вкладка «Статистика») позволяет в любой момент времени обновить статистику или просмотреть время ее последнего обновления.

Просмотреть текущее состояние статистики индекса **Ind** таблицы **T1** можно командой **DBCC show_statistics(T1,Ind)**.

Тот из альтернативных планов, стоимость которого оказалась минимальной, получает статус *предполагаемого плана* (*estimated execution plan*) и записывается в хранилище процедурных планов (*PlanCache*), где временно хранится вместе с планами других запросов в прекомпилированном виде (в виде дерева физических операторов) для последующего извлечения и многократного использования.

На этом работа оптимизатора запроса завершается и управление передается подсистеме выполнения запросов (Storage Engine) сервера баз данных.

13.3. Исполнение процедурного плана выполнения запроса

Предполагаемые планы выполнения запросов, записанные в хранилище, не хранятся там вечно — сервер ведет статистику использования сохраненных планов и регулярно удаляет из хранилища редко используемые планы.

При поступлении на обработку очередного SQL-запроса производится проверка наличия в хранилище предполагаемого плана его выполнения.

Если нужного плана в хранилище нет, запускается описанный выше полный процесс трансляции исходного SQL-кода (фазы с 1-й по 5-ю), по завершении которого сформированный оптимизатором запросов предварительный план записывается в хранилище.

При наличии соответствующего предварительного плана он извлекается из хранилища, проверяется возможность его выполнения в текущем состоянии базы данных, и если план осуществим, он получает статус действительного (*actual*) плана, который затем компилируется в машинный код и исполняется.

Если принимается решение о невозможности реализации ранее сохраненного предполагаемого плана, повторно запускается процесс оптимизации (4-я и 5-я фазы трансляции запроса — генерация альтернативных процедурных планов, их оценивание и выбор оптимального предварительного плана). По завершении процесса оптимизации старый план заменяется в хранилище новым планом.

Во многих случаях предполагаемый и действительный планы будут совпадать. Типичные причины нереализуемости сохраненного ранее предварительного плана:

– статистика, на основе которой был сформирован предполагаемый план выполнения запроса, к текущему моменту либо устарела («*out of date*»), либо была обновлена;

– логические объекты базы данных, затрагиваемые запросом, были модифицированы после создания процедурного плана (например, были созданы но-

вые индексы, изменены или удалены старые, удалены временные таблицы, на которые ссылается запрос, и т. д.).

13.4. Средства анализа и визуализации процедурных планов

На завершающем этапе процесса отладки SQL-запроса (хранимого представления, процедуры или пакета, состоящего из множества таких объектов в любых их комбинациях) можно просмотреть не только результат его выполнения, но также и соответствующий процедурный план, сгенерированный оптимизатором запросов. При этом предусмотрена возможность визуализации как предполагаемого плана, извлекаемого из хранилища, так и действительного плана, актуального в текущих условиях реализации запроса.

Язык TransactSQL содержит инструкции группы SET, позволяющие сохранить в текстовом формате или в формате XML-документа как предполагаемый, так и действительный план выполнения запроса (табл. 4.9).

Инструкция **SET SHOWPLAN_... ON/OFF** включает/выключает соответствующий режим сохранения предварительного плана, извлекаемого из хранилища, блокируя при этом исполнение запроса.

Инструкция **SET STATISTICS_... ON/OFF** включает/выключает режим отображения действительного плана, не препятствуя исполнению запроса.

Таблица 4.9

Операторы управления отображением процедурных планов

SET SHOWPLAN_XML	Возвращает сведения о приблизительном плане выполнения в виде XML-документа
SET SHOWPLAN_TEXT	Возвращает сведения о приблизительном плане выполнения
SET SHOWPLAN_ALL	Возвращает полную информацию о приблизительном плане выполнения запроса
SET STATISTICS XML	Возвращает сведения о действительном плане выполнения в виде XML-документа
SET STATISTICS PROFILE	Возвращает полную информацию о действительном плане выполнения запроса
SET STATISTICS IO	Отображает сведения о дисковой активности во время выполнения запроса
SET STATISTICS TIME	Отображает время (в миллисекундах), которое потребовалось для синтаксического анализа, компиляции и выполнения запроса

На рисунке 4.12 приведен пример отображения действительного плана выполнения следующего SQL-запроса:

```
SET STATISTICS XML ON
SET STATISTICS PROFILE ON
SET STATISTICS IO ON
SET STATISTICS TIME ON
Select * From T1 Inner Join T2 ON T1.c2 > T2.c2
Where T2.c3<500 OR T1.c3<500;
```

Такой формат табличного отображения плана запроса достаточно информативен: каждая строка представляет один физический оператор и содержит код этого оператора, информацию о времени его выполнения и количестве обработанных строк.

Примечание. Приведенный пример — результат реализации запроса в MS SQL-Server 2005. В более поздних версиях возможность отображения планов в текстовом формате не предусмотрена, хотя соответствующие команды оставлены для совместимости со старшими версиями. В более поздних версиях в результате применения любой из команд, приведенных в таблице 4.9, план запроса будет сохранен в XML-формате, однако утилита просмотра такого плана отобразит его на экране в графическом формате, как показано на рисунках 4.11 и 4.14.

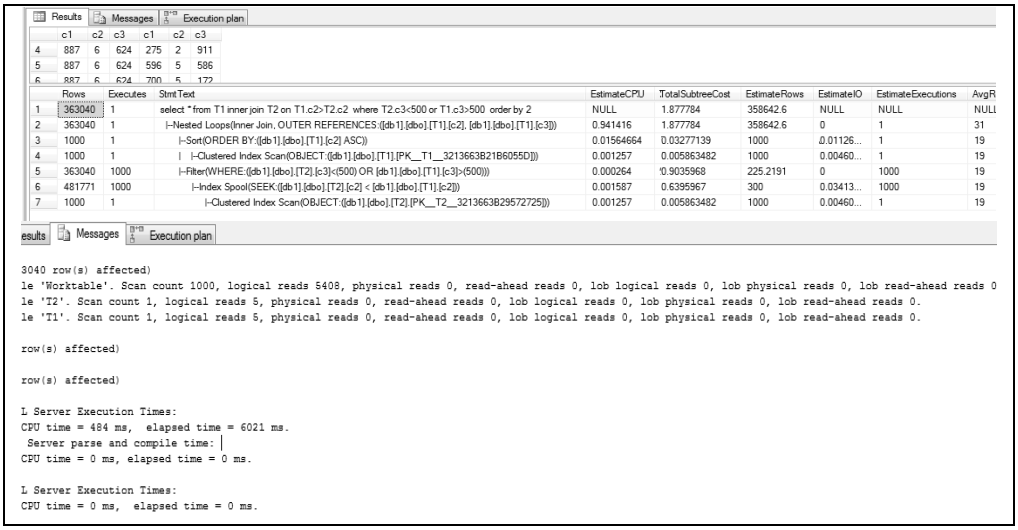


Рис. 4.12
Пример табличного представления процедурного плана

Среда SQL-Server Management Studio предоставляет возможность графического представления предварительного и действительного планов выполнения запроса в виде дерева физических процедурных операторов.

Для включения/отключения режимов графического отображения планов на экранной панели имеются соответствующие кнопки (рис. 4.13).

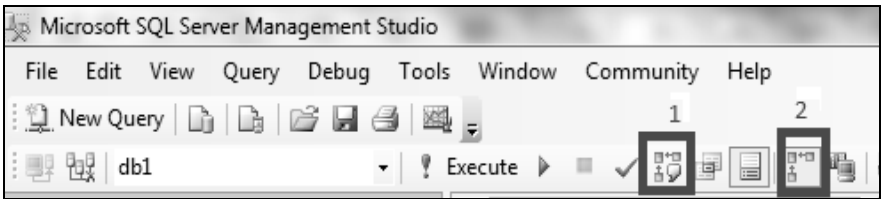


Рис. 4.13
Режимы просмотра графического плана:

1 — предполагаемый план; *2* — действительный план.

Следует помнить, что при включении режима отображения предварительного плана (так же как и в случае с применением команды **SET SHOWPLAN_...ON)** выполнение запроса блокируется.

На рисунке 4.14 приведен пример графического отображения действительного плана выполнения запроса из приведенного выше примера.

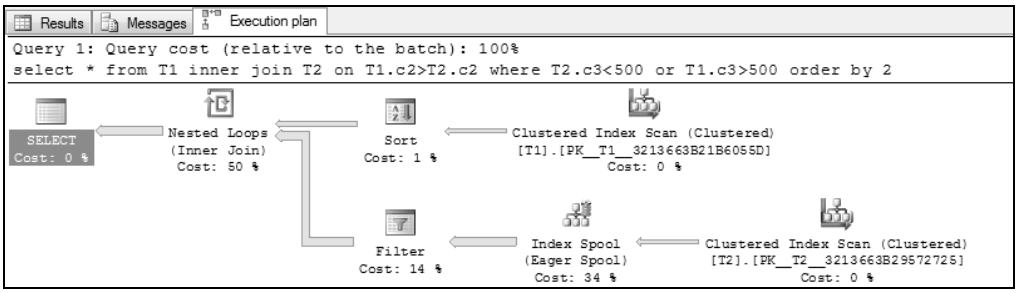


Рис. 4.14

Пример графического представления плана запроса

Каждый оператор на схеме плана представляется соответствующим графическим символом (табл. 4.10), под которым отображается код этого оператора. При наведении курсора на оператор на экране появляется всплывающая подсказка (ToolTips), как это показано на рисунке 4.15. Подсказка содержит имена логического и физического операторов, количество строк, возвращаемых оператором (предполагаемое и фактическое), стоимость операции, количество исполнений и другую информацию, полезную для анализа процедурного плана.

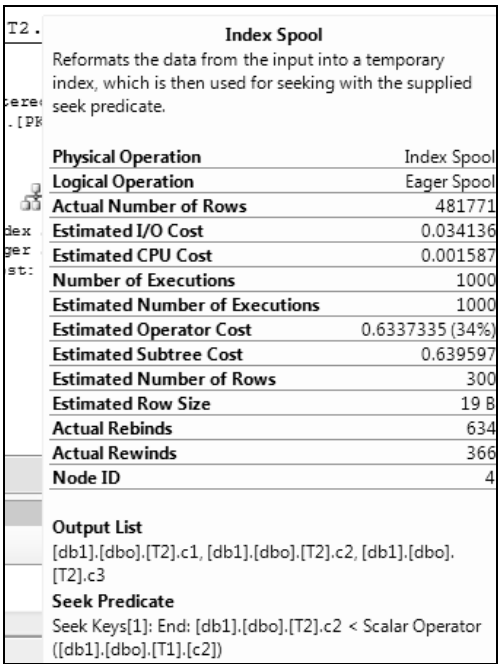


Рис. 4.15

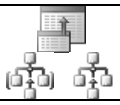
Всплывающая подсказка

Читать графические планы следует справа налево, в соответствии с направлением стрелок. Толщина стрелки также информативна — она пропорциональна количеству передаваемых строк. При наведении курсора на стрелку на экране также появляется соответствующая подсказка.

С полным перечнем низкоуровневых операторов процедурных планов можно ознакомиться на официальном ресурсе разработчика.

Таблица 4.10

Графическое представление операторов процедурных планов

	Table Scan	Сканирует таблицу (кучу)
	Clustered Index Scan Index Scan	Сканирует кластеризованный индекс/ некластеризованный индекс
	Clustered Index Seek Index Seek	Производит поиск по кластеризованному индексу/ некластеризованному индексу
	Key Lookup	Производит поиск закладок в таблице с кластеризованным индексом
	RID Lookup	Производит поиск закладки в куче по заданному идентификатору строки
	Table Insert Clustered Index Insert Index Insert	Вставляет строки в таблицу/ кластеризованный индекс/ некластеризованный индекс
	Filter	Просматривает входные данные и возвращает строки, удовлетворяющие критерию фильтрации
	Nested Loops	Соединяет таблицы по методу вложенных циклов

ГЛАВА 14. ПРАКТИКУМ ПО АДМИНИСТРИРОВАНИЮ

14.1. Общие методические указания

Структура и содержание. Практикум содержит четыре практические работы, отражающие следующие аспекты администрирования:

– управление физической моделью базы данных (работы № 1 и 2);

– управление индексными структурами данных (работа № 3);

– анализ процедурных планов выполнения SQL-запросов (работа № 4).

Каждая работа содержит несколько взаимосвязанных заданий, выполнение которых направлено на решение поставленных в работе задач и требует освоения и применения соответствующих инструментальных средств администрирования баз данных.

Программное обеспечение. Все работы выполняются в системе SQL-Server Management Studio, версия сервера баз данных — не старше 2008R2.

Отчет по работе должен содержать:

– цели и задачи, описание методики проведения работы, используемых структур данных и инструментальных программных средств;

– иллюстративный материал (листинги программных компонентов, выводимые на экран результаты их работы, графический материал и пр.);

– анализ полученных результатов с собственными выводами;

– ответы на контрольные вопросы (при их наличии).

Защита. Работа выполняется индивидуально, защита работы проводится в форме собеседования по материалу представленного отчета. В процессе защиты оценивается полнота и качество выполнения практических заданий, грамотность использования инструментальных средств, правильность и обоснованность выводов по результатам работы, качество оформления отчета.

14.2. РАБОТА № 1. Анализ файловой структуры баз данных

Цель работы: ознакомление с программной архитектурой сервера баз данных и приобретение практических навыков применения инструментальных программных средств, используемых разработчиками и администраторами для управления файловой структурой баз данных.

Задачи:

– изучить пользовательский интерфейс программной среды *SQL-Server Management Studio*;

– исследовать файловую структуру системной базы данных «***Model»***, используемую в качестве шаблона для создания пользовательских баз данных;

– освоить технику создания пользовательских баз данных средствами *MS SQL-Server Management Studio* и соответствующими средствами языка *Transact SQL* (операторы Create Database, Create Table, Alter Table);

– освоить технику модификации параметров файловой структуры пользовательских баз данных средствами *MS SQL-Server Management Studio,* средствами языка *Transact SQL* (оператор Alter Database);

– исследовать объекты системного каталога базы данных, ответственные за хранение параметров ее файловой структуры.

Задание 1. Анализ файловой структуры базы данных «Model»

1.1. Активизируйте системную БД «***Model***». Определите свойства файловой структуры (состав и имена файлов и файловых групп, размеры и прочие параметры файлов) этой базы данных.

1.2. Просмотрите и проанализируйте схемы и содержимое системных таблиц SysFileGroups и SysFiles этой базы данных (через соответствующие им одноименные системные представления (Sys.SysFileGroups и Sys.SysFiles) и прямым доступом к этим таблицам SQL-оператором Select).

1.3. На базе таблиц SysFileGroups и SysFiles создайте хранимое представление (Create View) для визуализации информации о технических параметрах файлов базы данных и их распределения по группам.

1.4. Сохраните результаты выполнения задания в отчете.

Задание 2. Создание пользовательских баз данных

2.1. Создайте пользовательскую БД и сформируйте ее схему (2–3 связанные таблицы) средствами *SQL-Server Management Studio.*

2.2. Активизируйте созданную БД и, не заполняя таблиц данными, выполните в контексте этой базы данных задание 1.2.

2.3. В контексте этой БД выполните хранимое представление, созданное при выполнении задания 1.3. Проанализируйте результаты.

2.4. Создайте еще одну пользовательскую БД средствами *Transact SQL* (оператор Create Database). Создайте в этой БД две вторичные файловые группы, одной из которых установите свойство «по умолчанию». Создайте по два вторичных файла в каждой из вторичных файловых групп. Создайте в этой БД 4–5 простых таблиц, определите для этих таблиц файловые группы.

2.5. Повторите задания 2.2 и 2.3 в контексте новой базы данных.

2.6. Сохраните результаты выполнения задания в отчете.

Задание 3. Модификация файловой структуры баз данных

3.1. Используя средства *SQL-Server Management Studio,* измените параметры файловой структуры одной из пользовательских БД, созданных при выполнении предыдущего задания:

– увеличите в 2 раза начальный размер первичного файла БД;

– уменьшите в 2 раза шаг приращения размера этого файла;

– создайте две дополнительные (вторичные) файловые группы;

– создайте во вторичных файловых группах по два (вторичных) файла *базы данных*.

3.2. В контексте этой (модифицированной) БД выполните запрос, созданный при выполнении задания 1.3. Проанализируйте результат.

3.3. Измените по своему усмотрению параметры файловой структуры системной базы данных «***Model***» (размер первичного файла БД, количество вторичных файловых групп и вторичных файлов).

3.4. Создайте новую пользовательскую БД, в контексте этой БД выполните задания 2.2 и 2.3, результаты сохраните в отчете.

14.3. РАБОТА № 2. Анализ алгоритмов резервирования памяти

Цель работы: исследование типовых алгоритмов управления процессами распределения файловых страниц между логическими объектами базы данных и приобретение практических навыков использования инструментальных средств администратора для анализа и управления физической моделью данных.

Задачи:

– изучить внутреннюю организацию файлов БД в системе MS SQL-Server и внутреннюю организацию файловых страниц;

– изучить структуру объектов системного каталога БД, ответственных за хранение параметров ее физической модели;

– освоить языковые (TransactSQL) средства создания и модификации логических объектов БД;

– освоить технику анализа физической модели БД с использованием команд системной утилиты DBCC и системных хранимых процедур;

– исследовать алгоритмы выделения дисковой памяти для хранения логических объектов базы данных (таблиц и индексов), реализованные в MS SQL-Server.

Методические указания

Лабораторная работа содержит четыре задания, каждое из которых предполагает постановку несложного эксперимента с привлечением программных компонентов базы данных и последующий анализ полученных результатов.

В процессе выполнения заданий продолжается знакомство с компонентами системного каталога базы данных (таблицами SysObjects, SysIndexes и соответствующими им системными представлениями), а также приобретается опыт использования встроенных программных средств, предназначенных для создания и анализа компонентов файловой структуры БД:

– SQL-операторы подмножества DDL языка TransactSQL: CreateDatabase, CreateTable, Create Proc;

– команды системной утилиты DBCC (*DataBase Console Command*): DBCC TraceON, DBCC Page, DBCC ExtentINFO, DBCC ShowContig;

– системные хранимые процедуры и функции: sp_helptext, sp_helpfile, sp_spaceused, Object_ID().

Задание 1. Анализ системного каталога пользовательской БД

1.1. Используя SQL-команду CREATE DATABASE, создайте пользовательскую базу данных с простейшей файловой структурой: один файл данных и одна файловая группа (листинг 4.13).

```
Use master;
CREATE DATABASE MyDB-1
ON
        (NAME = MyDB-1_Dat,
        FILENAME = 'C:\...\...\MyDB-1.mdf',
        SIZE = 4MB,
        MAXSIZE = UNLIMITED,
        FILEGROWTH = 1MB)
LOG ON
        (NAME = MyDB-1_log,
        FILENAME = 'D:\...\...\MyDB-1.ldf',
        SIZE = 2MB,
        MAXSIZE = 10MB,
        FILEGROWTH = 15%);
```

Листинг 4.13
Пример создания БД

1.2. Проконтролируйте и сохраните в отчете параметры БД, созданной в результате выполнения приведенного выше SQL-кода:

а) прямым доступом оператором Select к таблицам SysFiles и SysFileGroups системного каталога базы данных MyDB-1;

б) с использованием хранимой процедуры sp_helpfile (листинг 4.14).

```
Use MyDB-1;
EXEC sp_helpfile;
```

Листинг 4.14
Пример выполнения хранимой процедуры

1.3. Используя SQL-команду CREATE TABLE, создайте в этой *базе данных* простую унарную таблицу MyTable_1, каждая строка которой будет занимать ровно одну дисковую страницу (листинг 4.15):

```
Use MyDB-1;
CREATE TABLE MyTable_1
(column1 char(8000) NOT NULL
default 'One row in one page');
```

Листинг 4.15
Пример создания унарной таблицы

1.4. Прямым доступом оператором к таблице SysObjects системного каталога БД MyDB-1 определите и сохраните в отчете:

а) параметры name, Id, xtype и crdate для вновь созданного объекта (таблицы MyTable_1);

б) общее количество объектов такого же типа (xtype) в этой БД;

в) общее количество объектов с такой же датой создания (crdate) в этой БД;

г) общее количество пользовательских таблиц (xtype = 'U'), системных таблиц (xtype = 'S'), хранимых представлений (xtype = 'V') и хранимых процедур (xtype = 'P') в этой БД.

1.5. Используя встроенную функцию Object_ID(), определите идентификатор объекта (таблицы MyTable_1) по его имени.

```
Use MyDB-1;
Select Object_ID('MyTable_1');
```

Листинг 4.16

Пример использования функции Object_ID()

1.6. Прямым доступом к таблице SysIndexes системного каталога БД MyDB-1 определите и сохраните в отчете:

а) количество строк в этой таблице, соответствующих объекту MyTable1 (для ограничения выборки используйте конструкцию: where ID = Object_ID('MyTable_1'));

б) значения полей Id, IndId, First, Root, FirstIAM для таблицы MyTable_1;

в) количество и номера (адреса PageNum) дисковых страниц, владельцем которых является таблица MyTable_1 (в текущей ситуации эта таблица еще не содержит ни одной строки данных).

1.7. Сформулируйте и сохраните в отчете ответы на следующие вопросы.

1. Каковы результаты трансляции SQL-запроса CREATE TABLE … и в каких таблицах системного каталога MS SQL-Server сохраняет эти результаты?

2. Каково назначение системной таблицы SysObjects?

3. Какую информацию несут значения полей name, Id, xtype и crdate в таблице SysObjects?

4. Каково назначение системной таблицы SysIndexes?

5. Какую информацию несут значения полей Id и IndId в таблице SysIndexes?

6. Какую информацию несут значения полей First, Root и FirstIAM и в каком формате представлена эта информация в таблице SysIndexes?

7. Сколько страниц занимает в файле данных «пустая» таблица?

8. Для чего можно использовать встроенные хранимые процедуры sp_helpfile и sp_helptext и встроенную функцию Object_ID()? Приведите примеры.

Задание 2. Исследование алгоритма резервирования памяти в базах данных с простой файловой структурой

Продолжим эксплуатировать БД MyDB-1, созданную при выполнении предыдущего задания. Эта база содержит единственный (primary) файл данных, ассоциированный с единственной (primary) файловой группой.

В соответствии с результатами выполнения задания 1.4, часть страниц типа Data этого файла занята объектами системного каталога, а единственная пользовательская таблица MyTable_1 пока пуста и не владеет ни одной из файловых страниц этого типа.

Проведя несложный эксперимент, исследуйте процесс резервирования и заполнения файловых страниц и распределения их по экстентам соответствующих типов (смешанных — mixed или однородных — uniform) при последовательной вставке строк в таблицы.

2.1. Создайте в БД MyDB-1 еще одну таблицу, например MyTable-2, с такой же схемой, как у таблицы MyTable-1 (используйте SQL-код, подобный приведенному в листинге 4.15).

2.2. Вставьте по одной строке в обе эти таблицы (листинг 4.17) и затем повторно выполните задание 1.6 для двух таблиц (теперь обе таблицы не пусты и каждая из них является владельцем как минимум двух страниц: одной страницы типа Data и одной IAM-страницы).

```
Use MyDB-1;
INSERT INTO MyTable_1 DEFAULT VALUES;
INSERT INTO MyTable_2 DEFAULT VALUES;
```

Листинг 4.17
Вставка одной строки

Определите номера экстентов, содержащих страницы, выделенные двум таблицам при вставке в них строк данных. К какому типу относятся эти экстенты — uniform или mixed? Ответ обоснуйте и сохраните в отчете.

2.3. С помощью команды PAGE системной утилиты DBCC просмотрите заголовки и основное содержимое этих страниц (для каждой из двух таблиц).

```
DBCC TRACEON (3604)
DBCC PAGE ('MyDB-1',<Id файла>,<Id страницы>,0)
```

Листинг 4.18
Пример выполнения команды PAGE

С четвертым параметром команды PAGE придется поэкспериментировать: значение этого параметра (0, 1, 2 или 3) влияет (по-разному для различных типов страниц) на объем и формат выводимой на экран информации.

2.4. Используя хранимую процедуру sp_spaceused, определите количество страниц, занятых каждой из этих таблиц.

```
Use MyDB-1;
EXEC sp_spaceused MyTable_1;
EXEC sp_spaceused MyTable_2;
```

Листинг 4.19
Пример выполнения процедуры sp_spaceused

Прокомментируйте результаты работы этой процедуры, представленные на экране в табличном виде, как это показано на рисунке 4.16.

NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED
MyTable_1	1	–	–	–	–

NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED
MyTable_2	1	–	–	–	–

Рис. 4.16
Форма представления результатов выполнения процедуры sp_spaceused

Подтвердите или скорректируйте свой ответ на вопрос, сформулированный в задании 2.2.

2.5. Последовательно добавляя в обе таблицы еще по 4 строки (листинг 4.20), определите после каждой вставки общее количество зарезервированных страниц (поле RESERVED), количество страниц, занятых строками таблиц (поле DATA), и количество зарезервированных, но еще не использованных страниц (поле UNUSED).

```
Use MyDB-1;
INSERT INTO MyTable_1 DEFAULT VALUES;
EXEC sp_spaceused MyTable_1;
Go 4;
INSERT INTO MyTable_1 DEFAULT VALUES;
EXEC sp_spaceused MyTable_2;
Go  4;
```

Листинг 4.20
Вставка 4-х строк с контролем занятого пространства

Обратите внимание на динамику изменения трех указанных выше параметров при изменении количества вставленных в таблицы строк. Теперь в каждой таблице по 5 строк и, соответственно, каждая таблица является владельцем пяти файловых страниц типа DATA и одной IAM-страницы.

В экстентах какого типа (uniform или mixed) размещены страницы, выделенные двум этим таблицам?

2.6. Повторите предыдущий опыт — вставьте еще по 5 строк в обе эти таблицы (Go 5) и проанализируйте полученный результат.

2.7. Вставьте в эти таблицы еще по 50 строк, проанализируйте полученный результат с помощью процедуры sp_spaceused и дополнительно с помощью команды DBCC EXTENTINFO (листинг 4.21), которая отобразит на экране

информацию об идентификаторах занятых страниц (поля file_id и page_id), количестве выделенных (ext_size) и фактически заполненных (pg_alloc) страниц.

2.8. Если SQL-Server все еще выделяет таблицам страницы в смешанных экстентах, продолжайте вставлять строки в таблицы до тех пор, пока сервер не начнет резервировать однородные экстенты для страниц каждой из таблиц.

```
DBCC EXTENTINFO(MyDB-1,MyTable_1,-1[8])
```

Листинг 4.21
Пример выполнения команды EXTENTINFO

2.9. С помощью команды DBCC PAGE просмотрите содержимое страниц типа GAM и SGAM (позиции этих страниц в файле фиксированы: № 2 — для GAM и № 3 — для SGAM). Какие свойства (двухбитовые коды) получили экстенты, зарезервированные для хранения строк таблиц MyTable_1 и MyTable_2?

2.10. С помощью команды DBCC PAGE просмотрите содержимое страницы типа PFS (фиксированная позиция № 1 в файле данных). Определите степень заполнения нескольких страниц, выделенных таблицам MyTable_1 и MyTable_2.

2.11. Сформулируйте и сохраните в отчете ответы на следующие вопросы.

1. В каком формате хранятся номера страниц в таблице SysIndeses?
2. Может ли логический объект базы данных (например, таблица) быть владельцем единственной файловой страницы?
3. Может ли логический объект базы данных быть владельцем нескольких файловых страниц типа DATA?
4. Может ли одна файловая страница типа DATA иметь более чем одного владельца?
5. Может ли одна файловая страница типа DATA входить в состав более чем одного экстента?
6. В каких случаях MS SQL-Server резервирует смешанные экстенты?
7. В каких случаях MS SQL-Server резервирует однородные экстенты?
8. В каких структурах данных и в каком формате SQL-Server хранит информацию о свободных экстентах, типах зарезервированных экстентов и свободном пространстве внутри файловых страниц?
9. Какие эксплуатационные показатели использовались в качестве критериев при реализации стратегии резервирования экстентов?

Задание 3. Исследование алгоритма распределения памяти в базах данных со сложной файловой структурой

Если в предыдущем задании анализировался процесс резервирования экстентов и страниц единственного файла данных, то теперь ставится задача экспериментального исследования алгоритма распределения страниц одного логического объекта (таблицы) между несколькими файлами данных.

[8] Последним параметром команды EXTENTINFO можно указывать либо имя индекса таблицы (для вывода информации о страницах этого индекса), либо число 0 (для вывода информации о страницах, занятых строками таблицы), либо число –1 (для вывода информации о страницах, занятых строками таблицы и всеми ее индексами).

Рабочая гипотеза, которую следует подтвердить, опровергнуть или уточнить в результате выполнения этого задания, может быть сформулирована следующим образом: «*Если файловая группа содержит более одного файла типа* DATA*, то при вставке строк в таблицу, ассоциированную с этой файловой группой, количество файловых страниц, выделяемых сервером в каждом из файлов, будет пропорционально их размерам*».

Для выполнения задания потребуется создать несколько баз данных, имеющих более сложную (по сравнению с MyDB-1) файловую структуру: несколько файловых групп и несколько файлов разных размеров в каждой группе.

3.1. Создайте новую пользовательскую базу данных (например, MyDB-2) со следующей файловой структурой:

– две файловые группы (группа Primary со свойством по умолчанию и дополнительная группа Group2);

– четыре файла типа DATA (первичный файл и три вторичных файла (File1, File2 и File3), принадлежащих группе Group2);

– установите начальные размеры вторичных файлов: Size = 5, 10 и 15 Mb соответственно;

– установите остальные размерные параметры, одинаковые для всех этих трех файлов: MaxSize = Unlimited; Grows = 1 Mb.

3.2. Проконтролируйте и сохраните в отчете полученный результат с использованием прямого доступа к системной таблице sysfiles.

3.3. Создайте в базе данных MyDB-2 новую таблицу MyTable_3 (листинг 4.22) — каждая строка этой таблицы будет занимать ровно одну файловую страницу, и все эти страницы (в случае заполнения строк таблицы) будут размещены во вторичных файлах, включенных в группу Group2.

```
Use MyDB-2;
CREATE TABLE MyTable_3
(column1 char(8000) NOT NULL
default 'One row in one page')
ON Group2;
```

Листинг 4.22
Пример создания таблицы, связанной с файловой группой

3.4. Учитывая тот факт, что все системные объекты БД MyDB-2 будут размещены в ее первичном файле, а вторичные файлы будут заняты исключительно пользовательскими данными таблицы MyTable_3, рассчитайте (приблизительно) максимальное количество страниц этой таблицы, соответствующее начальному размеру каждого из трех файлов группы Group2.

3.5. Используя листинг 4.23, напишите пользовательскую хранимую процедуру AddRows, при выполнении которой в таблицу будет вставлено заданное количество строк со значениями полей по умолчанию, указанными при создании таблицы.

3.6. Используя процедуру AddRows, вставьте в таблицу MyTable_3 расчетное количество строк так, чтобы меньший из трех вторичных файлов оказался заполненным примерно наполовину. Проконтролируйте и сохраните в отчете полученный результат.

```
CREATE PROC AddRows @Tablename char(12),@maxrows int
 AS
  SET nocount off
  DECLARE @count INT
  SET @count = 0
  WHILE @count < @maxrows
  BEGIN
   INSERT INTO @Tablename DEFAULT VALUES
   SET @count = @count + 1
  END
```

Листинг 4.23
Пример SQL-кода для создания хранимой процедуры

3.7. Многократно повторяя п. 3.6, добейтесь ситуации, когда наибольший (по начальному размеру) из вторичных файлов увеличится по размеру примерно вдвое.

3.8. По результатам проведенного эксперимента опишите и сохраните в отчете алгоритм распределения страниц типа DATA между файлами одной файловой группы в случае, когда начальные размеры файлов различны, но их предельные размеры не ограничены.

3.9. Создайте новую пользовательскую базу данных (например, MyDB-3) с файловой структурой, аналогичной MyDB-2, но с различными параметрами MaxSize для вторичных файлов.

3.10. Повторите п. 3.2–3.7 этого задания.

3.11. По результатам проведенного эксперимента постройте графики зависимостей размеров файлов от количества строк таблицы, опишите и сохраните в отчете алгоритм распределения страниц типа DATA между файлами одной файловой группы в случае, когда различны и начальные, и предельные размеры файлов.

Задание 4. Исследование структуры файловой страницы типа DATA

При выполнении предыдущих заданий исследовался процесс резервирования файловых страниц при вставке строк в таблицы, и для этого было удобно использовать унарные таблицы с полями типа char(8000), чтобы каждая строка таблицы занимала целую страницу. В реальной ситуации страница может содержать несколько строк таблицы и при этом иметь свободное пространство для вставки в таблицу последующих строк.

Объектом исследования в этом задании является внутренняя структура файловой страницы и процесс ее заполнения строками таблицы.

Для выполнения задания рекомендуется создать новую базу данных (например, MyDB-4) с простой файловой структурой.

4.1. Создайте бинарную таблицу MyTable_4(Key1 INT, Data CHAR(10)) с длиной строки 14 байт.

4.2. Вставьте в таблицу 10 строк, заполнив поля случайными данными.

```
USE MyDB-4
DECLARE @key1 INT, @data CHAR(10)
SET @key1=1000*RAND(), @data=STR(@key1)
INSERT into MyTable_4 values(@Key1,@data)
Go 10
```

Листинг 4.24
Вставка 10-ти строк в таблицу

Примечание. Функция RAND() возвращает псевдослучайное число в диапазоне (0–1), а функция STR() преобразует число в соответствующую цифровую строку.

4.3. Определите номер первой файловой страницы (SysIndexes.First), выделенной этой таблице, и просмотрите страницу командой DBCC PAGE(). Определите и сохраните в отчете:

– количество слотов страницы, занятых строками таблицы;

– смещения (в байтах) каждого слота;

– длину (в байтах) каждого слота.

4.4. Вставьте еще 100 строк в таблицу и повторно выполните п. 4.3.

4.5. Просмотрите PFS-страницу командой DBCC PAGE, определите процент заполнения первой страницы, выделенной таблице.

4.6. Выполняйте п. 4.4 и 4.5 до 100%-ного заполнения первой страницы выделенной таблицы. Определите и сохраните в отчете:

– количество слотов первой страницы, занятых строками таблицы;

– суммарный объем страницы, занятый заполненными слотами;

– суммарный объем страницы (в байтах), занятый областью обратных ссылок (offset table).

4.7. Просматривая первую страницу таблицы, выберите по своему усмотрению один из заполненных слотов, запомните его номер и значения полей соответствующей строки таблицы. Затем удалите из таблицы эту строку (Delete MyTable_4 Where Key1=…).

4.8. Повторно просмотрите страницу командой DBCC PAGE, обращая внимание на слот с удаленной строкой. Прокомментируйте и попытайтесь объяснить полученный результат.

14.4. РАБОТА № 3. Исследование индексных структур данных

Цель работы: изучение индексных структур и приобретение навыков использования инструментальных средств управления индексами.

Задачи:

– освоить программные средства создания, модификации и анализа индексных структур данных;

– изучить структуру объектов системного каталога, ответственных за хранение параметров индексов;

– изучить формат индексных страниц для различных типов индексов.

Методические указания

Работа содержит четыре взаимосвязанных задания, каждое из которых направлено на изучение многоуровневых индексных структур данных для индексов четырех различных типов:

– некластеризованного индекса по неуникальным столбцам таблицы при условии отсутствия кластеризованного индекса;

– кластеризованного уникального индекса;

– некластеризованного индекса при условии наличия в таблице кластеризованного индекса;

– некластеризованного индекса с включенными столбцами.

Каждое задание предполагает постановку несложного эксперимента с привлечением программных компонентов базы данных и последующее проведение анализа полученных результатов.

В процессе выполнения заданий продолжится знакомство с компонентами системного каталога базы данных (таблица SysIndexes) и программными средствами, используемыми для управления индексами:

– SQL-операторы Create/Alter/Drop Index;

– команды Page и ShowContig системной утилиты DBCC;

– системная хранимая процедура sp_spaceused;

– TVF-функция sys.dm_db_index_physical_stats().

Задание 1. Анализ структуры индексных страниц для неуникального некластеризованного индекса

1.1. Используя SQL-команду Create Database, создайте пользовательскую базу данных (например, Index_Test_1) с простейшей файловой структурой: один файл данных в одной группе.

1.2. Используя SQL-команду Create Table, создайте в этой БД таблицу MyTable_4, схема которой включает три целочисленных столбца и четвертый столбец строкового типа.

```
Use Index_Test_1
CREATE TABLE MyTable_4
(Key_0 INT NOT NULL,
Key_1 INT NOT NULL,
Key_2 INT NOT NULL,
Data CHAR(61) NOT NULL)
```

Листинг 4.25
Создание таблицы MyTable_4

Примечание. Длина поля Data (61 байт) выбрана для удобства просмотра файловых страниц командой DBCC PAGE: так как каждый слот страницы

*содержит два служебных поля суммарной длиной в 7 байт, то общая длина каждого слота составит ровно 80 байт (7+3*4+61).*

1.3. Определите идентификатор этой таблицы и убедитесь в том, что системная таблица SysIndexes содержит ровно одну запись, соответствующую таблице MyTable_4, и при этом таблица MyTable_4 не является владельцем ни одной файловой страницы. Объясните этот факт.

1.4. Вставьте в таблицу MyTable_4 одну строку данных.

```
USE Index_Test_1
DECLARE @key0 INT, @key1 INT, @key2 INT, @dat CHAR(30)
SET @key0=1000*RAND(),@key1=1000*RAND(), @key2=1000*RAND()
SET @dat=STR(@key0)+STR(@key1)+STR(@key2)
INSERT into MyTable_4 values(@key0,@key1,@key2,@dat)
```

Листинг 4.26
Вставка строки в таблицу MyTable_4

1.5. Используя хранимую процедуру sp_spaceused, определите количество страниц, занятых данными этой таблицы и всеми ее индексами, включая IAM-страницу.

1.6. Повторным запросом к таблице SysIndexes определите:

– количество записей в системной таблице SysIndexes, соответствующих таблице MyTable_4;

– значения полей Root, First, FirstIAM и IndID для каждой из этих записей;

– категории объектов (Heap, ClusteredIndex или NonClusteredIndex), соответствующих таблице MyTable_4.

1.7. С помощью команды DBCC PAGE просмотрите содержимое всех страниц, владельцем которых является таблица MyTable_4.

1.8. Создайте *НЕ*кластеризованные *НЕ*уникальные индексы по полям Key_0, Key_1 и Key_2 таблицы MyTable_4 (листинг 4.26), затем повторно выполните п. 1.5 и 1.6.

```
Use Index_Test_1
Create NONCLUSTERED Index NonClInd_0
ON MyTable_4(Key_0)
```

Листинг 4.27
Создание некластеризованного индекса NonClInd_0 по полю Key_0

1.9. Командой DBCC PAGE просмотрите заголовки и основное содержимое корневых и листовых страниц всех трех индексов (используйте значения 1, 2 и 3 для четвертого параметра этой команды). Определите:

– глубину индексов;

– номера страниц корневого и листового уровней;

– формат ссылки с корневого уровня индекса на промежуточный (не листовой) уровень;

– формат ссылки с листовой страницы на страницу данных в куче (heap).

1.10. Вставьте в таблицу еще 1000 строк.

1.11. Проанализируйте полученный результат с помощью хранимой процедуры sp_spaceused, команды DBCC ShowContig и TVF-функции sys.dm_db_index_physical_stats().

1.12. Вставьте в таблицу еще 100 000 строк (возможно, придется немного подождать) и выполните повторный анализ полученного результата.

1.13. Сформулируйте ответы на следующие вопросы.

1. Для чего и в каких случаях рекомендуется использовать индексы следующих типов: «кластеризованный индекс», «некластеризованный индекс», «уникальный индекс», «индекс с включенными стобцами»?

2. На какие эксплуатационные показатели работы базы данных оказывает влияние индексирование данных в таблицах?

3. В каких ситуациях наличие индексированных столбцов таблиц может привести к снижению производительности работы базы данных?

4. Какие ограничения накладывает SQL-Server на использование индексов?

5. Каков формат ссылки с корневого уровня индекса на промежуточный (не листовой) уровень?

6. Каков формат ссылки с листовой страницы некластеризованного индекса при условии, что в таблице отсутствует кластерный индекс?

7. Как связаны между собой значения параметров «порядок индекса» и «глубина индекса» индексной структуры данных?

8. Какова глубина индексов, построенных при выполнении заданий 1.8, 1.10 и 1.12?

Задание 2. Анализ структуры индексных страниц для кластеризованного индекса

Для выполнения задания рекомендуется создать новую базу данных (например, Index_Test_2) с простейшей файловой структурой.

2.1. Создайте в БД Index_Test_2 таблицу MyTable_5 (листинг 4.27).

```
use Index_Test_2
CREATE TABLE MyTable_5
(Key_0 IDENTITY CONSTRAINT Key0_PK PRIMARY KEY,
Key_1 INT,Key_2 INT, Data CHAR(68))
```

Листинг 4.28

Пример создания таблицы с первичным ключом

Примечание 1. Параметр IDENTITY*, установленный для целочисленного поля Key_0, присваивает полю статус идентификатора, значения которого при вставке строк в таблицу сервер будет присваивать автоматически (по умолчанию автоинкрементно с шагом 1, начиная с 1).*

Примечание 2. Ограничение первичного ключа (PRIMARY KEY) *для поля* Key_0 *гарантирует уникальность значений этого поля в таблице, даже при*

отсутствии свойства IDENTITY. *По умолчанию сервер автоматически создает кластеризованный индекс по первичному ключу.*

2.2. Определите адреса файловых страниц Root, First и FirstIAM для кластеризованного индекса поля Key_0 таблицы MyTable_5.

2.3. Вставьте в таблицу 10 строк (листинг 4.28) и повторите п. 2.2.

```
USE Index_Test_2
DECLARE @key0 INT,@key1 INT,@key2 INT,@dat CHAR(30)
SET @key1=1000*RAND(),@key2=1000*RAND()
SET @dat=STR(@key1)+STR(@key2)
INSERT into MyTable_5 values(@key1,@key2,@dat)
Go 10
```

Листинг 4.29
Вставка 10-ти строк в таблицу MyTable_5

2.4. Выполните задания, аналогичные п. 1.8–1.12, в контексте базы данных Index_Test_2 применительно к таблице MyTable_5.

Задание 3. Анализ структуры индексных страниц некластеризованного индекса при условии наличия кластеризованного индекса

3.1. Для выполнения задания будет использоваться ранее созданная база данных Index_Test_2 и уже существующая и заполненная таблица MyTable_5.

3.2. Создайте в таблице MyTable_5 некластеризованный индекс по полю Key_1.

```
use Index_Test_2
Create NONCLUSTERED Index NonClInd_1
ON MyTable_5(Key_1)
```

Листинг 4.30
Создание индекса в ранее заполненной таблице

3.3. Дождитесь завершения процесса построения индекса и затем повторно выполните задания 2.2–2.4.

Задание 4. Анализ структуры индексных страниц некластеризованного индекса с включенным столбцом

4.1. Для выполнения задания будет использоваться ранее созданная база данных Index_Test_2 и заполненная таблица MyTable_5, в которой уже созданы индексы по двум полям: уникальный кластеризованный индекс по ключевому полю Key_0 и неуникальный некластеризованный индекс по полю Key_1. Остальные два поля таблицы неиндексированы.

4.2. Создайте в таблице MyTable_5 некластеризованный индекс по полю Key_2 с включенным полем Data.

```
use Index_Test_2
Create NONCLUSTERED Index incl_Ind_2
ON MyTable_5(Key_1) INCLUDE(Data)
```

Листинг 4.31
Создание индекса с включенным столбцом

4.3. Повторно выполните задание 3.3.

14.5. РАБОТА № 4. Анализ процедурных планов SQL-запросов

Эта работа завершает цикл из четырех работ, направленных на изучение физической модели данных, поддерживаемой MS SQL-Server.

Цель работы: изучение стратегий построения процедурных планов исполнения SQL-запросов, реализуемых оптимизатором, и приобретение практических навыков анализа и управления производительностью.

Задачи:

– ознакомиться с основными низкоуровневыми операторами, используемыми для построения и описания процедурных планов;

– освоить технику анализа процедурных планов соответствующими языковыми средствами (SET SHOWPLAN, SET STATISTICS), а также средствами их графической визуализации;

– провести экспериментальное исследование влияния индексирования таблиц БД на производительность выполнения типовых SQL-запросов;

– по результатам проведенного анализа сделать выводы о стратегии работы генератора процедурных планов и эффективности применения различных индексных структур.

Методические указания

Работа содержит два задания, каждое из которых направлено на изучение стратегий построения процедурных планов исполнения следующих типовых SQL-запросов при использовании различных типов индексов по столбцам базовых таблиц:

– процедурные планы реализации простейших однотабличных SQL-запросов вида Select … From <Table> Where <condition>;

– процедурные планы реализации SQL-запросов с соединением таблиц;

– процедурные планы реализации SQL-запросов с группировкой данных и использованием агрегатных функций;

– процедурные планы реализации модифицирующих SQL-запросов (Insert, Delete, Update).

Для выполнения работы потребуется создать базу данных, состоящую из нескольких взаимосвязанных таблиц достаточно большой мощности, по столбцам которых сформированы индексы различных типов.

Каждое задание предполагает постановку несложных экспериментов с выполнением типовых SQL-запросов, визуализацией процедурных планов их исполнения и сравнительной оценкой планов по производительности.

В процессе выполнения заданий потребуется использование следующих программных средств:

– SQL-операторов Create/Alter DataBase/Table/Index — для создания/модификации логических объектов базы данных;

– TVF-функции sys.dm_db_index_physical_stats() — для определения параметров индексов;

– команды DBCC show_statistics(t1,ind) — для отображения статистики индекса Ind таблицы T1;

– команды группы SET — для визуализации процедурных планов:

• SHOWPLAN_XML, SHOWPLAN_TEXT, SHOWPLAN_ALL — отображают информацию о предполагаемом (estimated) процедурном плане выполнения запроса, блокируя при этом его выполнение;

• STATISTICS XML, STATISTICS PROFILE — отображают информацию о фактическом (real) процедурном плане исполнения запроса;

• STATISTICS IO, STATISTICS TIME — отображают информацию о дисковой активности и времени выполнения запроса;

– логические и физические процедурные операторы (табл. 4.10), используемые генератором для формирования планов исполнения SQL-запросов.

Задание 1. Анализ процедурных планов реализации однотабличных SQL-запросов

1.1. Запросы выборки из структуры данных типа куча.

Для проведения эксперимента создайте БД (например, Plan_Test1) и в этой БД создайте таблицу MyTable_6.

```
Use Plan_Test1;
CREATE TABLE MyTable_6
(Key_0 INT NOT NULL, Key_1 INT NOT NULL,
Key_2 INT NOT NULL, Data CHAR(8000) NOT NULL);
```

Листинг 4.32
Создание таблицы MyTable_6

Заполните 10 000 строк этой таблицы случайными значениями.

```
USE Plan_Test1;
DECLARE @key0 INT,@key1 INT,@key2 INT,@dat CHAR(30)
SET @key0=1000*RAND(),@key1=1000*RAND(),@key2=1000*RAND()
SET @dat=STR(@key1)+STR(@key2)
INSERT into MyTable_6 values(@key0,@key1,@key2,@dat);
Go 10000
```

Листинг 4.33
Вставка 10 000 строк в таблицу MyTable_6

1.1.1. Прямым доступом к таблице SysIndexes убедитесь в том, что для таблицы MyTable_6 сформирована структура данных типа куча, и определите количество страниц, занятых строками этой таблицы с помощью хранимой процедуры sp_spaceused.

1.1.2. Подготовьте и выполните SQL-запрос выборки всех данных таблицы (Select * From MyTable_6), оцените ее мощность, сравните с результатами работы процедуры sp_spaceused.

1.1.3. Включите (рис. 4.13) режим графического отображения предполагаемого плана выполнения запроса и определите:

– графическую схему предполагаемого плана;

– используемые процедурные операторы;

– стоимость выполнения операторов;

– стоимость операций ввода-вывода;

– стоимость операций обработки данных.

1.1.4. Дополните пакет выполнения предыдущего запроса (п. 1.1.2) командами управления отображением процедурных планов:

– SET STATISTICS XML ON;

– SET STATISTICS PROFILE ON;

– SET STATISTICS IO ON;

– SET STATISTICS TIME ON.

1.1.5. Включите режим графического отображения фактического плана и повторно выполните запрос.

1.1.6. Просмотрите результаты выполнения запроса (вкладки «Результаты», «Сообщения» и «План выполнения»). Определите и сохраните в отчете параметры фактического процедурного плана.

1.1.7. Проведите анализ фактических планов выполнения следующих запросов, содержащих операторы ограничения и группировки строк таблицы:

1) Select * From MyTable_6 Where Key_1=555;
2) Select * From MyTable_6 Where Key_2>666;
3) Select Data From MyTable_6 Where Key_1>20 And Key_2<100;
4) Select * From MyTable_6 Order By Data;
5) Select Key_1,Count(*) From MyTable_6 Group By Key_1.

1.1.8. Сформулируйте выводы о стратегии работы генератора процедурных планов выполнения SQL-запросов выборки данных из кучи.

1.2. Запросы выборки данных из индексированных таблиц.

1.2.1. Создайте таблицу MyTable_7 (аналогичную MyTable_6).

1.2.2. Создайте *НЕ*кластеризованные *НЕ*уникальные индексы по полям Key_0, Key_1 и Key_2 таблицы MyTable_7.

1.2.3. Вставьте 10 строк в таблицу MyTable_7.

1.2.4. Используя данные таблицы SysIndexes, убедитесь в том, что для таблицы MyTable_7 сформирована структура данных типа куча и дополнительно три некластеризованных индекса.

1.2.5. Определите для каждого из индексов глубину индекса и количество индексных страниц на каждом уровне.

1.2.6. Выполните запросы (п. 1.1.2 и 1.1.7) на базе индексированной таблицы MyTable_7, сравните процедурные планы их выполнения с планами запросов выборки из кучи.

1.2.7. Вставьте в таблицу MyTable_7 еще 9990 строк (теперь ее мощность будет равна мощности таблицы MyTable_6).

1.2.8. Просмотрите процедурный план выполнения операции Insert, определите стоимость ее выполнения, сохраните в отчете.

1.2.9. Командой DBCC show_statistics('MyTable_7',<ind>) отобразите статистику всех трех индексов таблицы. Прокомментируйте результаты.

1.2.10. Повторно выполните п. 1.2.6. Оцените повышение производительности запросов на индексированной таблице по сравнению с запросами выборки данных из кучи.

1.2.11. Создайте (листинг 4.33) таблицу MyTable_8 с первичным ключом Key_0 автоинкрементного типа IDENTITY (по этому полю таблицы будет автоматически создан кластеризованный индекс).

```
USE Plan_Test1
CREATE TABLE MyTable_8
(Key_0 INT NOT NULL IDENTITY
CONSTRAINT Key0_PK PRIMARY KEY,
Key_1 INT NOT NULL,
Key_2 INT NOT NULL,
Data CHAR(8000) NOT NULL)
```

Листинг 4.34
Создание таблицы с первичным ключом

1.2.12. Создайте *НЕ*кластеризованные *НЕ*уникальные индексы по полям Key_1 и Key_2 таблицы MyTable_8.

1.2.13. Вставьте в таблицу MyTable_8 10 000 строк.

```
USE Plan_Test1
DECLARE @key1 INT
DECLARE @key2 INT
DECLARE @dat CHAR(30)
SET @key1=1000*RAND()
SET @key2=1000*RAND()
SET @dat=STR(@key1)+STR(@key2)
INSERT into MyTable_8
values(@key1,@key2,@dat)
Go 10000
```

Листинг 4.35
Вставка 10 000 строк

1.2.14. Командой DBCC show_statistics() отобразите статистику всех трех индексов таблицы MyTable_8. Прокомментируйте результаты.

1.2.15. Просмотрите процедурный план выполнения операции Insert, определите стоимость ее выполнения, сохраните в отчете.

1.2.16. Повторно выполните запросы, аналогичные запросам п. 1.1.2 и 1.1.7, сравните процедурные планы их выполнения с планами запросов выборки данных из кучи и с планами запросов выборки данных из таблицы с некластеризованными индексами.

Задание 2. Анализ процедурных планов выполнения SQL-запросов с соединениями таблиц

2.1. Запросы с соединением неиндексированных таблиц.

В этом эксперименте будет использоваться имеющаяся таблица MyTable_6 (индексов нет, мощность — 10 000 строк) и аналогичная ей по структуре новая таблицу MyTable_9 мощностью в 100 строк.

Создайте 4 запроса на базе таблиц MyTable_6 и MyTable_9 и проведите анализ фактических планов их выполнения.

```
1)  Select MyTable_6.Key_0, MyTable_9.Key_1, MyTable_6.Data
    From MyTable_6 Inner Join MyTable_9
        ON MyTable_6.Key_0=MyTable_9.Key_1;
2)  Select MyTable_9.Key_0, MyTable_6.Key_1, MyTable_6.Data
    From MyTable_9 Inner Join MyTable_6
        ON MyTable_9.Key_0 = MyTable_6.Key_1;
3)  Select MyTable_9.Key_0, MyTable_6.Key_1, MyTable_6.Data
    From MyTable_9 Left Join MyTable_6
        ON MyTable_9.Key_0 = MyTable_6.Key_1;
4)  Select MyTable_9.Key_0, MyTable_6.Key_1, MyTable_6.Data
    From MyTable_9 Right Join MyTable_6
        ON MyTable_9.Key_0 = MyTable_6.Key_1;
```

Листинг 4.36

Запросы с соединением неиндексированных таблиц

2.2. Запросы с соединением индексированных таблиц.

В этом эксперименте будет использоваться имеющаяся таблица MyTable_8 (мощность — 10 000 строк, кластеризованный уникальный индекс по полю — Key_0, некластеризованные неуникальные индексы по полям — Key_1 и Key_2) и аналогичная ей по структуре новая таблица MyTable_10 мощностью в 100 строк.

Создайте 4 запроса на базе таблиц MyTable_8 и MyTable_10 (аналогичные запросам, приведенным в п. 2.1) и проведите анализ фактических планов их выполнения.

Сформулируйте ответы на следующие вопросы.

1. В каких ситуациях наличие индексированных столбцов таблиц может привести к снижению производительности работы базы данных?

2. Какую информацию о состоянии базы данных использует генератор процедурных планов?

3. В каких типовых ситуациях «estimated plan» и «real plan» одного и того же SQL-запроса не совпадают?

Задание 3. Анализ процедурных планов выполнения SQL-запросов с группировкой строк

Выполнение задания потребует проведение эксперимента по анализу стратегии работы генератора процедурных планов выполнения SQL-запросов, содержащих операторы группировки строк Group By, операторы фильтрации групп Having и агрегатные функции (такие, например, как COUNT(), SUM() или MAX()).

Методика проведения экспериментов и средства анализа процедурных планов те же, что и при выполнении предыдущих заданий этой работы.

Предлагается самостоятельно спланировать, подготовить и провести две серии экспериментов:

– группировка строк таблицы с применением агрегатной функции;

– группировка строк таблицы с фильтрацией групп.

Серия содержит несколько опытов, каждый из которых выполняется для определенных условий:

– таблица не имеет индексов, группировка — по одному из столбцов;

– таблица не имеет кластеризованного индекса, группировка — по одному из индексированных столбцов;

– таблица имеет первичный ключ, группировка — по одному из неиндексированных столбцов;

– таблица имеет первичный ключ, группировка — по одному из индексированных столбцов.

ЧАСТЬ 5.
ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ БАЗ ДАННЫХ

ГЛАВА 15. КОНЦЕПЦИИ ЗАЩИТЫ ИНФОРМАЦИИ

Защита информации не ограничивается только рамками базы данных и даже рамками информационной системы в целом. Решение этой задачи требует рассмотрения различных аспектов информационной безопасности всего программно-аппаратного комплекса, включая использование операционных систем, сетевого оборудования, средств защиты сетевого трафика и, разумеется, средств разграничения доступа пользователей к объектам баз данных.

Требования к уровню защищенности информационной системы обеспечиваются как соответствующими проектными решениями, так и действиями персонала сопровождения в процессе ее эксплуатации, в том числе и действиями администраторов баз данных, реализующих (а в ряде случаев и определяющих) политику информационной безопасности предприятия.

Обеспечение информационной безопасности баз данных не является чисто технологической задачей, решаемой администраторами на стадии эксплуатации информационной системы. Многие проблемы с безопасностью «закладываются» в систему на стадии ее проектирования и вызваны недостаточной компетентностью разработчиков, другая категория проблем связана с действиями пользователей, непреднамеренно или сознательно нарушающих установленные правила.

При проектировании базы данных должны быть решены специфические задачи обеспечения защиты информации: классификация, группировка и определение ролевых функций субъектов доступа, разграничение прав доступа субъектов к объектам, детальное описание способов доступа к информации и определение используемых для этой цели серверных и клиентских программных приложений.

При этом следует понимать, что обеспечение жесткого режима безопасности неизбежно снижает производительность информационной системы, увеличивает трудоемкость ее администрирования, а также усложняет работу пользователей, «подталкивая» их к нарушениям установленных правил доступа к информации. Поэтому разработчики и администраторы баз данных сталкиваются с проблемой, существенно более сложной, чем техническая реализация методов защиты данных, — это проблема выбора разумного компромисса между безопасностью и доступностью информации.

Информационная безопасность базируется на трех основных концепциях защиты информации: ***целостность, доступность и конфиденциальность***.

15.1. Целостность информации

Целостность (integrity) определяется как ***способность сохранения неизменности информации в условиях ее случайного и/или преднамеренного искажения***. В теории и технологии реляционных баз данных рассматриваются как минимум четыре аспекта целостности информации, каждый из которых связан с определенной проблемой, методами и инструментальными средствами, применяемыми для ее решения.

Нормализация базы данных

В слабо нормализованной базе данных, обслуживающей OLTP-систему, ориентированную на интенсивное выполнение модифицирующих транзакций, могут проявляться аномалии включения, удаления и изменения данных. Решение этой проблемы (или сведение к минимуму ее негативных проявлений) достигается при проектировании базы данных путем ее нормализации: в хорошо нормализованной базе данных каждая таблица приведена к одной из сильных нормальных форм путем ее декомпозиции на несколько взаимосвязанных таблиц.

Заметим, что нормализация базы данных, решая задачу обеспечения целостности информации, может приводить и к негативным последствиям, снижая производительность выполнения запросов, связанных с поиском и чтением данных (вспомним процедурные планы выполнения запросов с соединением таблиц).

Ссылочная целостность

Обеспечение ссылочной целостности берет на себя сервер баз данных, контролируя соответствие типов данных и значений первичных и внешних ключей в строках связанных таблиц в процессе модификации соответствующих данных.

Единственное, что должен сделать разработчик базы данных для обеспечения ссылочной целостности, — это явно определить ключевые поля таблиц, с помощью которых будут реализованы связи между ними, используя соответствующие конструкции SQL-операторов Create/Alter Table:

FOREIGN KEY
REFERENCES <имя главной таблицы>(имя первичного ключа)

Явные ограничения целостности

Явные (*checked — проверяемые*) ограничения целостности могут быть определены для полей таблиц и отражают, по существу, семантические ограничения, накладываемые на значения атрибутов соответствующих сущностей предметной области. Простейшие ограничения этого типа — это ограничения типов данных и диапазонов допустимых значений полей, в более сложных случаях такие ограничения могут содержать логические выражения и вложенные SQL-запросы.

Серверы баз данных предоставляют несколько инструментов для работы с явными ограничениями целостности. Один из таких инструментов позволяет явно задать ограничение в процессе описания схемы таблицы SQL-операторами Create/Alter Table, используя конструкцию ADD/DROP Constraint. Сервер будет автоматически проверять выполнение ограничения при модификации данных таблицы и блокировать ввод некорректных значений. Листинг 1.3 содержит пример использования конструкции Constraint.

Другая ситуация, требующая контроля выполнения явного ограничения целостности информации, возникает в случаях, когда существует зависимость между значениями полей нескольких таблиц базы данных. Классический пример: таблица «*Студенты*» содержит список всех студентов университета,

а связанная с ней таблица «*Группы*» содержит список всех студенческих групп, причем одно из полей этой таблицы — количество студентов в группе. Очевидно, что любая из типовых операций с контингентом студентов (зачисление, отчисление или перевод) потребует внесения изменений в таблицу «*Студенты*» и, как следствие, перерасчета и изменения значения зависимого поля таблицы «*Группы*».

Одно из возможных решений проблемы — включение этих двух операций в одну транзакцию, что исключит возможность «одиночного» выполнения любой из них и, как следствие, сохранит непротиворечивость данных.

Другое решение связано с использованием *триггеров* — специальных хранимых процедур, выполнение которых обусловлено наступлением определенных событий. В нашем примере триггер, обрабатывая таблицу «*Студенты*», должен рассчитывать количество студентов в каждой группе и записывать полученные значения в соответствующие строки таблицы «*Группы*». Если связать такой триггер с событиями модификации данных в таблице «*Студенты*», то логическая целостность базы данных будет надежно обеспечена.

Физическая согласованность и надежность хранения данных

Завершая обсуждение концепции целостности информации, рассмотрим задачу сохранения физической целостности базы данных в условиях, когда в процессе эксплуатации информационной системы происходят технические сбои в работе оборудования. Обычно рассматривают две категории таких сбоев: мягкий сбой, связанный с потерей данных, накопленных в оперативной памяти и еще не сохраненных в файлах базы данных, и жесткий сбой, связанный с потерей работоспособности внешних запоминающих устройств.

Физическая целостность информации обеспечивается определенными действиями администраторов баз данных с использованием стандартных средств поддержки надежности, предоставляемых серверами баз данных.

Жесткий сбой. Практически единственным способом борьбы с жесткими сбоями является регулярное резервное копирование баз данных с последующим их восстановлением из резервных копий. Администратор базы данных может создать определенный сценарий, задающий периодичность создания резервных копий и сохраняющий их на хорошо защищенных и надежных устройствах внешней памяти. Такой сценарий будет выполняться автоматически и не потребует оперативного вмешательства администратора.

Важнейшей проблемой резервного копирования баз данных является существенная длительность этой операции, причем на время ее выполнения сервер накладывает монопольную блокировку доступа клиентских приложений ко всем физическим объектам базы данных (экстентам файлов типа Data), затрагиваемым процедурой резервного копирования.

Острота этой проблемы частично снимается применением технологии создания *разностных копий* базы данных, согласно которой копируются не все объекты базы данных, а только те из них, в которых произошли изменения с момента последнего резервного копирования базы данных. MS SQL-Server оперативно сохраняет информацию о номерах измененных экстентов в специальной файловой DCM-странице (Differential Changed Map, табл. 4.6).

Мягкий сбой. Проблема потери данных в оперативной памяти решается путем сохранения в специальном журнале (Log-файле) информации обо всех транзакциях, изменивших состояние базы данных, в том числе и о транзакциях, содержащих операции резервного копирования базы данных. Журнал транзакций — это LOG-файл специального формата, который обычно размещается на хорошо защищенном и надежном устройстве внешней памяти и содержит хронологически упорядоченную последовательность записей, каждая из которых описывает активную операцию доступа к данным с указанием идентификатора транзакции, в составе которой была выполнена эта операция.

Сервер баз данных формирует журнал транзакций в соответствии с протоколом WAL, гарантирующим актуальность состояния журнала транзакций к моменту наступления мягкого сбоя. Ниже приведено описание процесса формирования журнала транзакций и алгоритма его использования в процессе восстановлении базы данных после мягкого сбоя.

Если транзакция пытается изменить содержимое строк таблицы, удалить или вставить дополнительные строки в таблицу, то соответствующие файловые страницы считываются из файла в оперативную память и все необходимые изменения производятся в виртуальных копиях этих страниц. Параллельно информация обо всех таких изменениях оперативно регистрируется в сегменте журнала транзакций, также размещенном в буфере оперативной памяти.

В соответствии с ACID-требованиями к реализации механизма транзакций, любая транзакция должна «получить» базу данных в согласованном состоянии и сохранить ее в согласованном состоянии после своего завершения (как успешного, так и прерванного), и при этом все изменения, произведенные в оперативной памяти операциями успешно завершенной транзакции, должны быть гарантированно зафиксированы (Commit) в базе данных, а операции прерванной транзакции, в том числе и уже зафиксированные в базе данных, должны быть отменены путем отката (RollBack) транзакции.

Из этого, в частности, следует, что если транзакция успешно завершилась оператором Commit, то все виртуальные страницы, измененные операциями этой транзакции, должны быть сохранены в файле типа DATA.

Фактически все немного сложнее, так как SQL-код транзакции может содержать специальные метки — промежуточные «точки сохранения» (save points), в которых гарантируется физическая согласованность данных и при достижении которых все промежуточные результаты еще не завершенной транзакции должны быть сохранены в файле точно так же, как и при ее успешном завершении оператором Commit. При этом в журнале транзакций сохраняются и временны́е метки физической согласованности данных (так называемые TPC — Time of Physical Consistency), соответствующие промежуточным точкам сохранения транзакций.

В соответствии с протоколом WAL (Write Ahead Log — прежде запиши в журнал), любой операции записи виртуальных страниц в файл типа DATA должна предшествовать операция записи в LOG-файл соответствующего виртуального сегмента журнала транзакций. При соблюдении этого условия LOG-

файл будет содержать записи обо всех транзакциях, как завершенных, так и прерванных в момент наступления мягкого сбоя и оставивших базу данных в рассогласованном состоянии, что позволяет реализовать процедуру восстановления согласованного состояния базы данных.

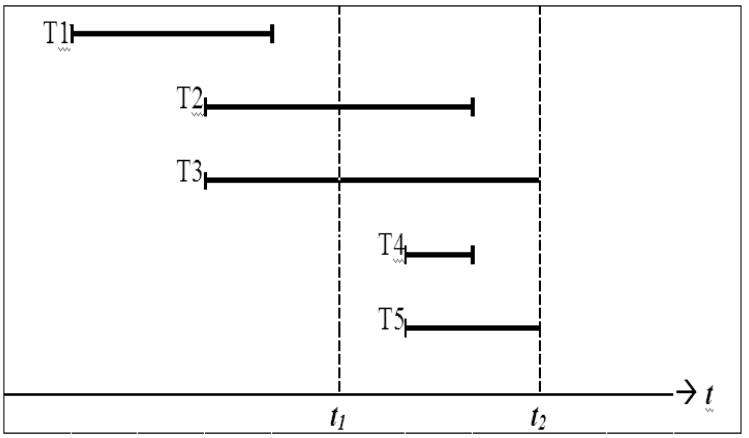


Рис. 5.1

Иллюстрация алгоритма восстановления базы данных после мягкого сбоя:

t — время фиксации записей в журнале транзакций; t_1 — последняя точка сохранения (t_{pc}); t_2 — момент наступления мягкого сбоя.

Рисунок 5.1 иллюстрирует пять типовых ситуаций, связанных с восстановлением согласованного состояния базы данных после мягкого сбоя по журналу транзакций:

– транзакция T1 успешно завершилась и была зафиксирована в журнале транзакций и в базе данных до наступления момента мягкого сбоя — никаких действий по ее восстановлению не потребуется;

– транзакция T2 успешно завершилась до момента наступления мягкого сбоя, все ее операции были зафиксированы в журнале транзакций, но часть результатов их выполнения (на рисунке — правее последней точки сохранения t_1) по каким-то причинам не была зафиксирована в базе данных до момента наступления сбоя. Для восстановления согласованности базы данных в этой ситуации потребуется повторно выполнить все операции (ReDo) этой транзакции, зарегистрированные в журнале транзакций после последней точки сохранения t_1;

– транзакция T3 не успела завершиться до момента наступления сбоя, однако в момент t_1 (точка физической согласованности) часть результатов ее выполнения была сохранена в базе данных. Несмотря на то что транзакция не нарушила согласованного состояния базы данных, потребуется выполнить откат транзакции (UnDo) путем последовательного выполнения операций, «противоположных» операциям, зафиксированным в журнале транзакций раньше последней точки сохранения t_1;

– транзакция T4 началась позднее последней точки сохранения t_1 и успешно завершилась. Все операции этой транзакции были записаны в журнал транзакций, однако результаты их выполнения по каким-то причинам не были

зафиксированы в базе данных до наступления сбоя. Данная транзакция не нарушила целостность базы данных, однако для восстановления ее результатов потребуется повторно выполнить все операции, зарегистрированные в журнале от имени этой транзакции;

– транзакция T5 началась позднее последней точки сохранения t_1 и не успела завершиться до момента сбоя t_2. Часть операций транзакции была зарегистрирована в журнале транзакций, но результаты их выполнения не были зафиксированы в базе данных. Эта транзакция не нарушила согласованного состояния базы данных, и никаких действий по ее восстановлению не потребуется.

15.2. Доступность и конфиденциальность информации

Терминология системы разграничения доступа приведена в таблице 5.1. Обеспечение доступа к информации легальным пользователям и защита конфиденциальной информации от несанкционированного доступа — две взаимосвязанные задачи, для решения которых могут использоваться различные методы и средства, определяемые требованиями к уровню защищенности системы.

Таблица 5.1

Терминология системы разграничения доступа

Термины		Определения
Доступ к информации	Access to information	Ознакомление с информацией и/или ее обработка (копирование, модификация, уничтожение)
Объекты доступа	Access objects, Securables	Единицы информации, доступ к которым регламентируется правилами разграничения доступа
Субъекты доступа	Access subjects, Principals	Пользователи, группы пользователей или процессы, действия которых регламентируются правилами разграничения доступа
Права доступа (разрешения)	Permissions	Набор операций над объектом доступа, выполнение которых разрешено субъекту доступа
Правила разграничения доступа	Security policy	Совокупность правил, регламентирующих права субъектов доступа по отношению к объектам
Идентификация	Identification	Присвоение объектам и субъектам доступа идентификатора и/или сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов
Аутентификация	Authentification	Проверка принадлежности субъекту предъявленного им идентификатора, подтверждение подлинности
Уровень полномочий субъекта	Subject privilege	Совокупность прав доступа (привилегий) субъекта доступа
Метка конфиденциальности	Sensitivity label	Информация, характеризующая уровень конфиденциальности объекта доступа
Дискреционная (логическая) защита	Discretionary secure	Разграничение доступа между поименованными субъектами и поименованными объектами
Мандатная (физическая) защита	Mandatory secure	Защита, обеспечивающая разграничение доступа субъектов с различными правами доступа к объектам различных уровней конфиденциальности

Пользователь информационной системы является важнейшим элементом системы разграничения доступа. По отношению к серверу баз данных пользователей можно разделить на три категории: конечные пользователи, прикладные программисты и администраторы.

Конечные пользователи, как правило, не имеют непосредственного доступа к серверу баз данных и получают доступ к базам данных через клиентские приложения, которые в этом случае получают статус «конечных пользователей». Права доступа конечных пользователей определяются их должностными обязанностями в информационной системе.

Прикладные программисты разрабатывают программы (клиентские и серверные приложения), использующие базы данных в интересах конечных пользователей. Прикладные программисты могут иметь права создания, модификации и выполнения программных компонентов баз данных, однако они лишены прав модификации структуры компонентов данных и схем баз данных.

Администраторы образуют особую категорию пользователей. Они наделяются правами создания баз данных и модификации их структуры, осуществляют контроль функционирования серверов баз данных и мониторинг работы конечных пользователей, в необходимых случаях оказывают консультативную помощь прикладным программистам. В обязанности администратора входит обеспечение высокой производительности работы системы, резервное копирование и восстановление работоспособности баз данных после сбоев оборудования, определение и контроль за соблюдением политики безопасности, а также регистрация пользователей и управление правами их доступа к данным.

Система управления доступом пользователей к объектам баз данных, обеспечивающая разграничение доступа к конфиденциальной информации, имеет многоуровневую архитектуру: на уровне сервера баз данных производится идентификация и аутентификация пользователей, контроль их принадлежности определенным категориям, наделение их глобальными правами выполнения определенных операций с базами данных, управляемыми этим сервером; на уровне базы данных решаются локальные задачи управления доступом со стороны индивидуальных пользователей и/или их групп.

15.3. Дискреционная защита информации

Технология дискреционного управления доступом (discretionary access control, от *discretion* — разделение, разграничение) обеспечивает так называемую *логическую* защиту данных путем *разграничения прав доступа субъектов базы данных к поименованным объектам логического уровня*.

Информация о зарегистрированных субъектах доступа — *пользователях* или *группах пользователей* (иначе называемых *ролями*), как и информация о логических объектах (таблицах, представлениях, процедурах и пр.), хранится в системном каталоге базы данных (например, MS SQL-Server хранит такую информацию в системных таблицах SysUsers и SysObjects). Наделение субъекта правом доступа к логическому объекту — это, по существу, установление связи (порядка M:N) между экземплярами (строками) соответствующих системных

таблиц, а определение типа (наименования) права доступа — это присвоение соответствующего значения атрибуту такой связи. Таким образом, право доступа также является именованным логическим объектом базы данных.

Язык SQL предоставляет функционально полный набор средств дискреционного управления доступом, состав и синтаксис которых могут несущественно отличаться в различных реализациях. Ниже приведен краткий обзор таких языковых средств.

Категории прав доступа:

– **CREATE/ALTER** — право создания/модификации объектов доступа: баз данных, таблиц, представлений, процедур, функций и др.;

– **SELECT —** право выборки (чтения) строк или отдельных столбцов таблицы или представления;

– **INSERT —** право вставлять в таблицу или представление новые строки;

– **UPDATE —** право изменять данные в таблице или ее отдельных столбцах;

– **DELETE —** право удалять строки из таблицы или представления;

– **REFERENCES —** право ссылаться на указанный объект, которое разрешает пользователю создавать внешние ключи в подчиненных таблицах без права доступа к главным таблицам;

– **EXECUTE —** право выполнения хранимых процедур и функций.

SQL-операторы управления правами доступа:

– **CREATE USER … , CREATE ROLE …** — создание субъектов доступа соответствующих типов;

– **GRANT** <привилегия> **ON** <объект> **TO** <субъект> [**WITH GRANT OPTION**] – разрешение доступа: оператор предоставляет субъекту указанные права доступа («привилегии») к объекту (опционально — с правом передачи полученных прав другим субъектам);

– **DENY** <привилегия> **ON** <объект> **TO** <субъект> — запрет доступа: оператор запрещает субъекту доступ к объекту с указанными правами («привилегиями»);

– **REVOKE** <привилегия> **ON** <объект> **TO** <субъект> — оператор отменяет действие выполненных ранее операторов **GRANT** или **DENY**.

Дополнительно к перечисленным выше SQL-средствам прямого управления доступом серверы баз данных предоставляют администраторам программные компоненты, а также неязыковые средства экранного интерфейса.

Дискреционная защита информации удовлетворяет потребностям большинства коммерческих приложений, однако для систем повышенного уровня защищенности ее возможностей оказывается явно недостаточно.

Завершая обзор методов дискреционного управления доступом, перечислим основные проблемы защиты конфиденциальной информации, которые либо не решаются, либо решаются лишь частично в рамках этой технологии.

Во-первых, дискреционная защита не позволяет надежно разграничить доступ *к различным строкам* одной таблицы. Частичное решение проблемы — запрет доступа к таблице и разрешение доступа к отдельным представлениям,

созданным на базе этой таблицы и содержащим различные условия выборки строк. Слабость такого решения очевидна: невозможно разграничить доступ к нескольким строкам таблицы, удовлетворяющим одному условию выборки.

Во-вторых, разграничение доступа *к различным столбцам* одной таблицы также создает определенные проблемы. Защита, основанная на создании представления, не содержащего информации «секретных» столбцов таблицы, легко обходится средствами SQL. Более радикальное решение предполагает модификацию схемы базы данных: защищаемые столбцы таблицы выделяются в отдельную таблицу, связанную с исходной таблицей, в которой остается только общедоступная информация. Такое решение может оказаться достаточно надежным, однако оно приведет к снижению производительности системы (еще раз вспомним процедурные планы выполнения SQL-запросов с соединением таблиц).

Третий (возможно, главный) недостаток дискреционного управления доступом — это отсутствие средств защиты от несанкционированного *распространения* конфиденциальной информации: ничто не препятствует пользователю, легально получившему доступ (с правом чтения *Select*) к таблице с конфиденциальной информацией, сделать эту информацию доступной другим пользователям путем вставки (Insert) этой информации в другую (например, временную) общедоступную таблицу.

И последнее: дискреционная защита не позволяет физически изолировать технического специалиста, выполняющего функции администратора базы данных, от управления конфиденциальными данными (в том числе и от ознакомления с ними), что требует повышения меры ответственности администратора и вряд ли соответствует политике информационной безопасности, реализуемой в хорошо защищенной системе.

Как видим, дискреционная логическая защита является довольно слабой, и основная причина такой «слабости» заключается в том, что информация о защите данных хранится отдельно от самих защищаемых данных. Существенно более высокий уровень защищенности информации обеспечивает так называемая мандатная, или физическая защита.

15.4. Мандатная защита информации

Мандатное управление доступом (mandatory access control) основано на использовании «меток безопасности», присваиваемых экземплярам защищаемых объектов базы данных, и официальном «мандате», выдаваемом субъекту и разрешающем ему доступ к информации с определенными метками безопасности.

Классическая модель системы мандатного управления доступом впервые была описана Дэвидом Беллом и Леонардом Лападулой в 1975 г. Согласно этой модели каждому субъекту и каждому объекту доступа присваивается метка конфиденциальности: от самой высокой («особой важности») до самой низкой («несекретный» или «общедоступный»).

При этом субъект не может читать информацию объекта, метка конфиденциальности которого выше его собственной, но также ему запрещена запись информации в объекты с более низким уровнем безопасности, что не позволит такому субъекту понизить уровень секретности информации, к которой он получил легальный доступ.

На рисунке 5.2[9] показана диаграмма информационных потоков модели Белла — Лападулы.

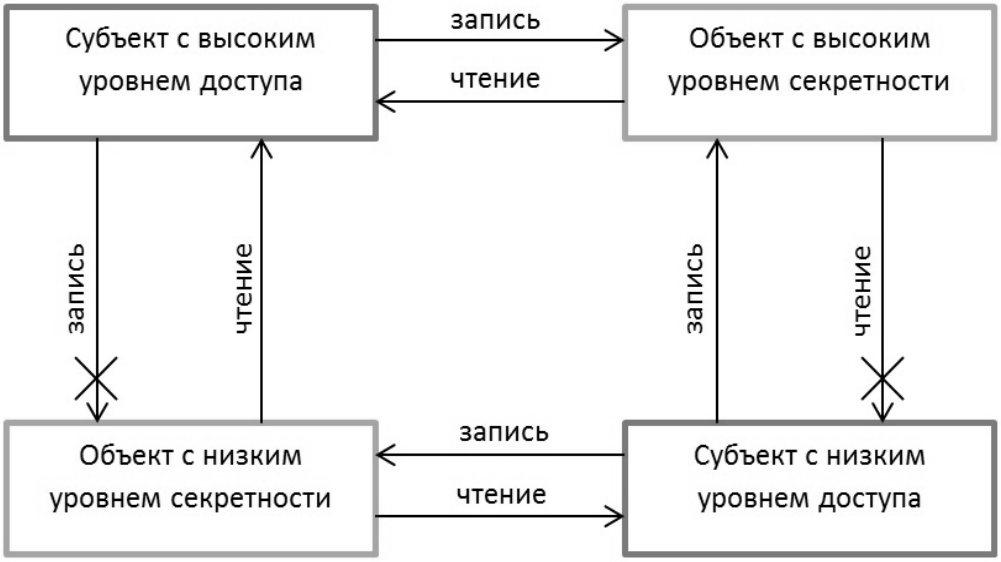


Рис. 5.2
Диаграмма информационных потоков модели мандатной защиты информации

Метка безопасности объекта (всей таблицы, отдельного ее столбца, строки или поля) может иметь сложную структуру, позволяющую определить группу принадлежности объекта, уровень его ценности и конфиденциальности, а также ряд других параметров, например уровень защищенности устройства, на котором хранится объект. Метка безопасности объекта неотделима от самого объекта, физически хранится вместе с ним (а не в системном каталоге, как это происходит при логической защите) и уничтожается только в момент уничтожения объекта.

Мандатная защита обеспечивается специализированными серверами баз данных, которые блокируют «обход» меток безопасности при получении доступа к информации и, кроме того, предоставляют средства аудита доступа субъектов к защищаемым объектам.

В качестве примера рассмотрим систему мандатного управления доступом, реализованную сервером баз данных «Линтер», имеющим сертификат по второму классу защиты от несанкционированного доступа, что соответствует классу B3 американского национального стандарта.

[9] Автор рисунка — М. Савельев. URL: https://ru.wikipedia.org/w/index.php?curid=2894287.

Группы принадлежности

Субъекты доступа группируются (например, по отделам организации), при этом могут устанавливаться «доверительные отношения» между различными группами принадлежности субъектов. Объекты доступа, независимо от иерархии в базе данных, также распределяются по группам принадлежности, причем объект может принадлежать только одной такой группе. Группы принадлежности субъектов связываются с группами принадлежности объектов, при этом субъект получает права доступа только к объектам групп, связанных с группой этого субъекта, а также с группами, находящимися в доверительных отношениях с его группой.

Уровни доступа

Каждому ***объекту*** присваивается определенный *уровень ценности* (WAL — Write Access Level), ограничивающий возможность его модификации, и *уровень конфиденциальности* (RAL — Read Access Level), ограничивающий возможность просмотра (чтения) этого объекта. WAL- и RAL-уровни присваиваются также и ***субъектам*** доступа. По умолчанию объект наследует уровни того субъекта, который создал или изменил этот объект.

Метки безопасности

Метка безопасности *объекта доступа* включает компоненты:

– группа принадлежности субъекта, создавшего объект;

– RAL-уровень объекта;

– WAL-уровень объекта.

Метка безопасности *субъекта доступа* включает компоненты:

– группа принадлежности субъекта;

– RAL-уровень субъекта, равный максимальному RAL-уровню объекта, доступному для прочтения этим субъектом;

– WAL-уровень субъекта, равный минимальному RAL-уровню объекта, доступному для модификации этим субъектом.

Субъект не получит права чтения объекта доступа, RAL-уровень которого выше его собственного RAL-уровня, тем самым конфиденциальная информация будет защищена от несанкционированного прочтения. WAL-уровень субъекта доступа ограничивает его возможности по понижению уровня конфиденциальности объекта: субъект не получит права модификации (изменения, удаления или вставки) объекта доступа, RAL-уровень которого выше WAL-уровня самого этого субъекта. Иными словами, пользователь не сможет сделать доступную ему информацию менее конфиденциальной, чем задано ее RAL-уровнем.

ГЛАВА 16. УПРАВЛЕНИЕ ДОСТУПОМ К ДАННЫМ

16.1. Двухуровневая архитектура управления доступом

MS SQL-Server поддерживает двухуровневую архитектуру системы управления доступом к данным (рис. 5.3).

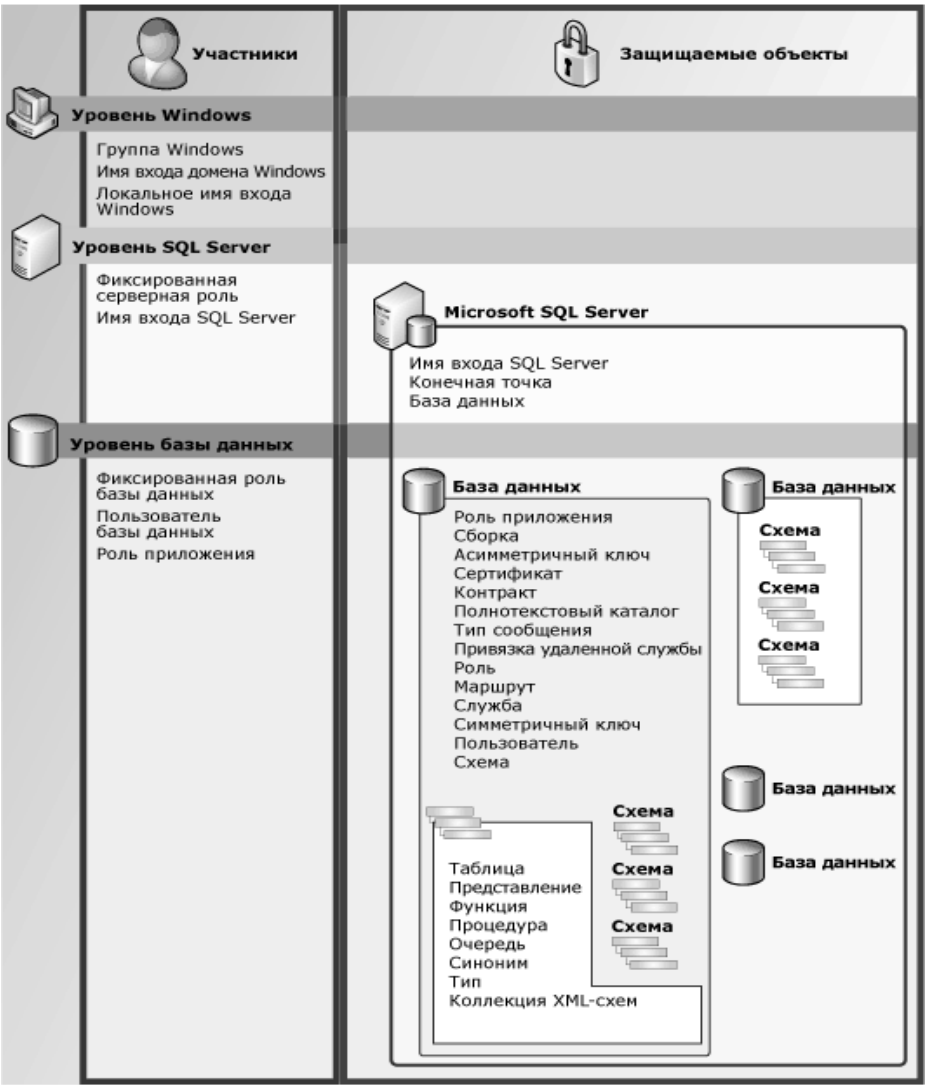


Рис. 5.3
Иерархия субъектов и объектов доступа MS SQL-Server

На верхнем уровне (уровень сервера) решаются задачи аутентификации пользователей и выдача им глобальных прав выполнения операций с базами данных, управляемыми этим сервером.

Объектами доступа на этом уровне являются пользовательские базы данных и типовые операции над ними.

В качестве субъектов доступа используются *учетные записи пользователей* (имена входа, login) и *фиксированные роли сервера*, за каждой из которых закреплено *имя роли* и определенный набор *разрешений* на выполнение операций с базами данных. Учетная запись — это средство идентификации и аутентификации пользователя, а роль сервера — средство группировки учетных записей для более удобного администрирования. Имеются два способа предоставления пользователю разрешений на выполнение глобальных операций с базами данных: индивидуальное назначение требуемых разрешений учетной записи пользователя или включение учетной записи в качестве члена в соответствующую серверную роль.

На нижнем уровне (уровень базы данных, доступ к которой получили пользователи через соответствующие учетные записи), решаются задачи разграничения прав их доступа к логическим объектам этой базы данных, а также задача шифрования данных объектов доступа.

Основные *субъекты доступа* нижнего уровня — это *пользователи базы данных* и *роли базы данных*, используемые для группового управления правами доступа пользователей к объектам базы данных. При этом *пользователь базы данных* всегда связан с определенной учетной записью, «предъявляемой» им на этапе аутентификации.

На уровне базы данных поддерживаются три категории ролей:

– предопределенный набор *фиксированных ролей базы данных*, через членство в которых пользователи получают соответствующие права доступа ко всем объектам базы;

– формируемые администратором *пользовательские роли*, члены которых могут получать необходимые права доступа к любым логическим объектам базы данных;

– *роли приложений*, членство пользователей в которых не предусмотрено, а права доступа, назначенные этой ролью, предоставляются клиентскому приложению, через которое пользователь подключается к базе данных.

Состав, имена и права доступа фиксированных ролей базы данных ко всем ее объектам также фиксированы, у администратора остается право управления членством пользователей в ролях этого типа. Пользовательские роли базы данных администратор вправе создавать, именовать и назначать им права доступа к логическим объектам базы по своему усмотрению (например, в соответствии с организационной структурой предприятия и должностными функциями сотрудников).

Основные объекты доступа на уровне базы данных — это схемы, таблицы и представления, отдельные столбцы таблиц и представлений, а также программные компоненты базы данных — хранимые процедуры и функции, которые, заметим, одновременно являются и субъектами доступа и могут наделяться соответствующими правами.

16.2. Управление доступом на уровне сервера баз данных

16.2.1. Режимы аутентификации

Аутентификация (проверка подлинности пользователя) — это первый рубеж, успешное преодоление которого дает право пользователю на соединение с сервером баз данных. MS SQL-Server реализует традиционный способ аутентификации пользователей — путем сравнения введенного пользователем пароля с «правильным» паролем, соответствующим имени учетной записи этого пользователя. При этом поддерживаются два режима аутентификации — «Аутентификация Windows» и «Аутентификация SQL-Server».

В процессе установки сервера баз данных по умолчанию назначается первый режим аутентификации, при котором пользователи, успешно прошедшие проверку подлинности при входе в операционную систему, автоматически получают право соединения с сервером. Такое решение позволяет использовать средства безопасности, предоставляемые операционной системой (как правило, более надежные по сравнению с соответствующими средствами, реализуемыми на уровне приложений), и, что также немаловажно, упрощает работу пользователей и снижает риск небезопасного хранения паролей, так как пользователю требуется помнить единый пароль доступа к различным приложениям Windows.

Второй режим аутентификации (называемый также «смешанным») предполагает дополнительную регистрацию пользователей средствами сервера баз данных, выдачу им имен учетных записей и паролей с последующей проверкой подлинности пользователей при их соединении с сервером.

Такой режим аутентификации предназначен в основном для клиентских приложений, функционирующих на платформах, отличных от Windows. Аутентификация средствами программного приложения считается менее безопасной, однако MS SQL-Server поддерживает шифрование трафика между клиентом и сервером, в том числе с помощью сертификатов, сгенерированных самим сервером.

16.2.2. Учетные записи и разрешения уровня сервера

MS SQL-Server использует два типа учетных записей: учетные записи Windows (*Windows Domain login* и *Windows Local login*) и учетные записи сервера (*SQL-Server login*), используемые в случае выбора смешанного режима аутентификации.

При подключении к серверу от имени учетной записи Windows и при создании учетной записи сервера имя и идентификатор учетной записи сохраняются в таблице SysLogins базы данных Master (табл. 5.3). Для учетных записей сервера в этой таблице дополнительно сохраняется пароль.

Для создания учетной записи сервера можно воспользоваться SQL-командой CREATE LOGIN, например:

```
CREATE LOGIN NewUser WITH PASSWORD = 'NewP@sssW000rd'
```

Графический интерфейс SQL-Server Management Studio содержит средства создания и удаления учетных записей сервера, а также средства просмотра и редактирования выданных им разрешений, доступные через вкладку «*Свойства»* учетной записи и вкладку «*Разрешения»* свойств сервера.

На рисунках 5.4 и 5.5 приведены примеры соответствующих экранных форм.

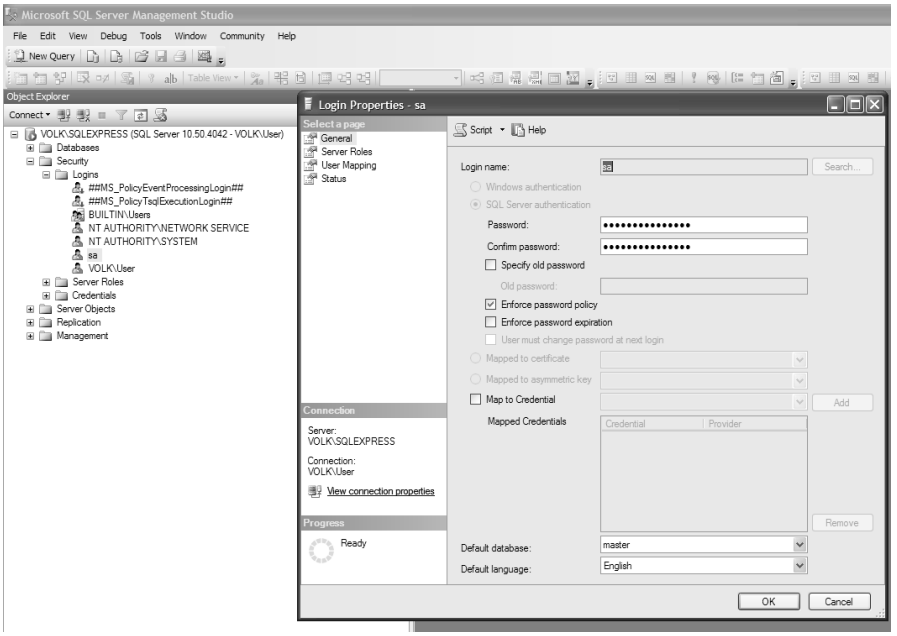


Рис. 5.4
Средства просмотра свойств учетных записей

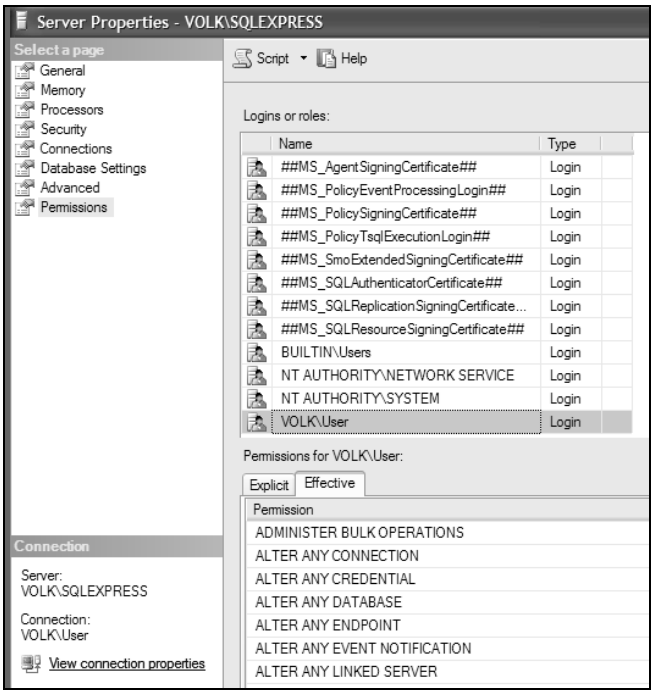


Рис. 5.5
Средства просмотра и редактирования разрешений

На вкладке **Logins** отображается набор создаваемых автоматически (так называемых встроенных) учетных записей Windows и учетная запись **sa** (*System Administrator*) — единственная учетная запись сервера, создаваемая по умолчанию. Эта учетная запись обладает правами системного администратора сервера, но использовать ее можно только в случае, если выбран режим аутентификации средствами SQL-Server.

16.2.3. Фиксированные роли сервера

Для предоставления пользователю фиксированного набора разрешений на выполнение работ по администрированию сервера учетную запись этого пользователя следует включить в соответствующую серверную роль. Состав таких ролей, их наименования и связанный с каждой ролью набор административных задач строго фиксированы (табл. 5.2), при этом ни один из администраторов сервера не может ни создавать новые серверные роли, ни удалять существующие роли, ни изменять их свойства.

Члены любой фиксированной роли сервера могут добавлять ***в эту роль*** любую учетную запись. Члены роли sysadmin могут добавлять учетные записи ***в любую фиксированную роль*** сервера.

Таблица 5.2

Фиксированные роли сервера

Имя роли	Назначение роли и права членов роли
sysadmin	Роль для системного администратора сервера баз данных. Полные права на SQL-Server, в том числе и право передачи этих прав другим пользователям
serveradmin	Роль для оператора, обслуживающего сервер с правами отключения сервера и изменения любых его настроек. Запрещен доступ к данным, отсутствует право изменения разрешений на доступ к объектам
dbcreator	Члены роли получают право создания баз данных (и автоматически становятся ее владельцами с полными правами в созданной базе данных)
securityadmin	Роль для администратора безопасности с правами управления «обычными» пользователями: создание учетных записей, изменение паролей, предоставление прав доступа к данным. Запрещено любое вмешательство в работу сервера, изменение административных прав и учетных записей администраторов
setupadmin	Члены роли получают право подключения внешних (так называемых связанных) серверов баз данных
processadmin	Члены роли получают единственное право: право закрытия пользовательских подключений к серверу (например, зависших)
diskadmin	Члены роли получают право выполнять любые операции с файлами баз данных
bulkadmin	Роль для сотрудников, которые выполняют специфическую операцию — массовую загрузку данных в таблицы

Есть еще одна серверная роль — это роль PUBLIC, членами которой автоматически становятся все пользователи, подключившиеся к серверу, причем лишить пользователя членства в этой роли нельзя. Роль PUBLIC используется для предоставления разрешений всем пользователям сервера. По умолчанию у этой роли есть только два разрешения: разрешение «CONNECT» на соединение

с сервером и разрешение «VIEW ANY DATABASE» на просмотр списка баз данных, управляемых сервером.

Роль PUBLIC занимает особое место в составе серверных ролей:

– во-первых, отсутствуют средства управления членством учетных записей в этой роли (в этом нет необходимости, так как членами роли автоматически становятся все учетные записи, получившие доступ к серверу);

– во-вторых, эта роль не отображается в списке серверных ролей, возвращаемых хранимой процедурой sp_helpsrvrole (табл. 5.2), однако она присутствует на вкладке \Безопасность\Роли_сервера обозревателя объектов SQL-Server Management Studio;

– в-третьих, роль PUBLIC, строго говоря, не является «фиксированной» — в отличие от других ролей сервера, в окне свойств этой роли можно изменить ее текущие разрешения.

Управление фиксированными серверными ролями реализуется соответствующими системными хранимыми процедурами, состав и форматы обращения к которым приведены в таблице 5.3.

Таблица 5.3

Средства управления фиксированными серверными ролями

Хранимая процедура	Входные параметры	Результат
sp_helpsrvrole	-	Список серверных ролей
sp_helpsrvrolemember	['role']	Список членов роли 'role'. Если входной параметр опущен, процедура возвращает список членов всех серверных ролей
sp_addsrvrolemember	'login', 'role'	Добавление учетной записи 'login' в роль 'role'
sp_dropsrvrolemember		Удаление учетной записи 'login' из роли 'role'
sp_srvrolepermission	['role']	Список разрешений указанной роли. Если входной параметр опущен, процедура возвращает список разрешений для всех серверных ролей

16.2.4. Хранение информации об учетных записях

Для хранения информации об учетных записях пользователей и их членстве в фиксированных серверных ролях используется системная таблица SysLogins системной базы данных Master. Каждая строка этой таблицы содержит информацию об одной учетной записи (табл. 5.4).

Таблица 5.4

Схема системной таблицы SysLogins

Имя поля	Тип данных	Описание
sid	varbinary(85)	Security Identifier — уникальный идентификатор безопасности учетной записи
createdate	datetime	Дата добавления учетной записи
updatedate	datetime	Дата обновления учетной записи
name	sysname	Имя учетной записи
dbname	sysname	Имя базы данных по умолчанию для пользователя при установлении соединения

Продолжение табл. 5.4

Имя поля	Тип данных	Описание
password	nvarchar(128)	Хешированное значение пароля (= NULL для учетных записей Windows)
language	sysname	Язык по умолчанию для пользователя
denylogin	int	= 1 — учетной записи Windows отказано в доступе к серверу
hasaccess	int	= 1 — учетной записи Windows разрешен доступ к серверу
isntname	int	= 1 — для учетных записей Windows; = 0 — для учетных записей SQL-Server
isntgroup	int	=1 — для учетных записей групп Windows
isntuser	int	=1 — для учетных записей пользователей Windows
sysadmin	int	= 1 — если учетная запись является членом одноименной серверной роли
securityadmin	int	
serveradmin	int	
setupadmin	int	
processadmin	int	
diskadmin	int	
dbcreator	int	
bulkadmin	int	

16.3. Управление доступом на уровне базы данных

16.3.1. Объекты доступа: таблицы, представления, команды и схемы

Пользователь, прошедший процедуру аутентификации и получивший через свою учетную запись доступ к серверу с соответствующими глобальными разрешениями на выполнение операций с базами данных, должен получить у администратора требуемый ему набор прав доступа к логическим объектам базы данных.

Объекты доступа базы данных — это *таблицы*, *представления*, *хранимые процедуры* и *функции*, а также *SQL-команды*, предназначенные для создания логических объектов (CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE FUNCTION, CREATE RULE, CREATE DEFAULT) и резервных копий (BACKUP DATABASE, BACKUP LOG) базы данных, и *схемы*, используемые для группировки объектов.

Схемы

Схема — это именованный объект-контейнер, предназначенный для группировки логических объектов БД с целью упрощения процесса управления правами доступа к ним со стороны пользователей.

При создании БД в ней автоматически создаются схемы, владельцами которых являются фиксированные роли и специальные (встроенные) пользователи БД. Этим схемам присваиваются имена их владельцев.

Объект БД (таблица или представление) не может одновременно входить в состав нескольких схем, при этом допускается перемещение объектов между схемами одной базы данных (ALTER SCHEMA).

Схема может быть удалена (DROP SCHEMA) при условии, что она пуста. Перед удалением схемы все ее объекты должны быть либо удалены, либо перемещены в другие схемы базы данных.

При создании схемы (CREATE SCHEMA) должен быть задан ее владелец — субъект доступа любого типа, получающий соответствующие права доступа ко всем объектам, включенным в схему.

16.3.2. Субъекты доступа: пользователи и роли базы данных

Права на выполнение операций с логическими объектами БД предоставляются именованным *субъектам доступа — пользователям* и *ролям* базы данных, а также *ролям приложений*.

Пользователи базы данных

Основным субъектом доступа на уровне базы данных является пользователь, которому могут быть индивидуально разрешены права доступа к объектам базы данных. При создании пользователя базы данных сервер связывает его с определенной учетной записью (субъектом доступа уровня сервера), и таким образом владелец учетной записи получает соответствующие права доступа к объектам базы данных. При этом учетная запись не может быть связана более чем с одним пользователем одной базы данных.

При создании базы данных в ней автоматически создаются два специальных пользователя: ***dbo*** и ***guest***. Пользователь ***dbo*** (database owner) является *владельцем* базы данных и имеет в этой базе абсолютные права, в том числе и право наделения других пользователей правами владельца базы данных. С пользователем ***dbo*** связана та учетная запись, которой на уровне сервера было выдано разрешение CREATE DATABASE и владелец которой воспользовался этим разрешением и создал базу данных. Пользователь ***dbo*** автоматически становится членом фиксированной роли базы данных ***db_owner***.

Если учетной записи явно не предоставлен доступ к базе данных, то она автоматически отображается сервером в пользователя ***guest***, следовательно, права доступа к объектам базы данных, назначенные пользователю guest, автоматически получают все учетные записи сервера. Для снижения рисков нарушения безопасности рекомендуется удалять пользователя guest из базы данных.

Роли базы данных

Для группировки пользователей базы данных используются *роли*, которые сами являются именованными субъектами доступа, то есть для ролей, как и для пользователей, могут быть определены права доступа к логическим объектам базы данных. При этом права роли автоматически получают все ее члены. Роль всегда имеет *владельца*, в качестве которого может выступать пользователь или другая роль базы данных, и *набор схем*, принадлежащих этой роли.

Фиксированные роли базы данных

В каждой базе данных всегда присутствует предопределенный набор фиксированных ролей (табл. 5.5), с каждой из которых связано определенное множество глобальных разрешений — прав доступа ко всем объектам базы данных.

Таблица 5.5

Фиксированные роли базы данных

Роль	Права членов роли
db_owner	Права владельца, выполнение любых действий с базой данных
db_securityadmin	Управление правами доступа к объектам базы данных других пользователей и членством их в ролях
db_accessadmin	Управление пользователями базы данных: создание, удаление и изменение
db_ddladmin	Создание, изменение и удаление объектов базы данных
db_datawriter	Разрешено изменение/просмотр данных любых таблиц или представлений базы данных
db_datareader	
db_denydatawriter	Запрещено изменение/просмотр данных любых таблиц или представлений базы данных независимо от выданных разрешений
db_denydatareader	
db_backupoperator	Резервное копирование базы данных
public	Любой пользователь, созданный в базе данных, автоматически включается в роль public и не может быть удален из нее. Роль предназначена для предоставления *прав доступа по умолчанию* (default right) всем пользователям базы данных

Роли этого типа предназначены в основном для пользователей, выполняющих функции администрирования базы данных.

Владельцем всех фиксированных ролей базы данных является пользователь ***dbo***, а членами фиксированной роли могут быть только пользователи базы данных.

Пользовательские роли базы данных

В отличие от фиксированных ролей, *пользовательские роли* формируются администратором и предназначены для группировки пользователей, которым следует назначить одинаковые права доступа к логическим объектам базы данных. Набор объектов, связанных с пользовательской ролью, и права доступа к каждому из них не являются предопределенными и могут быть сформированы индивидуально для каждой такой роли.

Другим существенным отличием пользовательских ролей от фиксированных является возможность членства пользовательской роли в другой пользовательской роли с соответствующим наследованием дочерними ролями прав доступа к объектам, установленным для родительских ролей.

Роли приложений

Имеется еще одна категория ролей уровня базы данных — это *роли приложений*. Членство пользователей в ролях приложений не предусмотрено — роли этого типа предназначены для предоставления прав доступа клиентским приложениям, через которые пользователи подключаются к базе данных. Отличительной особенностью ролей приложений является наличие у каждой из них специального пароля, который должен быть отправлен приложением серверу для получения прав доступа, назначенных соответствующей роли.

16.3.3. Хранение информации о субъектах доступа

Для хранения информации о пользователях и ролях базы данных всех трех перечисленных выше типов используется системная таблица `SysUsers` пользовательской базы данных (табл. 5.6).

Каждая строка этой таблицы представляет один субъект доступа — пользователя, фиксированную или пользовательскую роль базы данных или роль приложения.

Таблица 5.6

Схема системной таблицы SysUsers

Имя столбца	Тип данных	Описание
uid	smallint	Идентификатор субъекта доступа (пользователя или роли), уникальный в пределах базы данных. uid = 1 — для пользователя ***dbo*** uid = 2 — для пользователя ***guest*** 3 ≤ uid < 16384 — для прочих субъектов доступа uid ≥ 16384 — для фиксированных ролей
name	sysname	Имя субъекта доступа
sid	Varbinary (85)	Идентификатор безопасности учетной записи пользователя или владельца роли
roles	Varbinary (2048)	Битовая строка, определяющая членство субъекта доступа в ролях сервера. Если roles[uid] = 1, то субъект доступа является членом роли, идентификатор которой равен uid (в новых версиях не используется)
createdate	datetime	Дата создания субъекта доступа
updatedate	datetime	Дата последнего изменения субъекта доступа
password	varbinary (256)	Пароль для ролей приложения. = null — для других типов субъектов доступа
gid	smallint	Идентификатор роли (в старых версиях сервера — группы, откуда идет название поля): если gid = uid, то субъект доступа — фиксированная или пользовательская роль базы данных; для других типов субъектов доступа gid = 0
hasdbaccess	int	Если = 1, то пользователь имеет доступ к базе данных
islogin	int	Если = 1, то субъект доступа соответствует учетной записи любого типа
isntname	int	Если = 1, то пользователь является отображением учетной записи Windows
isntgroup	int	Если = 1, то пользователь является отображением учетной записи группы Windows
isntuser	int	Если = 1, то пользователь является отображением учетной записи пользователя Windows
issqluser	int	Если содержит 1, то пользователь является отображением учетной записи пользователя SQL-Server
isaliased	int	Если = 1, то пользователь является псевдонимом другого пользователя
issqlrole	int	Если = 1, то субъект доступа — пользовательская или фиксированная роль базы данных
isapprole	int	Если = 1, то субъект доступа — роль приложения

16.3.4. Средства управления пользователями и ролями

Команда CREATE USER создает пользователя базы данных, сопоставляя его с учетной записью (сертификатом, ассиметричным ключом) или схемой по умолчанию.

```
CREATE USER user_name
    [ FOR
      {
        LOGIN login_name
        | CERTIFICATE cert_name
        | ASYMMETRIC KEY asym_key_name
      }
      | WITHOUT  LOGIN
    ]
    [ WITH DEFAULT_SCHEMA = schema_name ]
```

Листинг 5.1
Формат SQL-команды CREATE USER

Параметр WITHOUT LOGIN создает пользователя, который не сопоставляется с учетной записью, такой пользователь может подключиться к базе данных как guest. Если параметр DEFAULT_SCHEMA не задан, пользователю в качестве схемы по умолчанию будет назначена схема dbo.

Команда ALTER USER позволяет переименовать пользователя базы данных, сопоставить его с новой учетной записью или определить для него новую схему по умолчанию.

```
ALTER USER userName
 WITH
{
   NAME = newUserName
  | DEFAULT_SCHEMA = schemaName
  | LOGIN = loginName
 }
 [ ,...n ]
```

Листинг 5.2
Формат SQL-команды ALTER USER

Команда DROP USER userName удаляет пользователя базы данных.

Команда CREATE ROLE *role_name* [AUTHORIZATION *owner_name*] создает роль базы данных *role_name*, владельцем которой назначается *owner_name*. Если параметр AUTHORIZATION не задан, владельцем созданной роли назначается субъект, выполнивший команду CREATE ROLE.

Команда ALTER ROLE позволяет переименовать роль, включить в нее новых членов или удалить членов из роли.

```
ALTER ROLE role_name
{
       [ADD   MEMBER database_principal]
     | [DROP MEMBER database_principal]
     | WITH NAME = new_role_name
}
```

Листинг 5.3
Формат SQL-команды ALTER ROLE

16.3.5. Средства управления правами доступа

Перечень типов прав доступа к объектам базы данных приведен в разделе 15.3 учебника. Каждый из этих типов может быть отнесен к одной из следующих категорий:

– права доступа к данным таблиц и представлений базы данных;
– права на выполнение хранимых процедур и функций;
– права на выполнение команд Transact-SQL.

При создании нового пользователя базы данных он автоматически становится членом роли PUBLIC и наделяется правами доступа, установленными для этой роли.

Субъекту доступа (пользователю или пользовательской роли базы данных) можно предоставить (или отнять) права *неявно* — через членство в фиксированных или пользовательских ролях базы данных, или *явно*, например SQL-операторами Grant, Deny или Revoke.

`Grant` — предоставление доступа

Для предоставления субъектам прав доступа к таблицам базы данных, а также к представлениям, процедурам и функциям используется SQL-оператор GRANT, имеющий синтаксис, представленный в листинге 5.4*а*.

```
GRANT
{ ALL | permission [,...n] }
ON table | view [( column [,...n] )]
| ON procedure | ON function
TO security_account [,...n]
[WITH GRANT OPTION]
[AS (role)]
```

а

```
GRANT
{ALL | statement[,...n]}
TO security_account [, ... .n]
```

б

Листинг 5.4
Формат SQL-команды GRANT:
а — предоставление прав доступа к объектам;
б — предоставление прав выполнения SQL-команд.

Параметры команды GRANT:

– ALL — позволяет предоставить все права доступа к объекту (актуальные для типа объекта, указанного в разделе ON);

– permission [,...n] — вид предоставляемого права доступа (SELECT, INSERT, DELETE, UPDATE, REFERENCES или EXECUTE), при этом одной командой может быть предоставлено несколько видов прав;

– ON table | view — имя объекта доступа (таблицы или представления);

– [(column [,...n])] — имена столбцов таблицы или представления, если задан этот необязательный параметр, то права (в этом случае только SELECT или UPDATE) будут предоставлены только указанным столбцам; предоставление прав на уровне таблицы или представления отменяет все права, ранее предоставленные на уровне столбцов;

– ON procedure — имя стандартной или расширенной хранимой процедуры (разрешается выдавать только право EXECUTE);

– ON function — имя пользовательской функции (для скалярных функций разрешается выдавать права EXECUTE или REFERENCES, для TVF-функций, возвращающих табличные значения, дополнительно разрешается выдавать права SELECT, INSERT и DELETE);

– TO security_account [,...n] — список субъектов доступа (пользователей и/или ролей базы данных), которым предоставляются права;

– [WITH GRANT OPTION] — при наличии этого параметра субъекты доступа, получившие указанные в операторе права, получают право предоставления этих прав другим субъектам доступа (за исключением прав доступа к отдельным столбцам таблиц и представлений);

– [AS (role)] — использование этого необязательного параметра дает возможность пользователю, не имеющему явно выданных ему прав на предоставление прав другим субъектам доступа, выдавать права доступа от имени своей «родительской» роли, имеющей такие права.

Право выполнения SQL-операторов CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE FUNCTION, CREATE RULE, CREATE DEFAULT, BACKUP DATABASE, BACKUP LOG также предоставляется командой GRANT (листинг 5.4*б*), в которой параметр statement [,...n] задает список SQL-операторов из приведенного выше перечня.

Deny — отклонение (запрет) доступа

Получив от администратора набор прав доступа, пользователь сможет выполнять разрешенные ему действия. Но в некоторых случаях бывает необходимо наложить запрет на выполнение им определенных операций с определенными объектами базы данных.

Глобальный запрет доступа субъекта ко всем объектам базы данных может быть реализован путем включения этого субъекта в фиксированные роли базы данных db_denydatareader (глобальный запрет чтения данных) или db_denydatawriter (глобальный запрет модификации данных).

Если же требуется запретить субъекту доступ к отдельным объектам базы данных, сохранив при этом предоставленные ему ранее права доступа к другим объектам, администратор может воспользоваться SQL-оператором DENY, синтаксис которого приведен в листинге 5.5*а*.

```
DENY
{ ALL | permission [,…n] }
ON table | view [( column [,…n])]
| ON procedure | ON function
TO security_account [,…n]
[CASCADE]
```

а

```
DENY
{ALL | statement[,...n]}
TO security_account [,... .n]
```

б

Листинг 5.5
Формат SQL-команды DENY:
а — запрет доступа к объектам базы данных;
б — запрет выполнения SQL-команд.

Эта же команда позволяет запретить субъекту право выполнения отдельных SQL-операторов, сохранив предоставленные ему ранее права (листинг 5.5*б*).

Параметры команды DENY аналогичны соответствующим параметрам команды GRANT. Необязательный параметр [CASCADE] предписывает *каскадирование* запрета доступа, указанного в команде DENY: если пользователь, которому эта команда запрещает доступ к объекту, ранее предоставлял к нему доступ другим пользователям, то команда запретит доступ к этому объекту и этим пользователям, а также всем их «потомкам» всех уровней.

Если запрет доступа требуется распространить на группу из нескольких пользователей, будет целесообразно создать специальную пользовательскую роль, SQL-оператором DENY запретить ей доступ к соответствующим объектам и затем включить в эту роль всю группу пользователей.

Согласно базовому правилу функционирования системы ограничения доступа, *запрет доступа* (Deny) имеет более высокий приоритет, чем его предоставление (Grant).

Если администратор запрещает пользователю (или пользовательской роли) доступ к объекту (явно или через членство в соответствующей роли), а ранее такой доступ был ему предоставлен, то система безопасности гарантирует отсутствие прав доступа до момента отмены (Revoke) этого запрета или исключения пользователя из числа членов «запрещающей» роли.

REVOKE — отмена явно предоставленных прав и запретов доступа.

Если пользователю был предоставлен доступ к объекту через членство в роли базы данных, но при этом ему был явно запрещен (DENY) доступ к этому же объекту, то запрет как более приоритетный по сравнению с разрешением будет действовать и пользователь будет лишен соответствующего доступа. В такой ситуации явный запрет может быть снят оператором REVOKE, после чего пользователь вновь получит доступ к объекту как член роли.

Оператор REVOKE может также отменить явно выданное субъекту право (GRANT) доступа к объекту (листинг 5.6*а*) или отменить явное предоставление или запрет права выполнения SQL-операторов (листинг 5.6*б*).

```
REVOKE [GRANT OPTION FOR]
{ ALL | permission [,...n] }
ON table | view [( column [,...n])]
| ON procedure | ON function
TO security_account [,...n]
[CASCADE]
[AS (role)]
```

а)

```
REVOKE
{ALL | statement[,...n]}
TO security_account [,... .n]
```

б)

Листинг 5.6

Формат SQL-команды REVOKE:
а — отмена явно выданных разрешений доступа;
б — отмена разрешений на выполнение SQL-команд.

Параметры оператора REVOKE аналогичны соответствующим параметрам команд GRANT и DENY, за исключением необязательного параметра [GRANT OPTION FOR]: если в операторе REVOKE этот параметр присутствует и при этом отменяемое право доступа было предоставлено субъекту оператором GRANT без использования параметра [WITH GRANT OPTION], то само разрешение доступа субъекта отменено не будет, а будет отменено только его право на предоставление указанного разрешения другим субъектам; в противном случае будет отменено само разрешение доступа.

Если в операторе REVOKE присутствует параметр [CASCADE], то каскадная отмена разрешений доступа, предоставленных субъекту с помощью параметра [WITH GRANT OPTION], приведет к отмене этих прав, независимо от наличия параметра [GRANT OPTION FOR] в операторе REVOKE.

Просмотр прав доступа

Среда SQL Management Studio содержит средства просмотра прав доступа через вкладку «*Свойства*» соответствующего объекта, пользователя или роли базы данных. Эту же информацию можно получить средствами Transact-SQL, используя системную хранимую процедуру sp_helprotect.

```
sp_helprotect ['object' | 'statement']
        [, 'security_account']
        [, 'grantor'][,  'type']
```

Листинг 5.7

Формат вызова системной процедуры sp_helprotect

Параметры процедуры sp_helprotect:

– ['object' | 'statement'] — имя объекта, категории объектов базы данных или команды Transact-SQL, о правах доступа к которым предполагается получить информацию, например 'sysusers', 'tables', 'views', 'procedures', 'Create Table', 'Backup Database';

– ['security_account'] — имя субъекта доступа;
– ['grantor'] — имя субъекта, который предоставил право;
– ['type'] — тип прав доступа:
• '**o**' — только права доступа к объектам;
• '**s**' — только права выполнения команд;
• '**os**' — оба типа прав (значение по умолчанию).

Ни один из параметров не является обязательным: при выполнении процедуры без параметров она возвратит список всех прав, выданных всем субъектам на все объекты базы данных. Указание (позиционно!) любого из параметров накладывает соответствующий фильтр на результирующий список прав доступа, например sp_helprotect 'Create Role', sp_helprotect Null,'User1' или sp_helprotect Null, Null, 'User2'.

Правом вызова хранимой процедуры sp_helprotect по умолчанию владеет фиксированная роль базы данных Public, следовательно, каждый пользователь базы данных может получить информацию о правах доступа любого из ее пользователей к любому ее объекту.

Системное представление *sys.database_permissions* и TVF-функция *fn_builtin_permissions*() позволяют получить более детальную информацию о правах доступа.

ГЛАВА 17. ПРАКТИКУМ ПО ЗАЩИТЕ ИНФОРМАЦИИ

17.1. Общие методические указания

Структура и содержание. Практикум содержит три практические работы, отражающие следующие аспекты информационной безопасности баз данных:

– архитектура подсистемы защиты информации сервера баз данных (работа № 1);

– программные средства управления доступом к данным (работа № 2);

– иерархия прав доступа к объектам баз данных (работа № 3).

Каждая работа содержит несколько взаимосвязанных заданий, выполнение которых направлено на решение поставленных в работе задач и требует освоения и применения соответствующих инструментальных средств администрирования баз данных.

Программное обеспечение. Все работы выполняются в системе SQL-Server Management Studio, версия сервера баз данных — не старше 2008R2.

Отчет по работе должен содержать:

– цели и задачи, описание методики проведения работы, используемых структур данных и инструментальных программных средств;

– иллюстративный материал (листинги программных компонентов, выводимые на экран результаты их работы, графический материал и пр.);

– анализ полученных результатов с собственными выводами;

– ответы на контрольные вопросы (при их наличии).

Защита. Работа выполняется индивидуально, защита работы проводится в форме собеседования по материалу представленного отчета. В процессе защиты оценивается полнота и качество выполнения практических заданий, грамотность использования инструментальных средств, правильность и обоснованность выводов по результатам работы, качество оформления отчета.

17.2. РАБОТА № 1. Подсистема защиты информации сервера БД

Цель работы: ознакомление со средствами управления информационной безопасностью на уровне сервера и пользовательских баз данных.

Задачи:

– освоить компоненты пользовательского интерфейса *SQL-Server Management Studio*, обеспечивающие возможность просмотра и настройки параметров информационной безопасности;

– исследовать свойства учетных записей, пользователей и ролей сервера и базы данных;

– освоить технику создания и модификации учетных записей сервера, ролей и пользователей баз данных соответствующими средствами языка *Transact SQL*;

– освоить программные средства управления членством в ролях;

– исследовать объекты системного каталога, ответственные за хранение параметров информационной безопасности.

Задание 1. Анализ серверных компонентов системы информационной безопасности

1.1. Выполните вход в систему, используя режим аутентификации Windows. Просмотрите свойства учетных записей (имен входа, logins) сервера. Переименуйте учетные записи. Какие из учетных записей относятся к категории «Учетные записи SQL-Server»?

1.2. Создайте новую учетную запись: выберите для нее режим аутентификации SQL-Server, задайте (и запомните!) пароль и установите свойства учетной записи по своему усмотрению.

1.3. Создайте учетную запись SQL-оператором CREATE LOGIN.

1.4. Просмотрите свойства созданных учетных записей, определите их членство в фиксированных ролях сервера.

1.5. Включите одну из созданных учетных записей в состав членов серверной роли Securityadmin, а другую — в состав членов серверной роли Sysadmin.

1.6. Прямым доступом к системной таблице SysLogins базы данных MASTER определите состав и основные параметры:

– учетных записей Windows;

– учетных записей пользователей Windows;

– учетных записей групп Windows;

– учетных записей SQL-Server.

1.7. Выполните вход в систему, используя режим аутентификации SQL-Server и новую учетную запись, просмотрите свойства доступных учетных записей. Создайте новую базу данных. Прокомментируйте полученные результаты.

1.8. Просмотрите и проанализируйте свойства серверных ролей Sysadmin, Securityadmin и Public.

1.9. Используя системные хранимые процедуры (табл. 5.2):

– определите состав членов всех серверных ролей;

– добавьте учетную запись в состав членов роли Securityadmin;

– определите перечень разрешений, установленных для серверной роли Securityadmin.

1.10. Ознакомьтесь с перечнем и назначением хранимых процедур и функций информационной безопасности, права на выполнение которых получают члены фиксированной роли сервера Securityadmin (используйте документацию разработчика technet).

1.11. Сделайте выводы по результатам выполнения задания и сохраните их в отчете по лабораторной работе.

Задание 2. Анализ компонентов информационной безопасности уровня базы данных

2.1. Выполните вход в систему, используя режим аутентификации Windows. Активизируйте одну из пользовательских баз данных, созданных при выполнении предыдущих лабораторных работ. Просмотрите свойства пользователей базы данных, их членство в фиксированных ролях базы данных.

2.2. Создайте в контексте этой базы данных нового пользователя, сопоставьте его с одной из учетных записей, созданных при выполнении предыдущего задания, включите его в фиксированную роль db_owner.

2.3. Просмотрите и проанализируйте свойства ролей базы данных db_owner, db_securityadmin и public.

2.4. Используя SQL-команды CREATE USER, CREATE ROLE и ALTER ROLE, создайте нового пользователя базы данных, пользовательскую роль, включите пользователя в состав членов этой роли.

2.5. Прямым доступом к системной таблице SysUsers базы данных определите состав и основные параметры пользователей и ролей.

Контрольные вопросы

1. Дайте сравнительные характеристики, назовите преимущества и недостатки двух режимов проверки подлинности пользователей: аутентификация Windows и аутентификация SQL-Server.
2. Членство в каких серверных ролях позволяет пользователю создавать, удалять и модифицировать учетные записи пользователей?
3. Выполнение каких операций разрешено членам серверной роли public?
4. Перечислите основные объекты и субъекты доступа уровня БД.
5. Какие пользователи автоматически создаются при создании БД? Какими правами обладают эти пользователи?
6. Может ли пользователь базы данных одновременно быть членом нескольких ролей базы данных?
7. Может ли пользовательская роль базы данных быть членом фиксированной роли или другой пользовательской роли?

17.3. РАБОТА № 2. Анализ средств управления доступом к данным

Цель работы: изучение средств управления правами доступа субъектов к логическим объектам базы данных MS SQL-Server.

Задачи:

– освоить компоненты пользовательского интерфейса *SQL-Server Management Studio* и программные средства, обеспечивающие возможность просмотра и настройки разрешений доступа;

– исследовать объекты системного каталога, ответственные за хранение информации о правах доступа.

Задание 1. Управление пользователями и ролями БД средствами SQL-Server Management Studio

1.1. Выполните вход в систему, используя режим аутентификации Windows. Активизируйте одну из пользовательских баз данных (или создайте новую). Просмотрите свойства пользователей и фиксированных ролей базы данных. Какие пользователи были созданы сервером самостоятельно, а какие — в результате действий администратора? Определите членство пользователей в ролях базы данных.

1.2. Создайте три новые учетные записи SQL-Server, установите для каждой из них текущую базу данных в качестве базы данных по умолчанию. Создайте в этой базе данных трех новых пользователей, сопоставив их с новыми учетными записями. Включите этих пользователей в состав нескольких фиксированных ролей базы данных (по своему усмотрению).

1.3. Создайте пользовательскую роль базы данных, включите в состав ее членов всех трех новых пользователей, а также пользователя ***guest***. Затем удалите пользователя ***guest***.

1.4. Прямым доступом к системной таблице SysUsers определите состав пользователей, фиксированных и пользовательских ролей базы данных.

1.5. Сделайте выводы по результатам выполнения задания и сохраните их в отчете по лабораторной работе.

Задание 2. Программные средства управления разрешениями уровня базы данных

2.1. Используя системную хранимую процедуру sp_helprotect, определите права доступа к данным и права на выполнение операций для пользователя ***dbo***, а также для созданных при выполнении предыдущего задания пользователя и пользовательской роли базы.

2.2. Используя SQL-команды GRANT и DENY, предоставьте и запретите права доступа пользователям (по своему усмотрению), а затем отмените SQL-командой REVOKE некоторые из предоставленных разрешений и запретов.

2.3. Просмотрите информацию о правах доступа к объектам базы данных, используя хранимую процедуру sp_helprotect, системное представление sys.database_permissions и системную функцию fn_builtin_permissions().

2.4. Выполните вход в систему, используя режим аутентификации SQL-Server и одну из учетных записей пользователей, созданных при выполнении задания 1.2. Активизируйте базу данных, в которой ранее был создан пользователь, выполнивший вход в систему, и экспериментально проверьте действие выданных ему разрешений и запретов.

2.5. Повторно выполните задание 2.4, используя учетную запись другого пользователя этой базы данных.

2.6. Сделайте выводы по результатам выполнения задания.

17.4. РАБОТА № 3.
Исследование иерархии прав доступа к данным

MS SQL-Server поддерживает сложную иерархическую систему разрешений на выполнение операций с объектами баз данных и множество методов и инструментальных средств управления этими разрешениями. При этом параллельно существуют несколько (не всегда строгих) иерархий, например «сервер — база данных», «роль — пользователь», «база данных — схема — таблица — столбец», «процедура — представление — таблица».

Все это позволяет администраторам реализовывать надежные, гибкие и технологичные системы защиты информации, но, с другой стороны, может создавать проблемы, связанные с необходимостью понимания и учета приоритетов разрешений, предоставленных субъектам доступа на разных уровнях и/или разными методами.

Цель работы — изучение системы приоритетов разрешений доступа к данным, реализованной в MS SQL-Server.

Задачи. При выполнении лабораторной работы ставится комплексная задача экспериментального подтверждения, уточнения или опровержения следующих рабочих гипотез, касающихся приоритетов прав доступа к объектам баз данных.

Гипотеза № 1. Запрет доступа (Deny) к объекту всегда имеет более высокий приоритет по сравнению с предоставлением (Grant) доступа.

Гипотеза № 2. Глобальные права доступа к данным, предоставленные пользователю через членство в фиксированных ролях базы данных, имеют более высокий приоритет по сравнению с локальными правами, предоставленными ему явно или через членство в пользовательских ролях базы данных.

Гипотеза № 3. Права доступа, предоставленные пользователю неявно через членство в пользовательской роли базы данных, имеют более высокий приоритет по сравнению с правами, предоставленными ему явно (персонально).

Гипотеза № 4. Права доступа к отдельным столбцам таблицы имеют более низкий приоритет по сравнению с правами, предоставленными на таблицу в целом.

Гипотеза № 5. Право на выполнение (Execute) процедуры или хранимого представления, содержащих SQL-оператор доступа к таблице, позволяет пользователю получить несанкционированный доступ к этой таблице, несмотря на:

– явный запрет (Deny) доступа к этой таблице;

– явный запрет на выполнение операций (например, Select);

– членство пользователя в фиксированных ролях базы данных DB_denydatareader и DB_denydatawriter.

Гипотеза № 6. Право на создание процедуры (Create Proc) позволяет пользователю включить в нее SQL-оператор чтения данных таблицы, доступ к которой ему явно запрещен, и получить таким образом несанкционированный доступ к этой таблице.

Задание

1. Спланируйте эксперименты по проверке рабочих гипотез:

– определите состав субъектов и объектов доступа и соответствующих разрешений;

– подготовьте текстовый документ для формирования отчета по лабораторной работе с описанием каждого из запланированных экспериментов;

– включите в отчет бланки протоколов для регистрации результатов экспериментов.

2. Подготовьте базу данных для проведения экспериментов (можно использовать выполненные ранее собственные разработки).

3. Подготовьте SQL-запросы, хранимые представления и процедуры, необходимые для проверки рабочих гипотез.

4. Создайте в этой базе данных необходимое количество пользователей (сопоставленных с соответствующими учетными записями SQL-Server) и пользовательских ролей.

5. Определите для пользователей и пользовательских ролей соответствующие права доступа к объектам базы данных.

6. Распределите пользователей по фиксированным и пользовательским ролям базы данных.

7. Проведите эксперименты, сохраните в отчете их результаты и собственные выводы по каждой из проверяемых гипотез.

ПРИЛОЖЕНИЕ А
СТАНДАРТНЫЕ ФОРМЫ БЭКУСА — НАУРА (BNF)

В BNF-обозначениях используются следующие элементы:

– символ «**::=**» означает равенство по определению. Слева от знака стоит определяемое понятие, справа — собственно определение понятия;

– КЛЮЧЕВЫЕ СЛОВА (зарезервированные слова, составляющие часть оператора) записываются прописными буквами;

– *заполнители конкретных значений элементов и переменных* записываются курсивом;

– *круглые скобки* ( ) являются элементом оператора;

– *фигурные скобки* { } указывают на то, что все, находящееся внутри них, является единым целым;

– в *квадратные скобки* [ ] заключаются необязательные элементы оператора;

– *вертикальная черта* **|** указывает на то, что все предшествующие ей элементы списка являются необязательными и могут быть заменены любым другим элементом списка, записанным после этой черты;

– *троеточие* «**…**» означает, что предшествующая часть оператора может быть повторена любое количество раз;

– *многоточие*, внутри которого находится запятая «**... , ...**», указывает на то, что предшествующая часть оператора, состоящая из нескольких элементов, разделенных запятыми, может иметь произвольное число повторений.

Листинг А.1 содержит пример BNF-формулы оператора DELETE языка Transact-SQL (в реализации MS SQL-Server-2017). Читателю предлагается самостоятельно проанализировать эту BNF-формулу и сравнить ее с примерами реализации оператора DELETE языка Microsoft Jet SQL, приведенными на листингах 4.6, 4.9 и 4.10 учебника.

```
[ WITH <common_table_expression> [ ,...n ] ]
DELETE
    [ TOP ( expression ) [ PERCENT ] ]
    [ FROM ]
    { { table_alias
      | <object>
      | rowset_function_limited
      [ WITH ( table_hint_limited [ ...n ] ) ] }
      | @table_variable
    }
    [ <OUTPUT Clause> ]
    [ FROM table_source [ ,...n ] ]
    [ WHERE { <search_condition>
            | { [ CURRENT OF
                    { { [ GLOBAL ] cursor_name }
                        | cursor_variable_name
                    }
                ]
              }
            }
    ]
    [ OPTION ( <Query Hint> [ ,...n ] ) ]
[; ]
<object> ::=
{
    [ server_name.database_name.schema_name.
      | database_name. [ schema_name ]
      | schema_name
    ]
    table_or_view_name
}
```

Листинг А.1
BNF-формула оператора DELETE

ПРИЛОЖЕНИЕ Б
ПЕРЕЧЕНЬ ПРОФЕССИОНАЛЬНЫХ СТАНДАРТОВ

1. Standard Occupational Classification [Electronic resource] / U. S. Bureau of Labor Statistics. — Electronic data (1 file: 974848 bytes). URL: https://www.bls.gov/ soc/2018/soc_structure_2018.pdf, free. Title from screen.

2. ГОСТ Р 56413-2015. Информационные технологии. Европейские профили профессий ИКТ-сектора / CWA 16458:2012 Information technologies. European ICT professional profiles. — Введ. 01.06.2016 приказом Федерального агентства по техническому регулированию и метрологии от 29 мая 2015 г. № 465-ст.

3. Профессиональный стандарт 06.011 «Администратор баз данных». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 12 декабря 2016 г. № 727н.

4. Профессиональный стандарт 06.015 «Специалист по информационным системам». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 12 декабря 2016 г. № 727н.

5. Профессиональный стандарт 06.026 «Системный администратор информационно-коммуникационных систем». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 5 октября 2015 г. № 684н.

6. Профессиональный стандарт 06.033 «Специалист по защите информации в автоматизированных системах». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 15 сентября 2016 г. № 522н.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Бейли, Л.* Изучаем SQL. — СПб. : Питер, 2012. — 592 с.
2. *Бобровский, С.* Oracle Database 10g для Linux. Эффективное использование. — М. : Лори, 2004. — 487 с.
3. *Вьейра, Р.* Программирование баз данных Microsoft SQL Server 2008. Базовый курс. — СПб. : Диалектика-Вильямс, 2010. — 816 с.
4. *Дейт, К.* Основы будущих систем баз данных. Третий манифест / К. Дейт, Х. Дарвен ; под ред. С. Д. Кузнецова. — 2-е изд. — М. : Янус-К, 2004.
5. Документация PostgreSQL и Postgres Pro [Электронный ресурс]. — Режим доступа: https://postgrespro.ru/docs/
6. *Ицик, Бен-Ган.* Microsoft SQL Server 2008. Основы T-SQL. — СПб. : БХВ-Петербург, 2009. — 430 с.
7. *Кузнецов, С.* Базы данных. Вводный курс [Электронный ресурс]. — Режим доступа: http://citforum.ru/ database/advanced_intro/
8. *Мамаев, Е. В.* Microsoft SQL Server 2000. — СПб. : БХВ-Петербург, 2004. — 1280 с.
9. *Риккарди, Г.* Системы баз данных. Теория и практика использования в Internet и среде Java. — М. : Издат. дом «Вильямс», 2001. — 480 с.
10. Справочник по Transact-SQL [Электронный ресурс]. — Режим доступа: http://msdn.microsoft.com/ru-ru/library/bb510741.aspx.
11. *Тарасов, С. В.* СУБД для программиста. Базы данных изнутри. — М. : СОЛОН-Пресс. — 320 с.
12. *Туманов, В.* Основы проектирования реляционных баз данных [Электронный ресурс]. — Режим доступа: https://www.intuit.ru/studies/courses/ 1095/191.
13. *Цикритизис, Д.* Модели данных / Д. Цикритизис, Ф. Лоховски. — М. : Финансы и статистика, 1985. — 344 с.
14. *Чен, П.* Модель «сущность — связь» — шаг к единому представлению данных [Электронный ресурс] / пер. с англ. С. Кузнецова. — Режим доступа: http://citforum.ru/database/classics/chen/.
15. *Четвериков, В. Н.* Базы и банки данных : учебник для вузов / В. Н. Четвериков [и др.]. — М. : Высш. шк., 1987. — 248 с.
16. *Bachman, C. W.* Data Structure Diagrams. — N. Y. : ACM SIGMIS Database: the DATABASE for Advances in Information Systems. 1969. — Vol. 1, is. 2. — P. 4–10.
17. *Brown, A. P. G.* Modelling a Real-World System and Designing a Schema to Represent It / eds. Douque and Nijssen. — Data Base Description, 1975.
18. *Burleson, D. K.* The CODASYL Network Model [Электронный ресурс]. — Режим доступа: http://www.remote-dba.net/t_object_codasyl_ network.htm.
19. CA IDMS (Integrated Database Management System) [Электронный ресурс]. — Режим доступа: https://ru.bmstu.wiki/CA_IDMS.

20. Cachè. High performance multi-model database [Электронный ресурс]. — Режим доступа: http://www.intersystems.com/ru/our-products/cache/cache-overview/.

21. *Chen, P.* The Entity-Relationship Model — Toward a Unified View of Data // ACM Transactions on Database Systems (TODS). — 1976. — Vol. 1. — P. 9–36.

22. *Codd, E. F.* Extending the Relational Database Model to Capture More Meaning // ACM TODS. — 1970. — № 4.

23. *Codd, E. F.* A Relational Model of Data for Large Shared Data Banks. CACM 13 (6). — June 1970.

24. *Codd, E. F.* The Relational Model For Database Management Version 2. — Reading, Mass. : Addison-Wesley, 1990.

25. *Darwen, H.* The Second Life of Relational Model / H. Darwen, C. J. Date // DB/M Magazine. — 1995. — № 1–2.

26. *Darwen, H.* The Third Manifesto / H. Darwen, C. J. Date //ACM SIGMOD Record 24. — 1995. — № 1.

27. *Date, C. J.* Notes Toward a Reconstitued Definition of the Relational Model Version 1 (RM/V1) // Date C. J. (with Darwen H.) Relational Databases: Selected Writings 1989–1991. — Reading, Mass. : Addison-Wesley, 1992.

28. *Date, C. J.* The Relational Model // Date C. J. An Introduction to Database Systems. 7th ed. — Reading, Mass. : Addison-Wesley, 2000.

29. *Date, C. J.* Temporal Data and The Relational Model / C. J. Date, H. Darwen, N. A. Lorentzos. — Reading, Mass. : Addison-Wesley, 2003.

30. A History and Evaluation of System R. — IBM Research Laboratory San Jose, California, 1981.

31. https://people.eecs.berkeley.edu/~brewer/cs262/SystemR.pdf.

32. *Kempe, S. A.* Short History of the ER Diagram and Information Modeling [Электронный ресурс]. — Режим доступа: http://www.dataversity.net.

33. *Long, R.* IMS Primer [Электронный ресурс] / R. Long, M. Harrington, R. Hain, G. Nicholls. — IBM, 2000. — 300 с. — (IBM Redbook). — Режим доступа: http://www.redbooks.ibm.com/redbooks/ pdfs/sg245352.pdf.

ОГЛАВЛЕНИЕ

Владимир Константинович ВОЛК

БАЗЫ ДАННЫХ
ПРОЕКТИРОВАНИЕ, ПРОГРАММИРОВАНИЕ,
УПРАВЛЕНИЕ И АДМИНИСТРИРОВАНИЕ
Учебник

Зав. редакцией
литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *Т. С. Спирина*
Подготовка макета *Э. Я. Юзеев*
Корректор *Т. А. Кошелева*
Выпускающий *Е. Е. Егорова*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр. Юрия Гагарина, д. 1, лит. А
Тел./факс: (812) 336-25-09, 412-92-72
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 09.10.19.
Бумага офсетная. Гарнитура Школьная. Формат 70×100 $^{1}/_{16}$.
Печать офсетная. Усл. п. л. 19,83. Тираж 100 экз.

Заказ № 658-19.

Отпечатано в полном соответствии с качеством
предоставленного оригинал-макета в АО «Т8 Издательские Технологии».
109316, г. Москва, Волгоградский пр., д. 42, к. 5.