ISBN 978-5-4217-0472-0

Курганский
государственный
университет

Библиотечно-издательский
центр
65-48-12

БАЗЫ ДАННЫХ. Часть 1. Проектирование и программирование

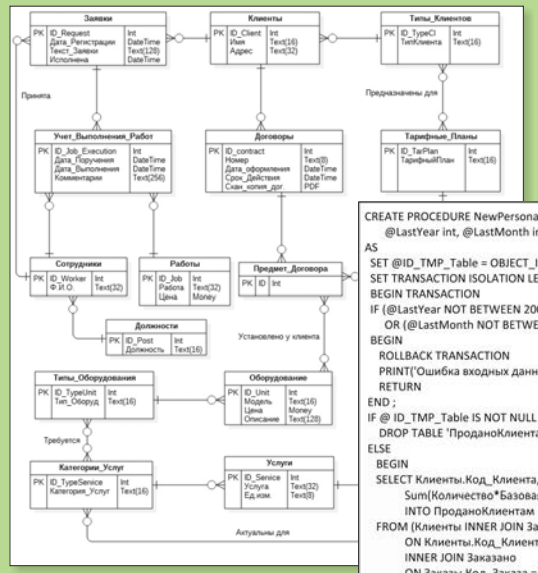
УЧЕБНОЕ ПОСОБИЕ

В.К.Волк

В.К.Волк

БАЗЫ ДАННЫХ

Часть 1
ПРОЕКТИРОВАНИЕ
И ПРОГРАММИРОВАНИЕ




```
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
```

Министерство науки и высшего образования
Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

В.К. Волк

Базы данных

Часть 1

Проектирование и программирование

Учебное пособие

Курган 2018

УДК 004.658.2(075.8)
ББК 32.973я73
В 67

Рецензенты

кафедра прикладной информатики и автоматизации бизнес-процессов Шадринского государственного педагогического университета (канд. физ.-мат. наук, профессор В.Ю. Пирогов);

кафедра информационных систем Тюменского государственного университета (канд. физ.-мат. наук, доцент П.К. Моор).

Печатается по решению методического совета Курганского государственного университета.

Научный редактор – канд. физ.-мат. наук, проф. В.А. Симахин.

Волк В. К.

Базы данных. Часть 1. Проектирование и программирование : учебное пособие. – Курган : Изд-во Курганского гос. ун-та, 2018. – 178 с.

Учебное пособие посвящено 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. Пособие состоит из двух взаимосвязанных частей, представляющих все фазы жизненного цикла базы данных. Первая часть содержит введение в проблематику баз данных и описание технологии их проектирования и программирования. База данных рассматривается как информационная модель предметной области, а ее проектирование – как многоэтапный процесс последовательного преобразования концептуальной модели в логическую (реляционную) модель данных, включающий процедуру ее нормализации, и последующую программную реализацию средствами языка SQL.

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

Рисунков − 31, таблиц − 11, листингов программного кода – 28.

ISBN 978-5-4217-0472-0

УДК 004.658.2(075.8)
ББК 32.973я73

Содержание

ПРЕДИСЛОВИЕ

Высокая ценность информации признавалась во все времена, но только во второй половине XX века, 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, а подсистема обработки информации включает множество прикладных программ, получающих доступ к базе данных для чтения и/или модификации информации.

АИС относятся к категории социотехнических систем, важнейшей частью которых, наряду с аппаратным и программным обеспечением, являются разработчики, администраторы и конечные пользователи, участвующие в создании или эксплуатации АИС на различных стадиях ее жизненного цикла. Материал первой части учебного пособия, ориентирован на проектировщиков и программистов баз данных, обычно выделяемых в отдельную категорию разработчиков программных систем. Рассматривается технология разработки концептуальной ER-модели предметной области АИС, методы ее последующего преобразования в логическую (реляционную) модель данных и средства программной реализации логической модели на языке SQL.

В первой главе обсуждается проблема и пути достижения автономности БД в составе программного обеспечения АИС, приведена типовая схема процесса разработки БД путем последовательного преобразования ее концептуальной, логической и физической моделей, рассмотрены основные свойства иерархической, сетевой, реляционной и объектно-ориентированной логических моделей данных.

Вторая глава – введение в реляционную модель данных Кодда: рассматриваются три ее базовых компонента: структурный, целостностный и манипуляционный, включая элементы реляционной алгебры и реляционного исчисления кортежей, в объеме, достаточном для понимания методов

нормализации реляционной БД и освоения базовых конструкций языка SQL.

В третьей главе детально рассмотрена 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. Завершает главу проектный практикум, содержащий задания для проектирования несложных БД с примерами выполнения и оформления проектов.

Четвертая глава учебного пособия посвящена программной реализации реляционных БД и является, по существу, практическим введением в язык SQL. На примере одного из диалектов языка, соответствующего стандарту SQL-89, рассмотрен базовый набор операторов языка определения данных, языка запросов и языка манипулирования данными. Рассмотрены также элементы процедурного расширения языка, соответствующие стандарту SQL-92. Практикум по SQL-программированию, завершающий главу, может быть использован как при проведении лабораторных занятий, так и при самостоятельном изучении языка.

В пятой главе, завершающей первую часть учебного пособия, обсуждаются базовые функции реляционных систем управления базами данных (СУБД) и более детально – проблемы многопользовательского доступа к БД и пути их решения средствами управления транзакциями и блокировками.

Изложение материала носит прикладной характер, основное внимание уделено вопросам технологии проектирования и программирования реляционных баз данных. Теоретическим аспектам построения баз данных посвящены многочисленные публикации [4, 7, 8, 16, 21, 24 – 33], среди которых можно выделить курс лекций С. Кузнецова [7], охватывающий основные разделы технологии БД, в том числе и теоретические разделы, в которых более детально, чем в настоящем учебном пособии, рассмотрена реляционная модель данных: реляционная алгебра Кодда, алгебра «А» Дэйта и Дарвена, реляционное исчисление кортежей и доменов, теория нормальных форм отношений.

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

Глава 1
ОСНОВНЫЕ КОНЦЕПЦИИ

1.1 Автономность баз данных

Автономность подсистемы хранения данных, в основе которой 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 существенного 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 кода программы.

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 изменений 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 (рисунок 1.1).

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.

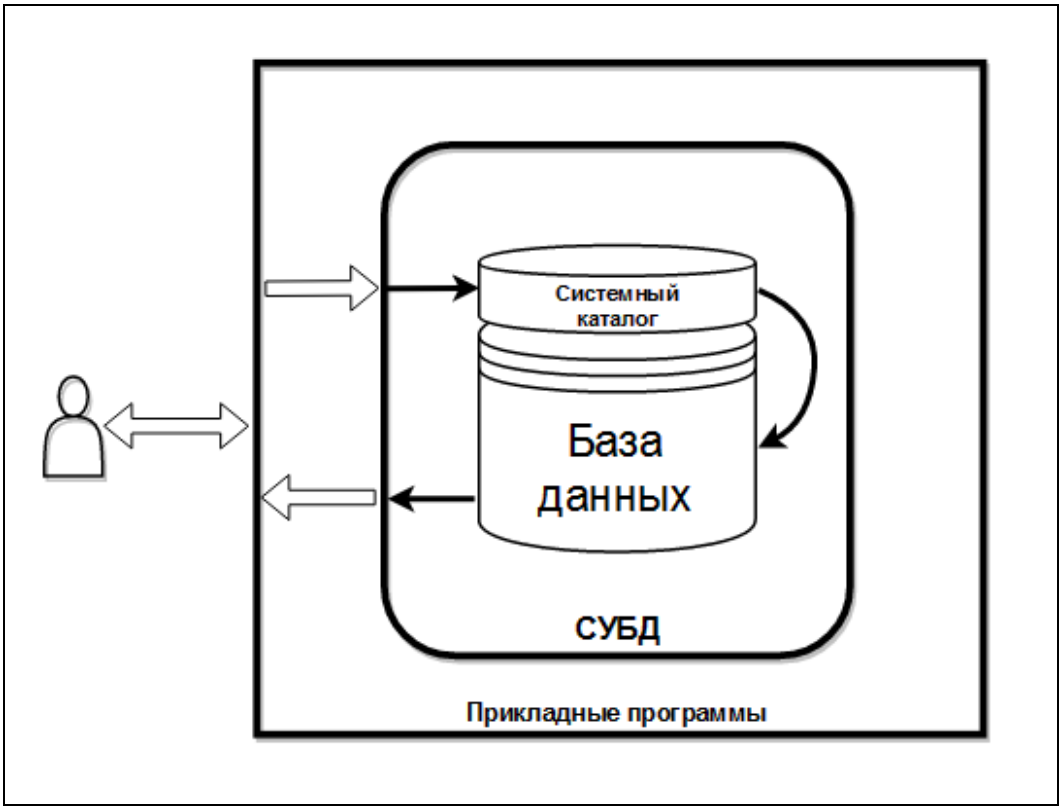


Рисунок 1.1 – Схема взаимодействия прикладных программ с базой данных

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.

1.2 Технологии проектирования баз данных

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 моде-*

ли – 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 учебном пособии 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.

1.3 Концептуальная информационная модель

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 60-х – 70-х годах прошлого века. В этих работах была предложена концептуальная модель предметной области, представляющая результат ее объектной декомпозиции. В основе концептуальной модели – 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.

1.4 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.

1.4.1 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 года *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 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 (рисунок 1.2) корневой сегмент «*Клиент*», 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 поступивших от них заявках.

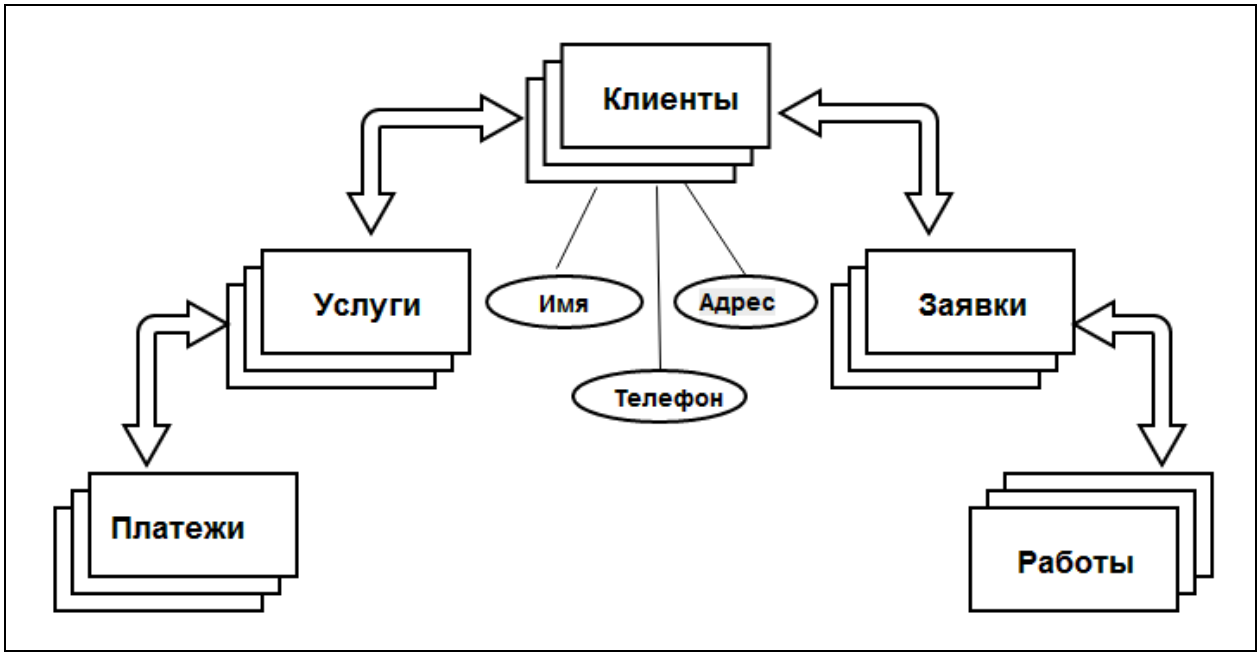
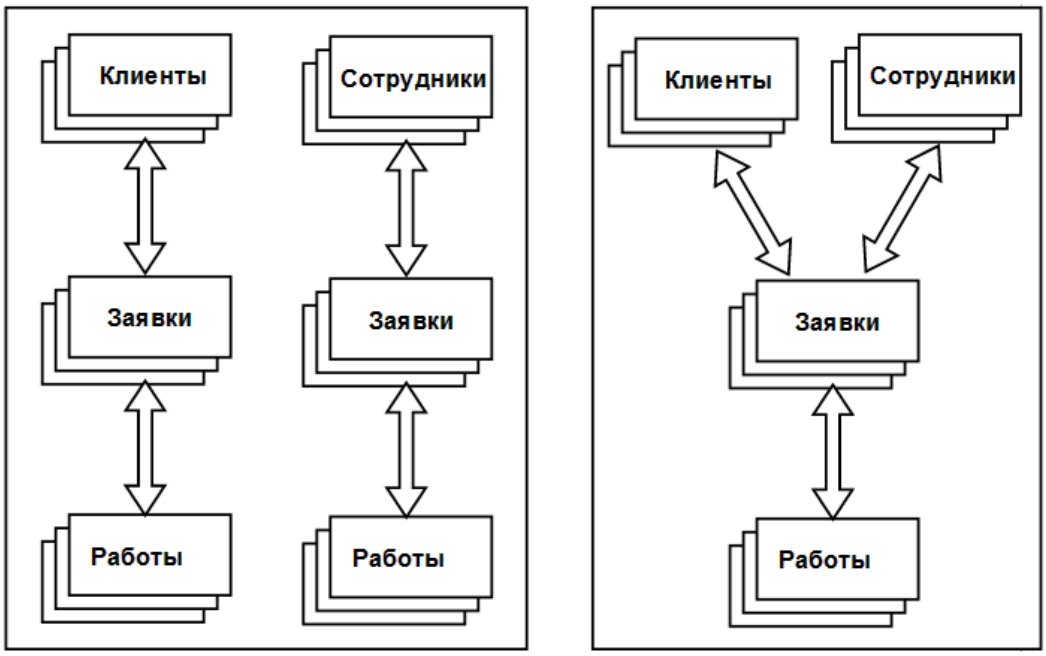


Рисунок 1.2 – Фрагмент иерархической модели данных

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 (рисунок 1.2) необходимо дополнительно учитывать распределение работ по выполнению заявок между сотрудниками, потребуется создать дубликаты вершин в различных деревьях базы данных (рисунок 1.3 а).



а б

а) иерархическая модель; б) отказ от строгой иерархии

Рисунок 1.3 – Дублирование данных в иерархической модели

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

1.4.2 Сетевая модель *CODASYL*

В 1969 году Конференцией по языкам систем данных (Conference on Data Systems Languages) была разработана *сетевая модель данных*, получившая название «*модели CODASYL*» [18]. Спецификация CODASYL содержит детальную 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 были также определены функции администрирования, включающие операции резервного копирования и восстановления БД, управления производительностью, сбора статистики, аудита, авторизации пользователей и др.

Основу логической модели данных CODASYL (рисунок 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.

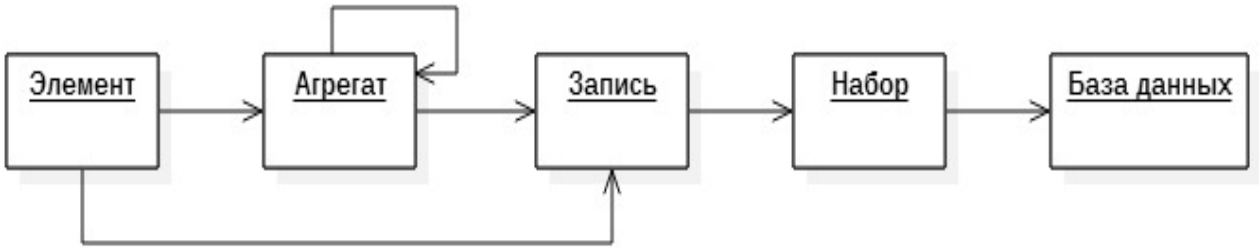


Рисунок 1.4 – Компоненты сетевой модели данных CODASYL

Таким образом, агрегат представляет 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.

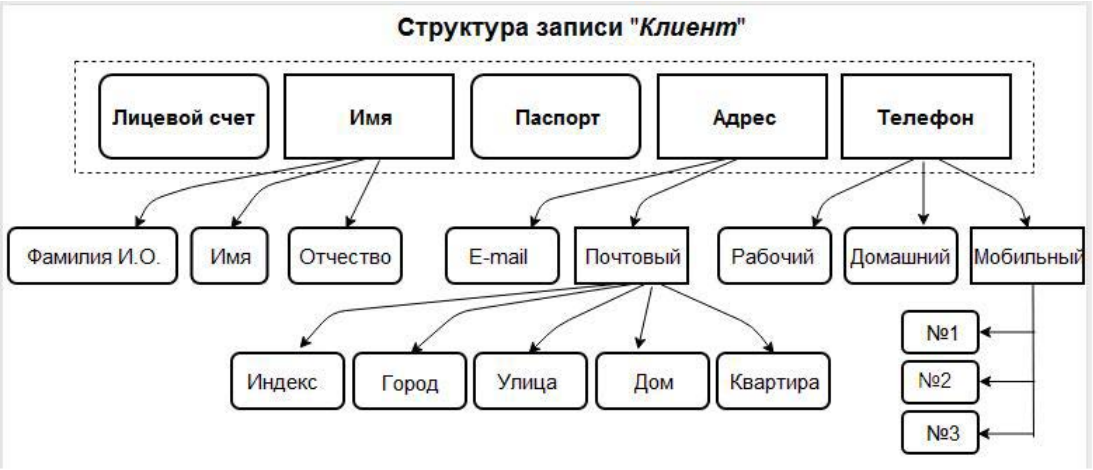


Рисунок 1.5 – Структура записи сетевой модели CODASYL

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.

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 атомарно-

сти 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 «диаграммы Бахмана» и дающая программисту визуальное 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 (рисунки 1.2 и 1.3 а). Наборы записей на диаграмме обозначены линиями со стрелками, направленными от записей-владельцев к записям-членам наборов.

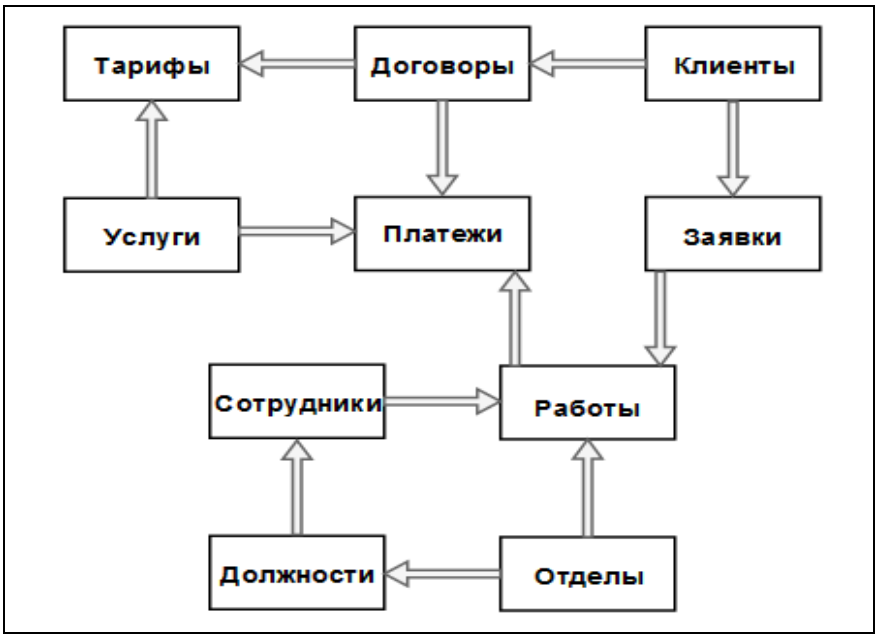


Рисунок 1.6 – Графическое представление фрагмента модели CODASYL

Набор «*Услуги→Тарифы*» представляет 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, устанавливающих 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) могла 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 году 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.

1.4.3 Реляционная модель

Теоретические разработки в области реляционных моделей данных были выполнены в 60-х – 70-х годах 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 фундаментальная 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 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 его до-

ступным 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 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 привело к тому, что к середине 1980-х годов высокая технологичность реляционных СУБД сделала их более популярными, а традиционная их критика за низкую производительность перестала быть актуальной.

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.

1.4.4 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 до-реляционных и реляционных систем. Постреляционные СУБД поддерживают объектные и объектно-реляционные модели данных

---

[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.

и обеспечивают возможность разработчикам 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 учебного пособия – 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 год. В 1996 году 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 года, поддерживает *транзакционную многомерную модель данных* (TMDM), которая позволяет оптимально хранить данные в виде многомерных разреженных массивов, а 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 объ-

ектную 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.

Глава 2
РЕЛЯЦИОННАЯ МОДЕЛЬ 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.

2.1 Структуры данных

Концепции построения реляционной модели были рассмотрены в первой главе учебного пособия (п.1.4.3). Единственной структурой данных, допустимой в 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).

2.2 Ограничения целостности

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 – это 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.

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.

---

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

Требование «экономичности» первичных ключей вытекает из способа реализации связей (п. 3.3.2) 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 рассмотрено во второй части данного учебного пособия и связано с использованием на уровне физической модели 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 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 отношения *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* (п. 3.3.1).

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 2.1, 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 (п. 2.3.2). 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.

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).

```
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 2.1 – Примеры использования ограничений целостности

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).

2.3 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 данных определено два базовых «механизма» реализации таких методов – это *реляционная алгебра*, основанная на теории множеств, и основанное на математической логике *реляционное исчисление*. 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 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 учебного пособия, 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].

2.3.1 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-операнды 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 (конкатенации, 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, таких, 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 2.1 приведены примеры, иллюстрирующие применение операций реляционной алгебры для реализации запросов к базе данных, моделирующей контингент студентов университета (листинг 2.1).

Таблица 2.1 – 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	

2.3.2 Реляционное исчисление

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 2.1) заданы кортежные переменные: **RANGE** *Group* **IS** *Groups* и **RANGE** *Student* **IS** *Students*.

Таблица 2.2 иллюстрирует области истинности и семантику результатов вычисления *WFF*-формул, заданных на этих кортежных переменных.

Таблица 2.2 – 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 курсов
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 стипендии
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 стипендии
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%

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 2.2:

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*)

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, накладываемые 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 2.1 и данные таблиц 2.1 и 2.2, подготовьте собственные примеры выражений реляционной алгебры и WFF-формул реляционного исчисления кортежей.

Глава 3
ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

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 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.

Результаты *физического проектирования* базы данных должны обеспечить эффективное хранение данных и высокую производительность реализации пользовательских запросов с учетом специфики конкретной СУБД. 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 запросов и высокопроизводительный доступ к данным.

В этой главе рассматривается технология логического проектирования реляционных баз данных; проблемы физического проектирования баз данных и их реализации в среде СУБД MS Microsoft SQL Server обсуждаются во второй части данного учебного пособия.

3.1 Типовые стадии проекта базы 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.

Стадии и результаты проекта базы данных приведены на рисунке 3.1.

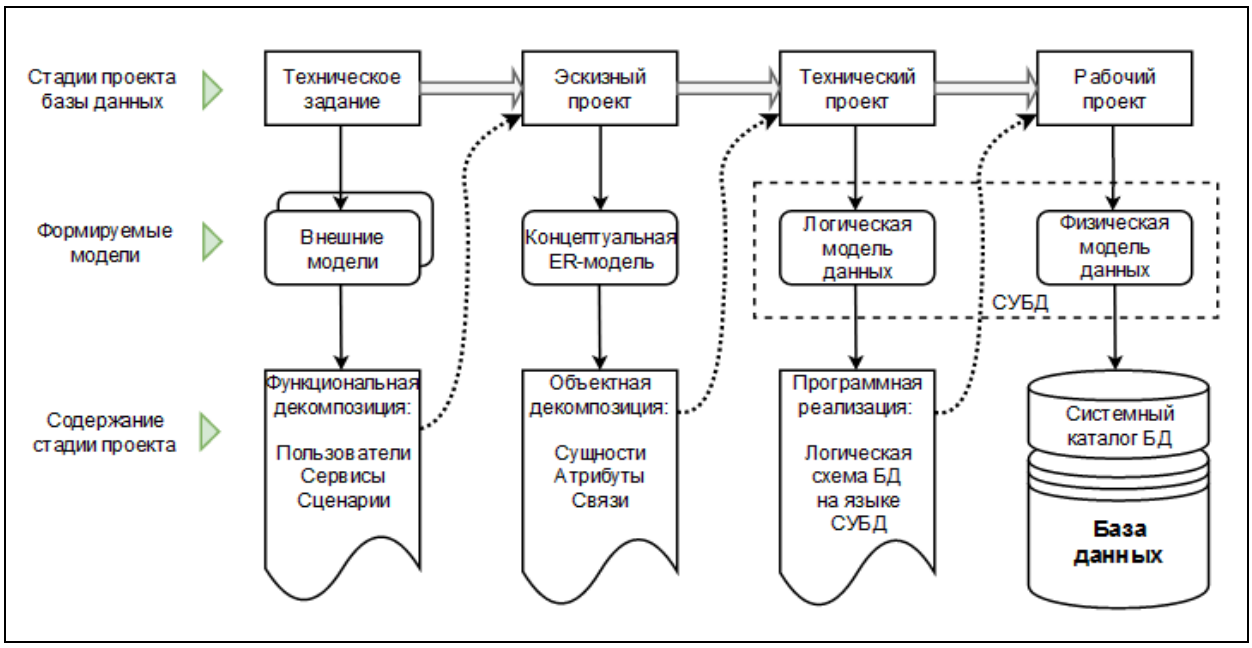


Рисунок 3.1 – Стадии проекта базы данных

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 – стадии *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*.

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 производится администратором базы данных по результатам мониторинга работы пользователей АИС в процессе ее эксплуатации. Структура физической модели данных и средства ее оптимизации рассмотрены во второй части данного учебного пособия.

3.2 Эскизный проект. Разработка концептуальной ER-модели

3.2.1 Два уровня объектной декомпозиции

В соответствии с базовыми принципами проектирования сложных объектов, обсуждавшимися выше (п. 1.2), 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 отношениями вложенности и 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* (п. 1.3) – 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.

3.2.2 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 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 свой-

ством 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 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 3.2).

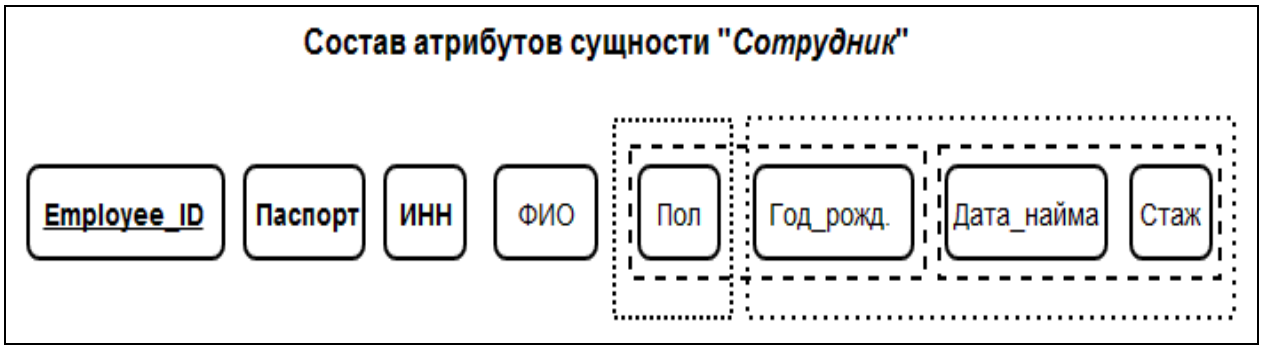


Рисунок 3.2 – Пример описания сущности ER-модели

[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.

Все атрибуты сущности, за исключением атрибута *Employee_ID*, имеют статус *описательных*, так как представляют свойства сотрудников предприятия, существенных для системы кадрового учета.

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 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.

3.2.3 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 различной 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 участвует 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 со следующими 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 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*», кратность данной 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 атрибу-

ты *Доля_должностной_ставки*, *Дата_назначения* и *Номер_приказа* – эти атрибуты 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 3.3 – 3.7.

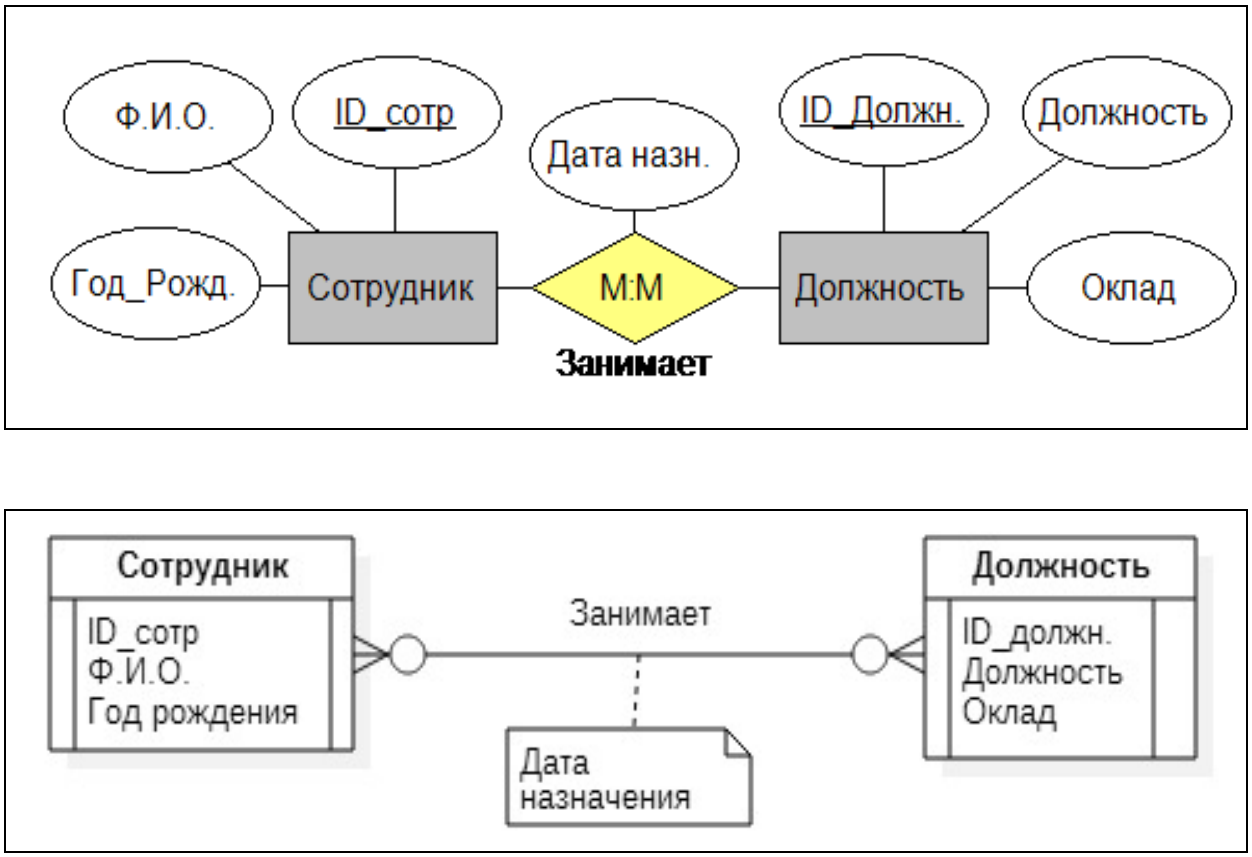


Рисунок 3.3 – Два способа обозначений связей кратности M:N

ER-диаграмма, представленная на рисунке 3.4, 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 принадлеж-

ность строго одной специальности, одной форме обучения и одному образовательному уровню.

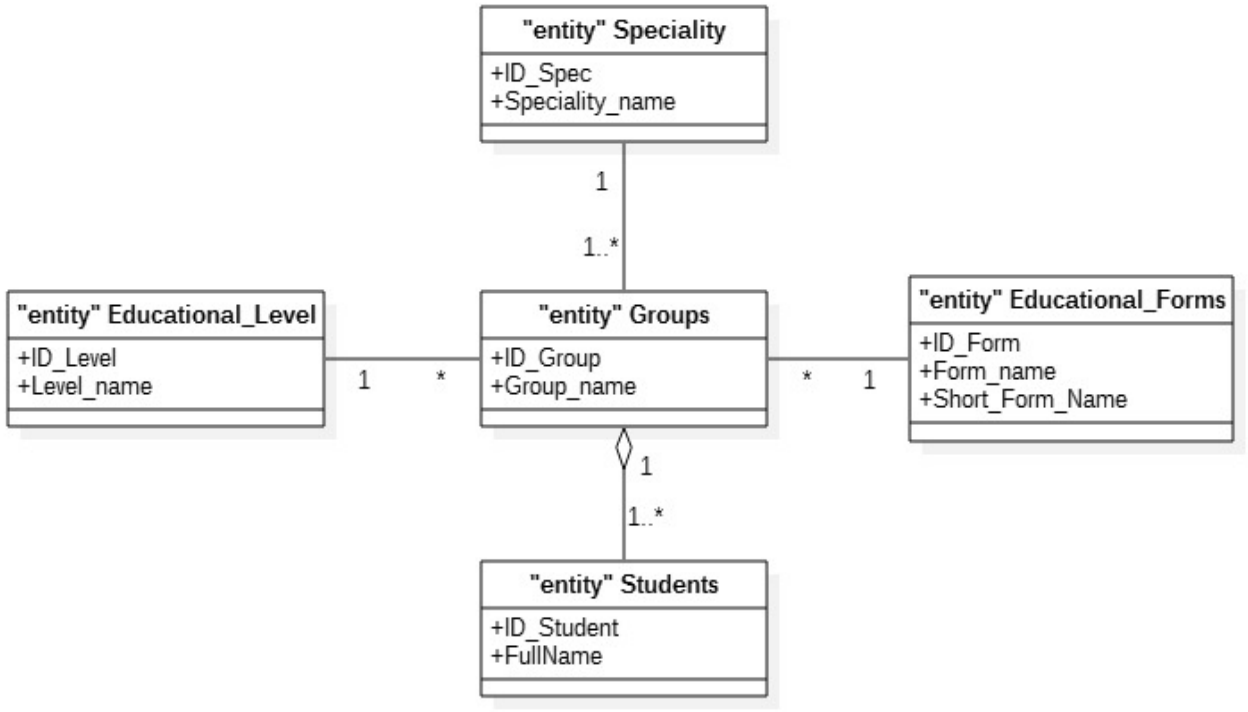


Рисунок 3.4 – Фрагмент ER-диаграммы модели «Контингент студентов»

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.

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.

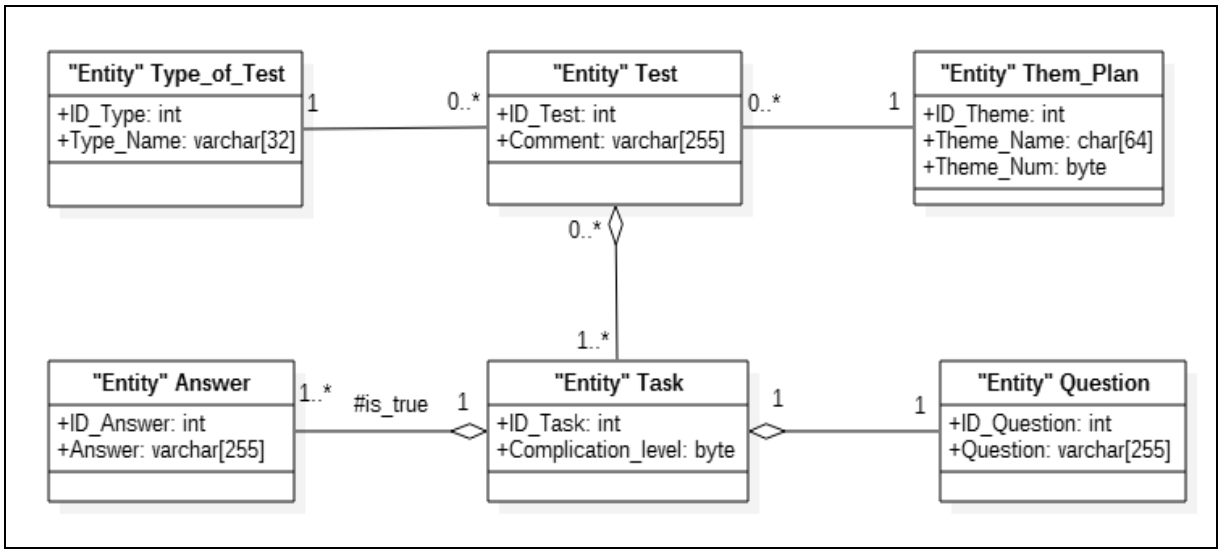


Рисунок 3.5 – Фрагмент ER-диаграммы модели «Тестирование»

На рисунке 3.5 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*), часть из которых являются правильными (описательный атрибут экземпляра связи заданий с правильными ответами получит значение «*is_true*»).

На рисунке 3.6 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 может иметь собственные атрибуты и может участвовать в разных связях с другими сущностями.

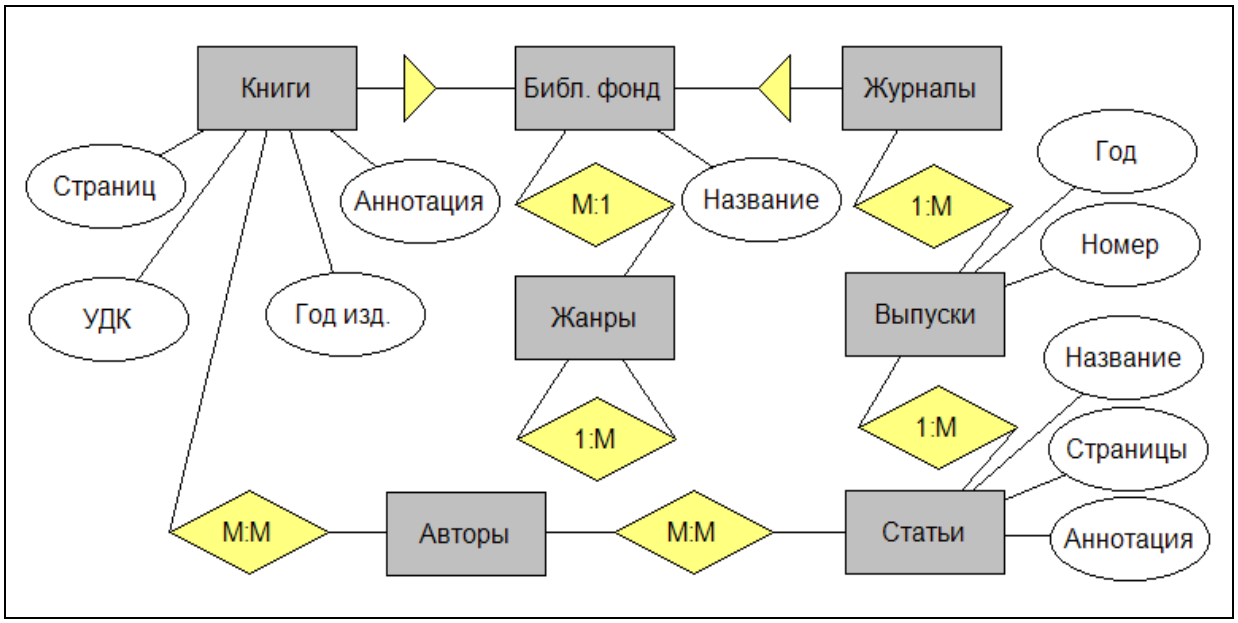


Рисунок 3.6 – Пример использования унарных связей
и связей вида «обобщение»

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.

3.2.4 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 собственный первичный ключ и может участвовать в связях с другими слабыми сущностями в качестве сильной сущности (рисунок 3.7).

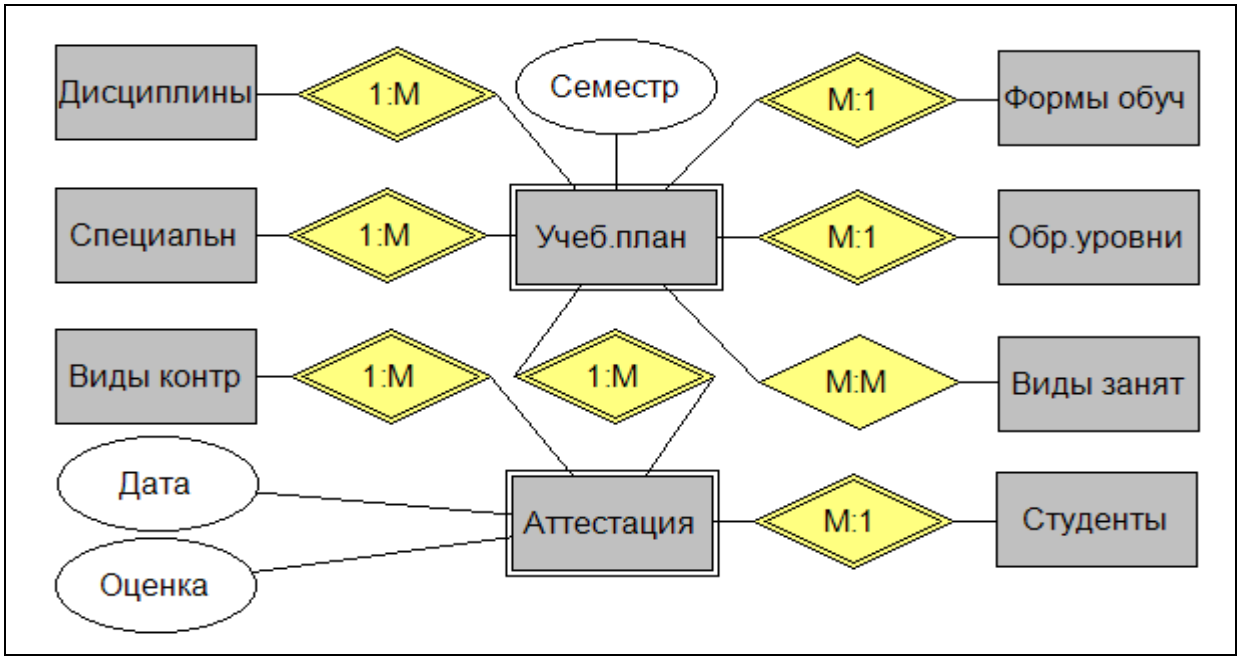


Рисунок 3.7 – Пример использования «слабых сущностей»

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.

3.2.5 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 при-

мера при обсуждении концепций иерархической (п. 1.4.1) и сетевой (п. 1.4.2) 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 3.8.

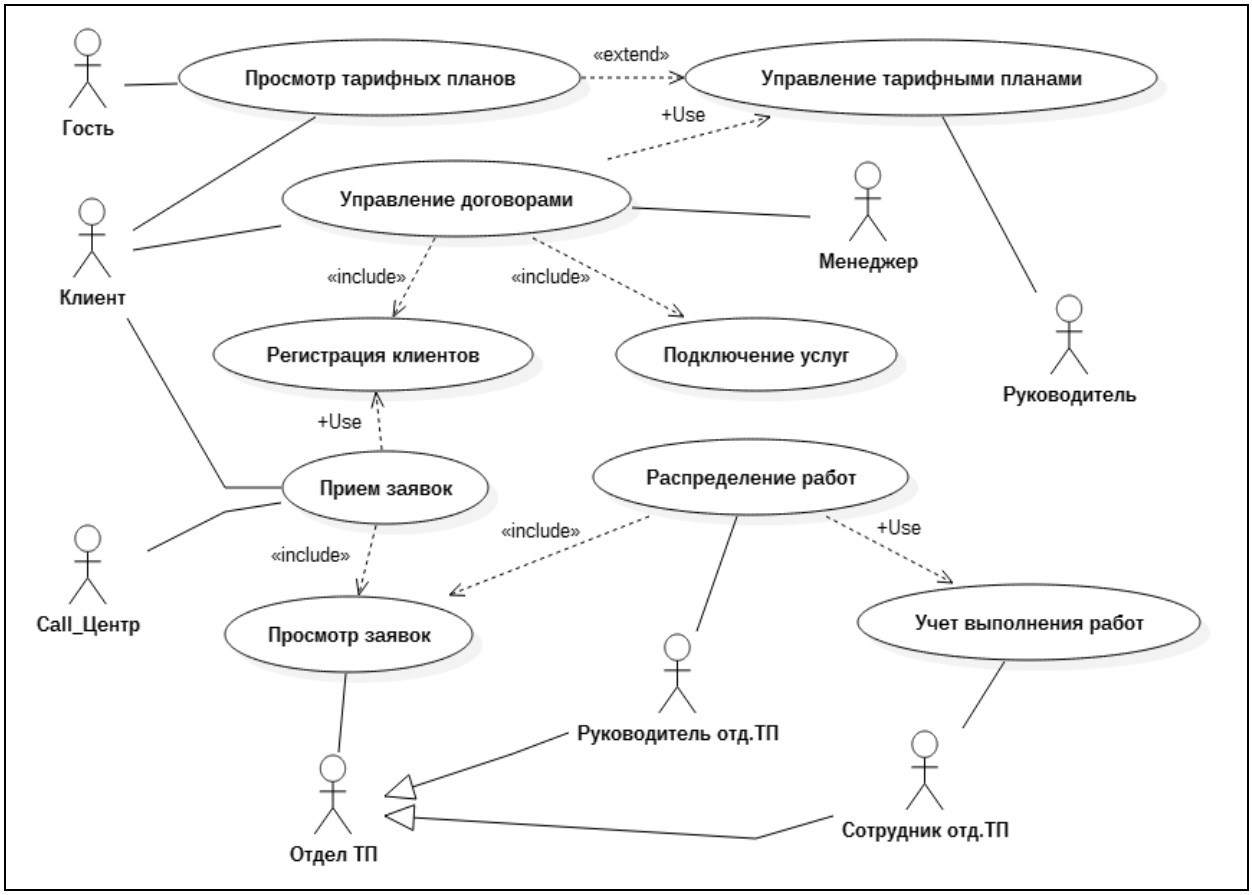


Рисунок 3.8 – UseCase-диаграмма подсистемы учета работы с клиентами интернет-провайдера

Внешними пользователями подсистемы являются *Гость* и *Клиент*, которым доступен сервис просмотра тарифных планов, а *Клиенту* – 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*.

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 – вторая 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, выработанных на стадии технического задания.

При выполнении крупномасштабных проектов рекомендуется проводить структурную декомпозицию системы в три этапа (п. 3.2.1), последовательно увеличивая степень детализации рассмотрения ее свойств. По завершению 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 3.9 – 3.13.

На первом этапе по результатам анализа UseCase-модели АИС (рисунок 3.8) 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* (рисунок 3.9).

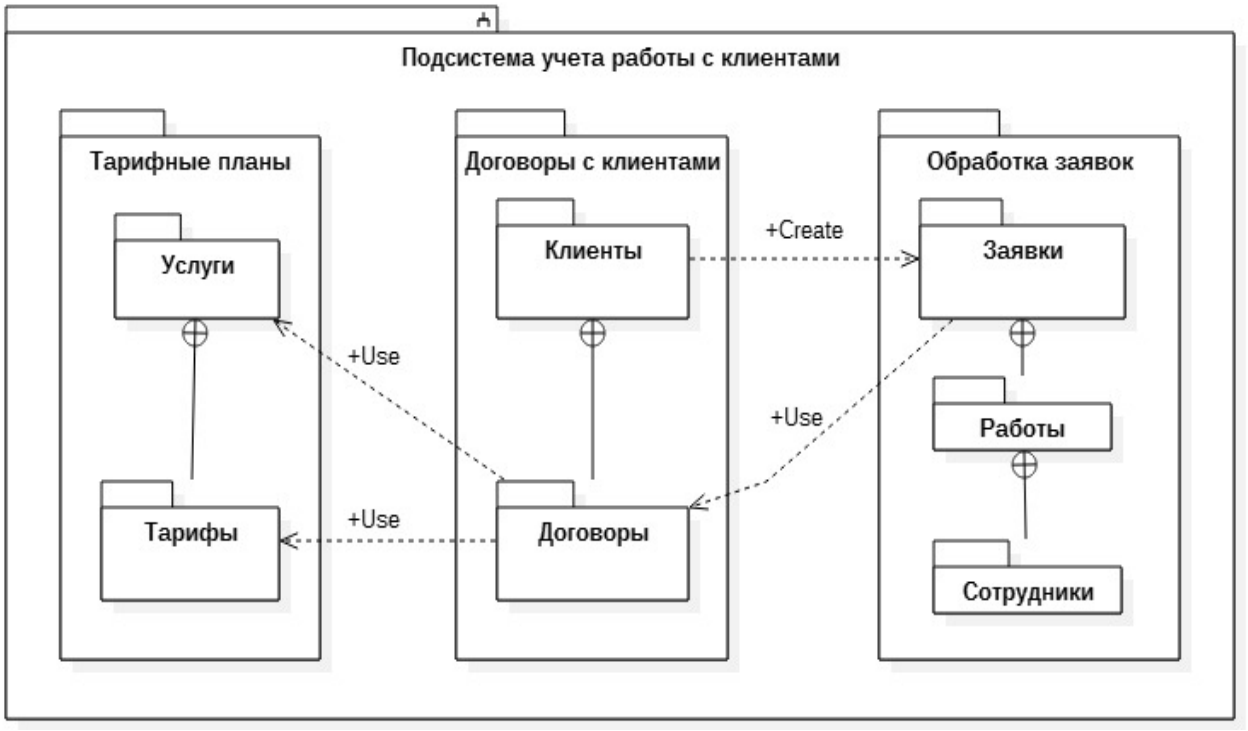


Рисунок 3.9 – UML-диаграмма пакетов подсистемы учета работы с клиентами

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.

Пакет *Обработка заявок* обеспечивает регистрацию заявок на обслуживание, поступающих от клиентов, а также планирование и учет вы-

полнения 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* (рисунок 3.10) 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.

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*», и категория «*Телефония*» может включать услуги «*Мобильная связь*» и «*IP-телефония*», а 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*».

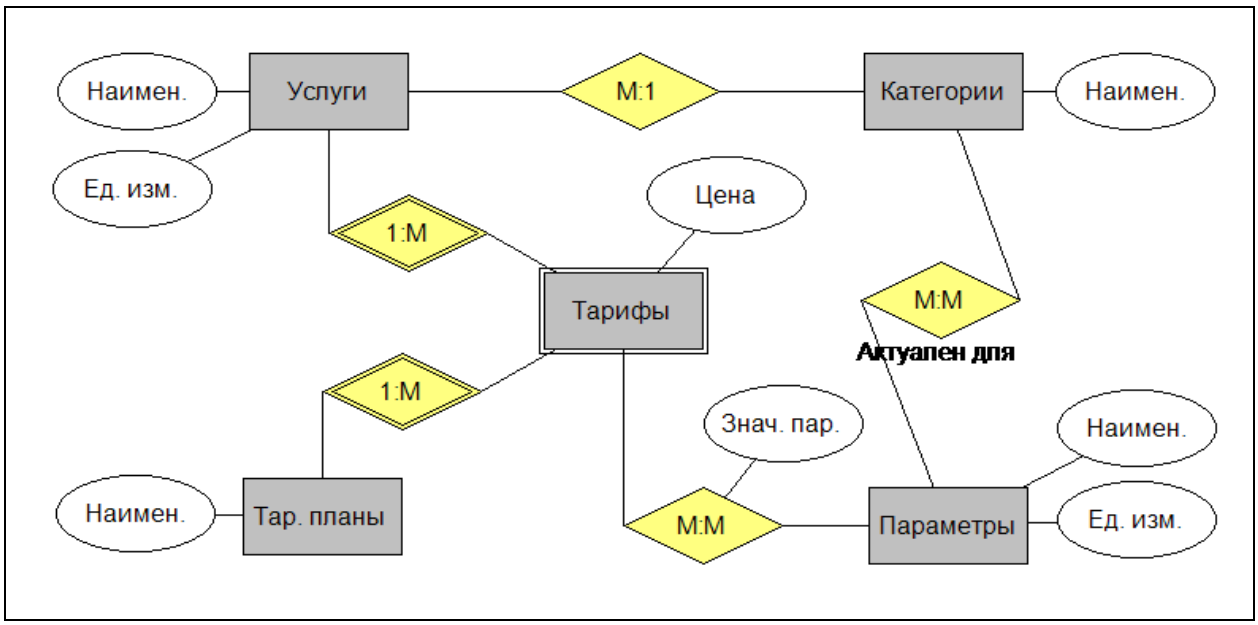


Рисунок 3.10 – ER-диаграмма пакета *Тарифные планы*

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 раз-

работанных локальных ER-моделей. Пример такого анализа для пакета *Тарифные планы* приведен выше, оценку качества ER-моделей для двух остальных пакетов (рисунки 3.11 и 3.12) читателю предлагается провести самостоятельно.

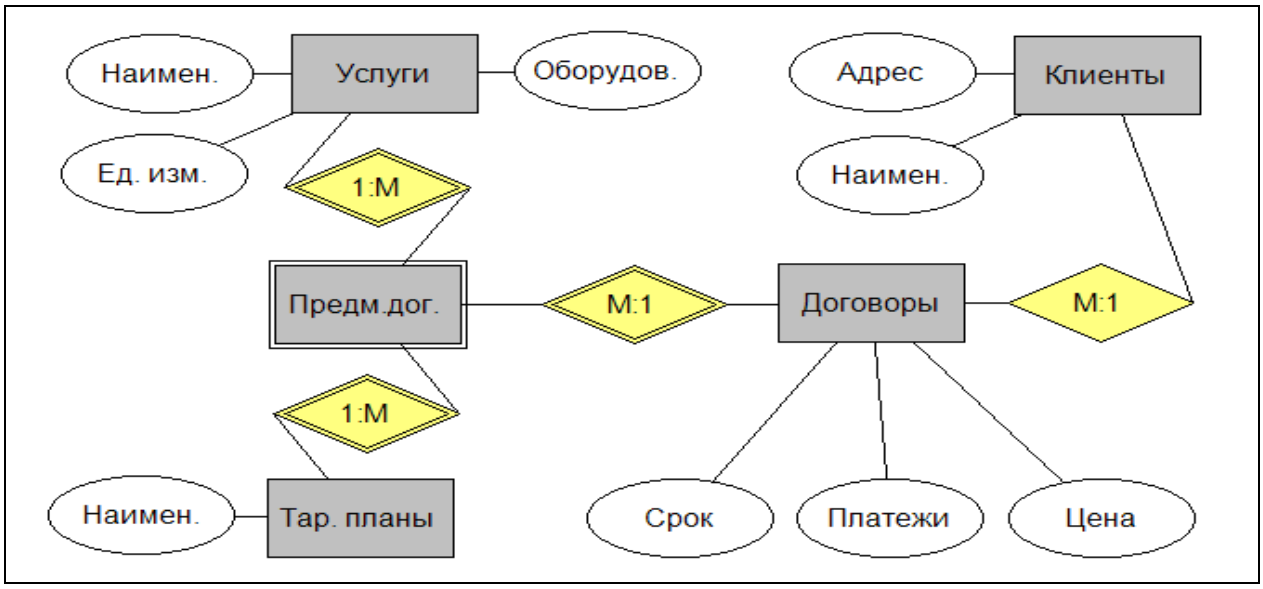


Рисунок 3.11 – ER-диаграмма пакета *Договоры с клиентами*

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.

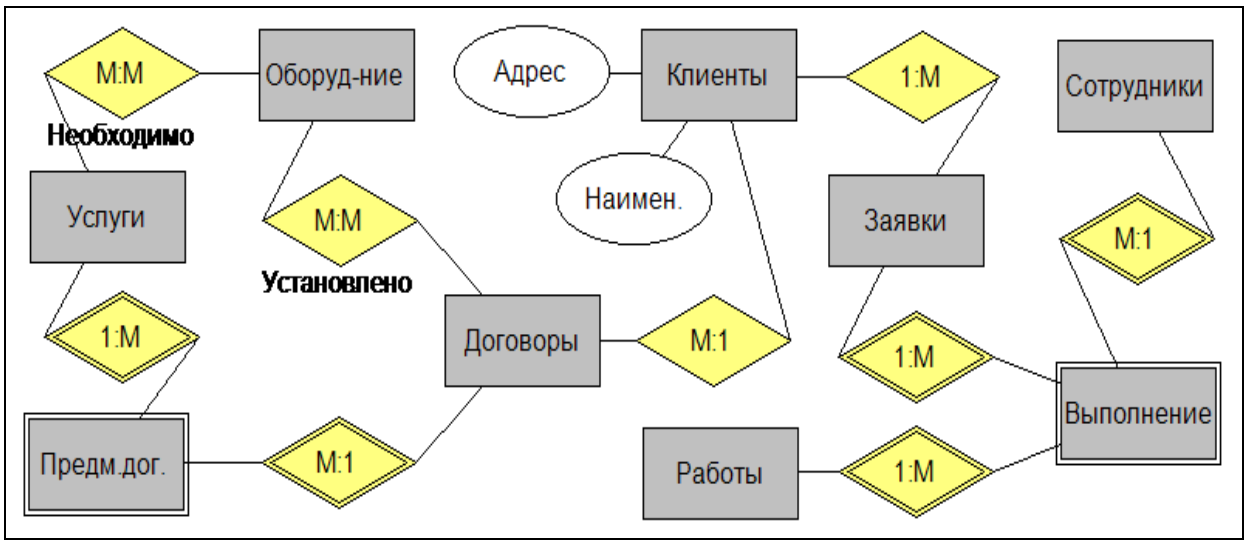


Рисунок 3.12 – ER-диаграмма пакета *Обработка заявок*

Результат объединения локальных ER-моделей в единую ER-модель предметной области проектируемой АИС представлен на рисунке 3.13.

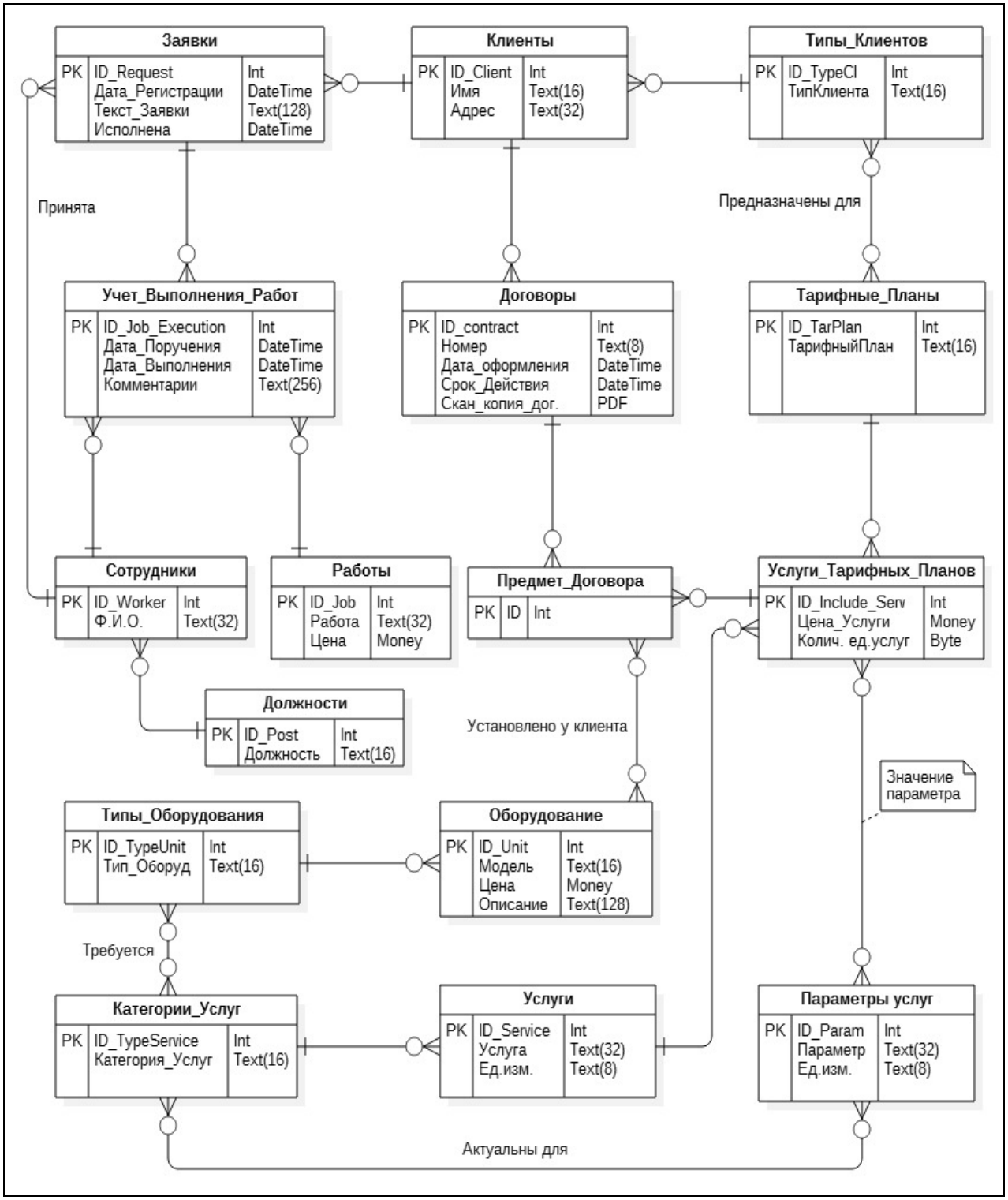


Рисунок 3.13 – Объединенная ER-диаграмма

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*» (рисунок 3.5).

3 Определите понятие «*слабая сущность*». Какую роль выполняют сущности «*Предмет договора*» и «*Услуги_Тарифных_Планов*» (рисунок 3.13)?

4 На рисунке 3.10 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.

3.3 Технический проект. Разработка реляционной модели данных

Реляционная модель (*R-модель*) относится к категории логических моделей данных и формируется на следующей стадии разработки базы данных – стадии технического проекта (рисунок 3.1). 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.

3.3.1 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 (п. 2.2):
  - 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) были 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 информации о связях между сущностями, представленной 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 (R-модель) 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.

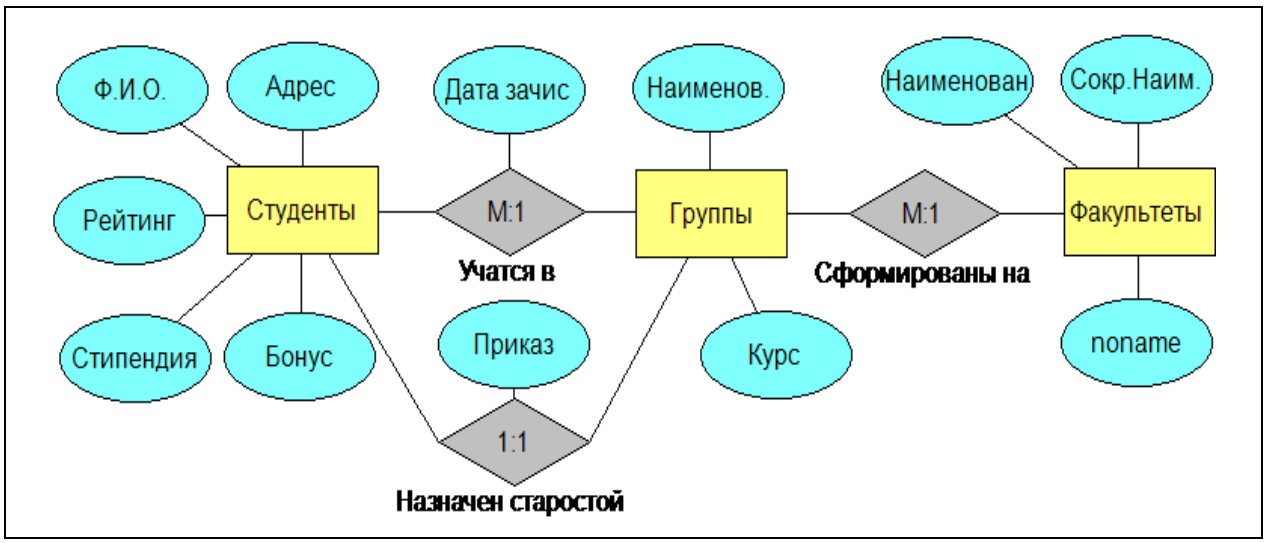
Внешний ключ дочернего отношения должен быть совместим по типу и домену с первичным ключом родительского отношения и может наследовать все его свойства, за исключением свойства уникальности. Значение внешнего ключа в кортежах дочернего отношения должно быть равным значению первичного ключа в связанном 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 (п. 3.2.3) между сущностями ER-модели в соответствии с тремя *базовыми правилами*.

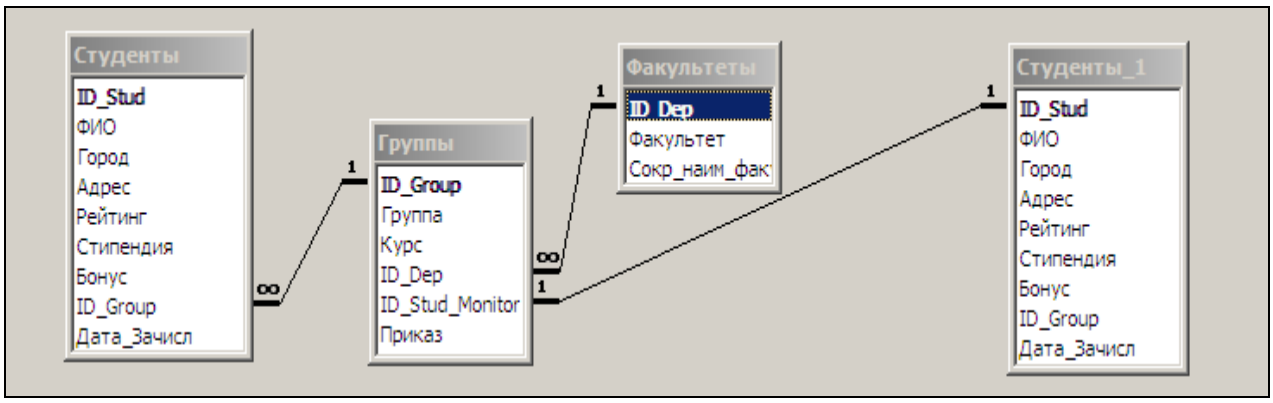
Правило №1. Если между парой сущностей ER-модели установлена асимметричная связь кратности ***1:M*** («*один ко многим*»), то дочерним считается отношение, реализующее сущность, участвующую в связи со стороны ***M*** («*много*»), а другое отношение в этой паре считается родительским.

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 кратности ***1:M*** и ***1:1*** между сущностями ER-модели иллюстрируются на примере модели системы учета контингента студентов (рисунок 3.14). 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

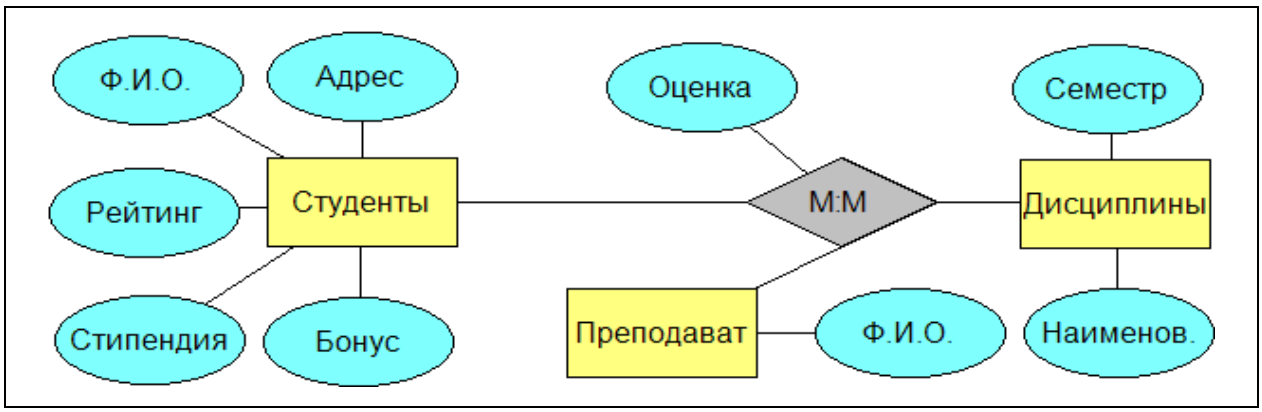
а) ER-диаграмма; б) схема реляционной БД

Рисунок 3.14 – Модель системы учета контингента студентов

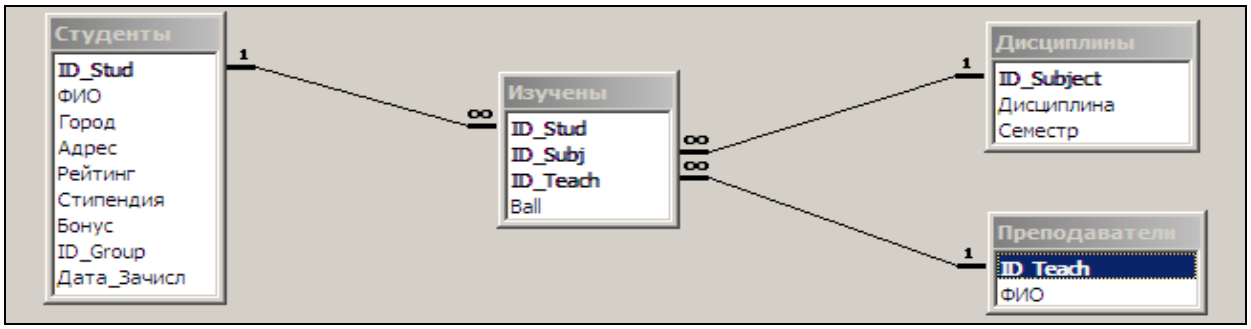
Правило №3. Если между ***n*** сущностями ER-модели установлена связь кратности ***N:M*** («*многие ко многим*»), то схема базы данных дополняется ***n***-арным *ассоциативным* отношением, которое получает статус дочернего отношения со схемой, содержащей ***n*** внешних ключей, а все ***n*** отношений, 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 3.15 на примере модели системы учета успеваемости студентов.



а



б

а) ER-диаграмма; б) схема реляционной БД

Рисунок 3.15 – Модель системы контроля успеваемости студентов

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 может 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 3.16, пред-

ставляющем 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 (рисунок 3.16 б).

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

Между экземплярами сущности «*Категории*» (рисунок 3.16 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 (рисунок 3.15 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*.

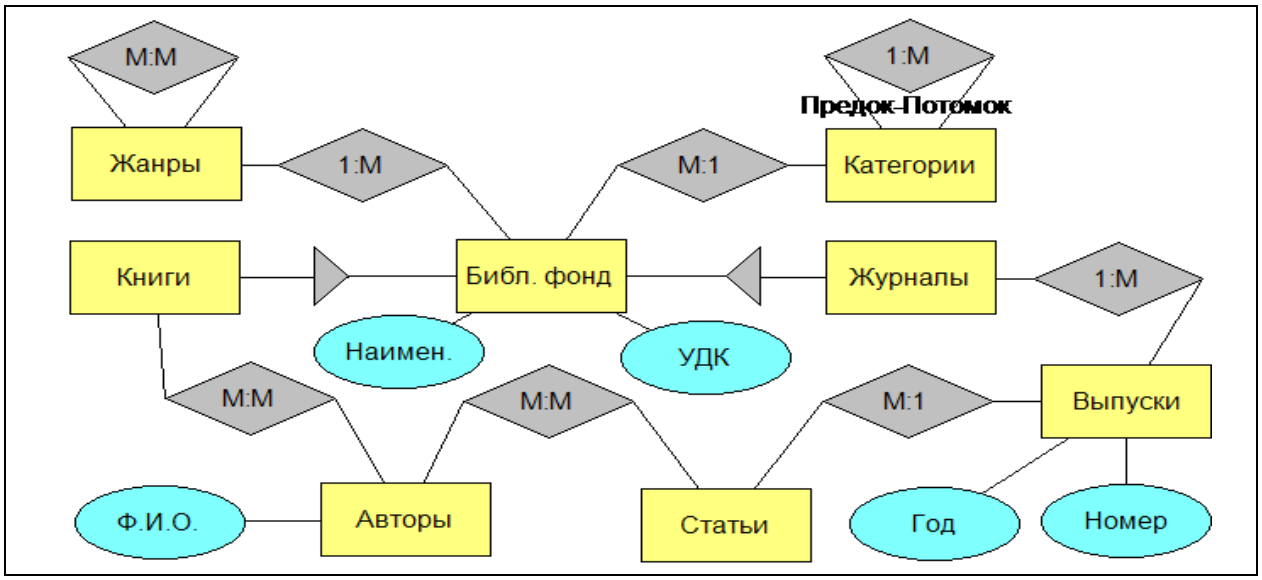
В качестве примера реализации унарной связи множественного наследования можно рассмотреть сущность *Жанры* (рисунок 3.16 а). Кратность этой связи определена, как ***M:N***, что дает возможность любому жанру выступать как в роли потомка, так и в роли предка любого количества других жанров. Например, жанр «*Фэнтэзи*» 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 отношения, схема которого сформирована из двух атрибутов – внешних ключей, представляющих первичные ключи двух связываемых кортежей основного отношения.

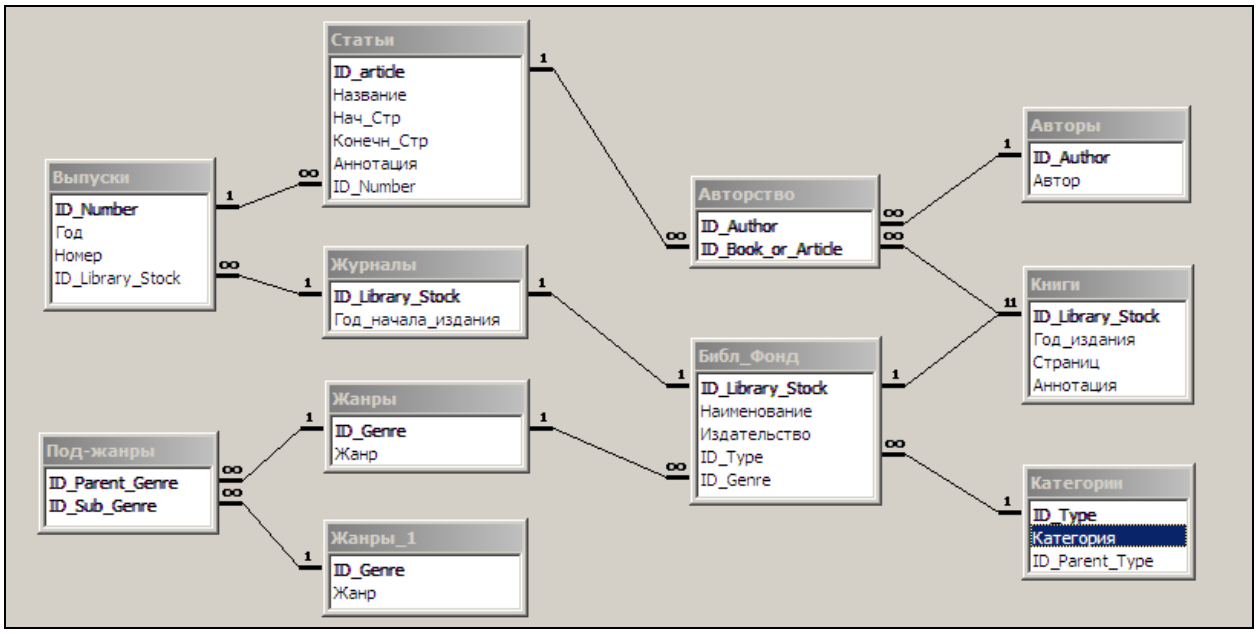
Так, например, схема ассоциативного отношения «*Под_жанры*» (рисунок 3.16 б) включает два таких атрибута – «*Код_жанра_предка*» и «*Код_жанра_потомка*».

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 (рисунки 3.15 и 3.16).



а



б

а) ER-диаграмма; б) схема реляционной БД

Рисунок 3.16 – Модель библиотечного каталога

3.3.2 Пример разработки исходной схемы реляционной БД

В качестве примера продолжим рассмотрение подсистемы учета работы с клиентами интернет-провайдера, ER-диаграмма которой приведена на рисунке 3.13. Применяя рассмотренные выше *правила преобразования* к сущностям и связям этой ER-модели, получим R-модель, представленную на рисунке 3.17.

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 (рисунок 3.16)?

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 3.17?

6 Задание: напишите выражения реляционной алгебры, реализующие следующие запросы к базе данных, схема которой приведена на рисунке 3.17.

- 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 3.10 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.

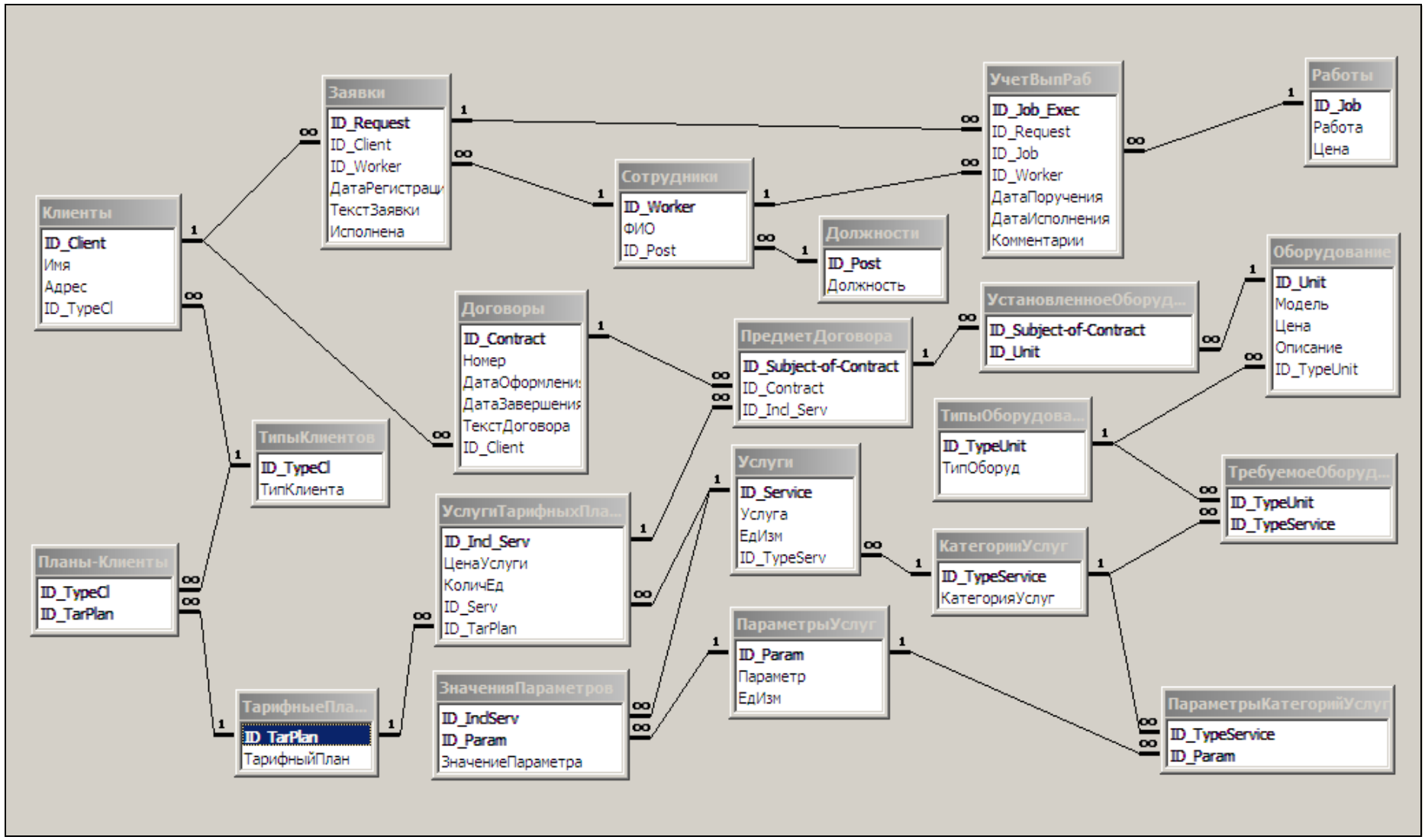


Рисунок 3.17 – Исходная R-модель подсистемы учета работы с клиентами интернет-провайдера

3.3.3 Нормализация реляционной базы данных

3.3.3.1 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 реляционной модели (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 атрибутам 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 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 нескольких десятках 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 3.15 а.

Схема реляционной БД (рисунок 3.15 б), сформированная на основе этой ER-модели, лишена рассмотренных выше недостатков первоначальной схемы, что избавит БД от многих аномалий в процессе ее эксплуатации.

3.3.3.2 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 – *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. 3.3.3.3) – 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. 2.3.1) на соответствующие подмножества атрибутов.

Гарантией сохранения информационной адекватности схемы нормализованной базы данных является строгое следование *правилу декомпозиции без потерь* (п. 3.3.3.5) 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. 3.3.3.3 и п. 3.3.3.4), по результатам которого определяется, в какой из НФ находится это отношение (п. 3.3.3.6);
2) отношения, находящиеся в сильных НФ (обычно это НФБК и выше), сохраняют свои исходные схемы;
3) каждое отношение, находящееся в слабой НФ, декомпозируются на несколько взаимосвязанных отношений с учетом *правила декомпозиции без потерь* (п. 3.3.3.5);
4) для каждого из отношений, полученных на 3-м этапе нормализации, выполняются этапы с 1-го по 2-й до тех пор, пока в схеме БД не останется отношений, находящихся в слабых НФ.

3.3.3.3 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 учебном пособии 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 3.18.

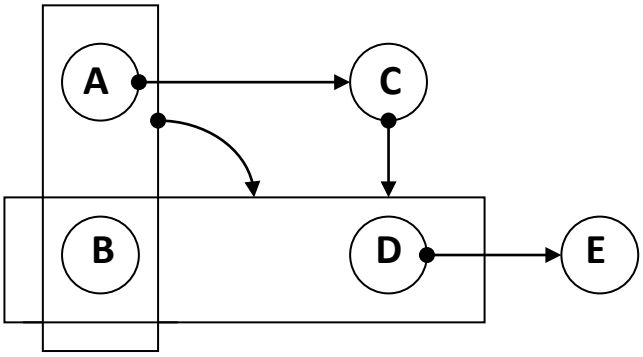


Рисунок 3.18 – Графические обозначения функциональных зависимостей
A → C; (A,B) → (B,D); (C → B,D); D → E

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 рассматриваемого 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*.

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 $\mathbf{A}\rightarrow(\mathbf{B},\mathbf{C})$, то существуют ФЗ $\mathbf{A}\rightarrow\mathbf{B}$ и $\mathbf{A}\rightarrow\mathbf{C}$. Доказательство. Согласно *правилу рефлексивности* существует ФЗ $(\mathbf{B},\mathbf{C})\rightarrow\mathbf{B}$, следовательно, по *правилу транзитивно-*

сти, 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 *псевдоранзитивность*): 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**).

3.3.3.4 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.

1 Правая (зависимостная) часть любой ФЗ минимального покрытия является простым (не составным) атрибутом.
2 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.
3 Удаление любой ФЗ из минимального покрытия порождает множество ФЗ, не эквивалентное их исходному набору.

3.3.3.5 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: **R1 := R** PROJECT **(A,B)** и **R2 := R** PROJECT **(B,C)**, связанные по их общим атрибутам **R1.B** и **R2.B**.

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 3.19.

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 подмноже-

ства 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**.

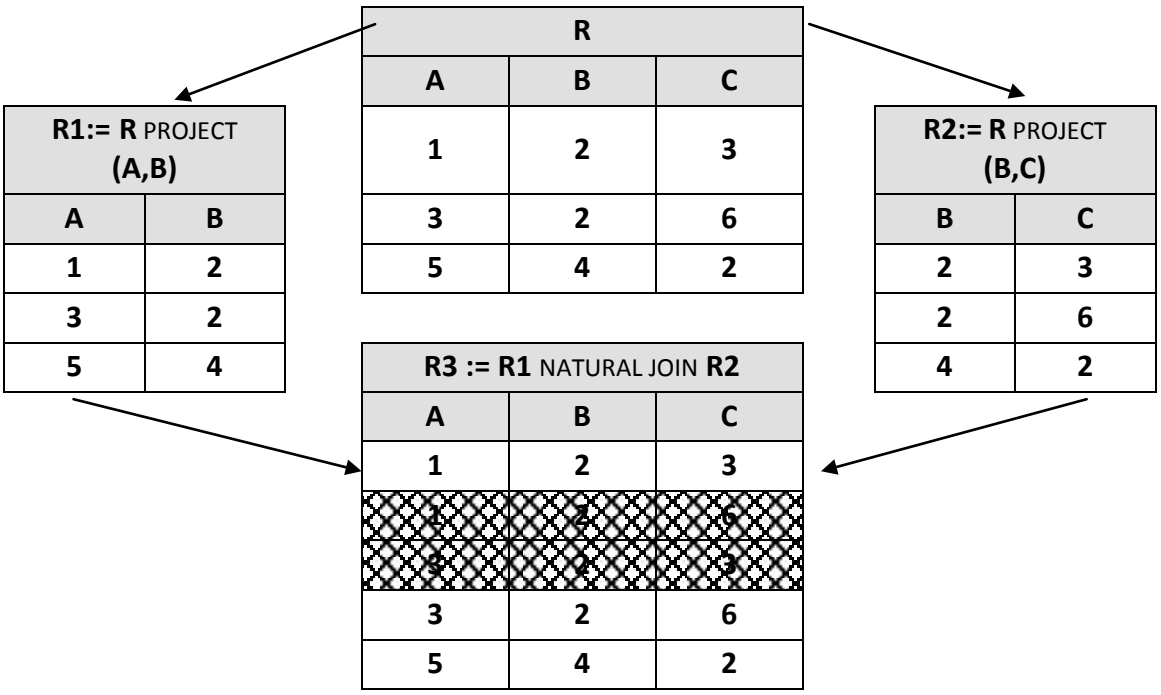


Рисунок 3.19 – Пример некорректной (необратимой) декомпозиции

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** (рисунок 3.19), 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**.

В другом примере (рисунок 3.20) 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**.

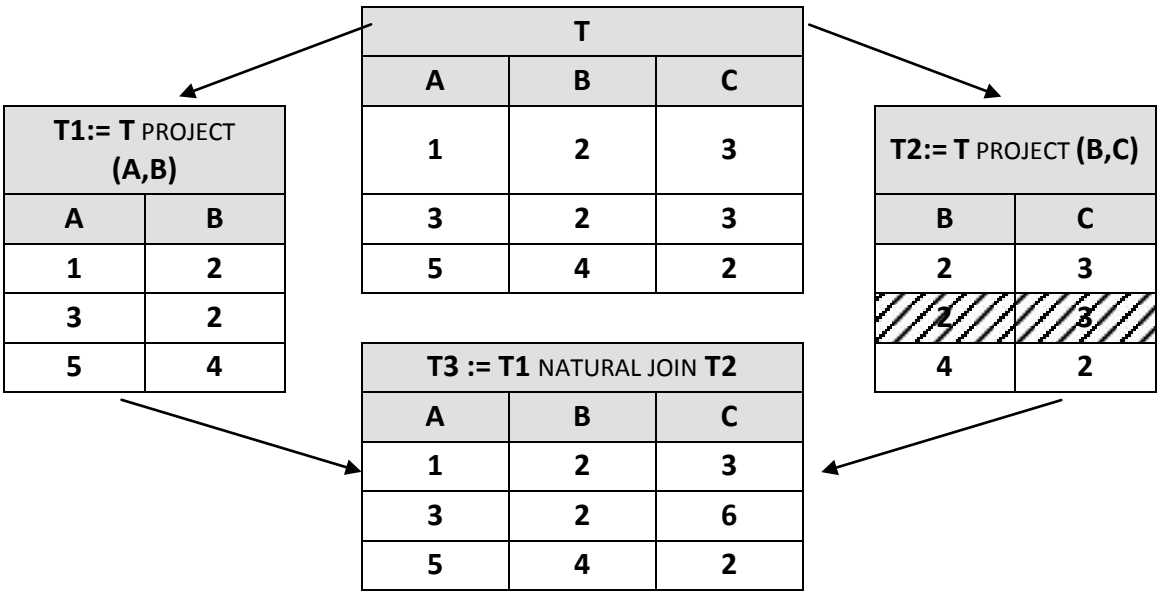


Рисунок 3.20 – Пример декомпозиции без потерь

3.3.3.6 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**.

3.3.4 Пример нормализации реляционной базы данных

Продолжим рассмотрение модели экзаменационной ведомости, использованной ранее (п. 3.3.3.1) 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. 3.3.3.4), исключив из исходного набора избыточные ФЗ, которые могут быть получены из других ФЗ применением к ним соответствующих *правил вывода* (п. 3.3.3.3).

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.*)** $\rightarrow$ **(*Дата, Препод.*)**:

ФЗ-5.1 **–** **(*Сем., Группа., Дисц.*)** $\rightarrow$ ***Дата;***

ФЗ-5.2 **–** **(*Сем., Группа., Дисц.*)** $\rightarrow$ ***Препод.***

Декомпозиция **ФЗ-6**: **(*Студ., Дисц., Сем.*)** $\rightarrow$ **(*Дата, Препод., Оценка*)**:

ФЗ-6.1 – **(*Студ., Дисц., Сем.*)** $\rightarrow$ ***Дата***;

ФЗ-6.2 – **(*Студ., Дисц., Сем.*)** $\rightarrow$ ***Препод.***;

ФЗ-6.3 – **(*Студ., Дисц., Сем.*)** $\rightarrow$ ***Оценка***.

Зависимости **ФЗ-4** и **ФЗ-5.2** – являются избыточными, так как могут быть выведены из **ФЗ-6.2** по *правилу рефлексивности* – 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.*** $\rightarrow$ ***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.*** (*определение 7*), а также наличие *неполной транзитивной зависимости* между атрибутами ***Студ.*** → ***Группа*** → ***Спец.*** (*определение 8*).

Для приведения БД ко второй нормальной форме 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 ключом 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 дублирует 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 3.21.

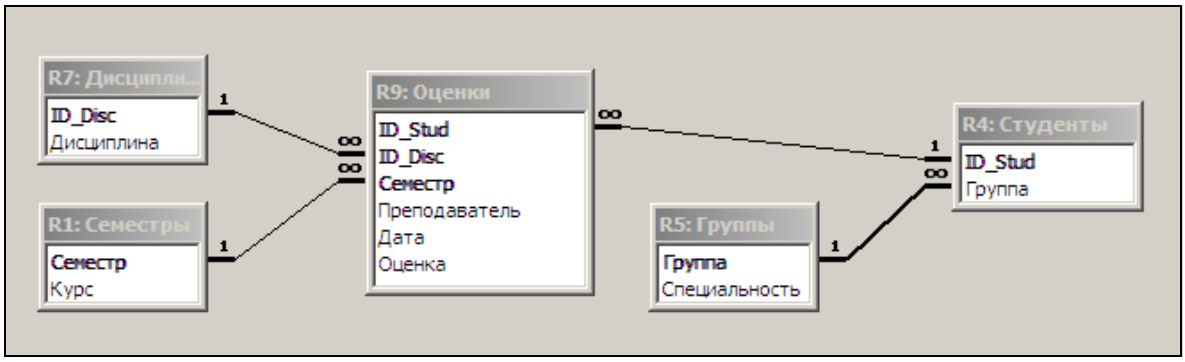


Рисунок 3.21 – Схема БД, полученная в результате нормализации исходного отношения **R_0**

Завершая рассмотрение технологий проектирования реляционных баз данных, попытаемся сравнить два подхода к реализации этих технологий, долгое время развивавшиеся 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 3.21, 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 неё отношений к одной из сильных нормальных форм. В большинстве случаев оказывается достаточной НФБК, гарантирующей отсутствие перечисленных выше аномалий, однако не исключаю-

щей 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.

3.4 Проектный практикум

3.4.1 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 выполняется студентом 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.

3.4.2 Содержание практических занятий

Основная часть работы над проектом выполняется студентом 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. 3.2.5 и п. 3.3.4 учебного пособия;
- 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.

- демонстрация и анализ результатов проектирования и программной реализации.

3.4.3 Типовые варианты тем учебных проектов

<table>
<tr><td colspan="3">Тема №1. УПРАВЛЕНИЕ РАБОТАМИ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– руководители;
– менеджеры;
– исполнители;
– клиенты (заказчики)</td><td>– 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</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>1.1</td><td>Управление проектами</td><td>– структуризация проектов;
– специализация исполнителей;
– повышение квалификации и профессиональная переподготовка сотрудников</td></tr>
<tr><td>1.2</td><td>Интернет-провайдер</td><td>– call-центр и тех.поддержка клиентов;
– услуги и тарифные планы;
– категории клиентов;
– оборудование, установленное у клиентов</td></tr>
<tr><td>1.3</td><td>Компьютер-сервис</td><td>– прайс-лист (сервис, ремонт, установка и настройка программного обеспечения);
– прием заявок;
– выездные работы;
– доставка оборудования</td></tr>
<tr><td>1.4</td><td>Малое промышленное предприятие
(по отраслям –
по выбору студента)</td><td>– номенклатура изделий;
– технологическая документация;
– складской учет материалов и комплектующих;
– складской учет готовой продукции</td></tr>
<tr><td>1.5</td><td>Фермерское сельхозпредприятие
(по отраслям –
по выбору студента)</td><td>– номенклатура производства;
– учет основных фондов;
– договоры с заказчиками и поставщиками;
– складской учет готовой продукции</td></tr>
</table>

<table>
<tr><td colspan="3">Тема №2. УПРАВЛЕНИЕ ОБРАЗОВАНИЕМ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– руководители;
– преподаватели
– (учителя);
– технические
– сотрудники;
– студенты (учащиеся, слушатели)</td><td>– управление контингентом студентов (учащихся, слушателей);
– кадровый учет преподавательского состава;
– планирование;
– распределение и контроль исполнения работ;
– формирование аналитической и отчетной документации</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>2.1</td><td>Электронный дневник школьника</td><td>– 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</td></tr>
<tr><td>2.2</td><td>Распределение годовой учебной нагрузки преподавателей вуза</td><td>– 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</td></tr>
<tr><td>2.3</td><td>Мониторинг успеваемости студентов</td><td>– 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</td></tr>
<tr><td>2.4</td><td>Центр профессиональной переподготовки специалистов</td><td>– управление образовательными программами;
– бухгалтерский учет (прием платежей от слушателей, оплата работы преподавателей);
– регистрация выдачи сертификатов.</td></tr>
<tr><td>2.5</td><td>Администрирование компьютерных классов</td><td>– оборудование и ПО рабочих мест;
– закрепление учебных дисциплин;
– расписание занятий в компьютерных классах;
– загруженность классов в течение семестра</td></tr>
</table>

<table>
<tr><td colspan="3">Тема №3. АВТОМАТИЗИРОВАННЫЕ БИБЛИОТЕЧНЫЕ СИСТЕМЫ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– библиотекари;
– читатели</td><td>– учет движения библиотечного фонда (поступление, списание);
– поиск объектов библиотечного фонда;
– регистрация читателей;
– регистрация выдачи/возврата;
– анализ читательского спроса</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>3.1</td><td>Абонемент публичной библиотеки</td><td>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</td></tr>
<tr><td>3.2</td><td>Абонемент университетской библиотеки</td><td>– 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</td></tr>
<tr><td>3.3</td><td>Читальный зал периодических изданий</td><td>– 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</td></tr>
<tr><td>3.4</td><td>Читальный зал научно-технических изданий</td><td>– 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</td></tr>
</table>

<table>
<tr><td colspan="3">Тема №4. СПОРТИВНЫЕ СОРЕВНОВАНИЯ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– участники;
– болельщики;
– руководители;
– администраторы;
– аналитики</td><td>– регистрация участников;
– планирование;
– оперативный учет результатов;
– анализ</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>4.1</td><td>Командные спортивные соревнования (вид спорта – по выбору разработчика)</td><td>– 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</td></tr>
<tr><td>4.2</td><td>Командные спортивные соревнования (вид спорта – по выбору разработчика)</td><td>– несколько спортивных лиг;
– один спортивный сезон;
– регистрация и учет состояния спортивных арен;
– регистрация команд – участников спортивных соревнований;
– регистрация спортсменов – участников команд;
– рейтинги</td></tr>
<tr><td>4.3</td><td>Индивидуальные спортивные соревнования (вид спорта – по выбору разработчика)</td><td>– одна спортивная лига;
– хранение истории нескольких спортивных сезонов;
– регистрация спортсменов – участников соревнований в каждом сезоне;
– рейтинги спортсменов с учетом их личных достижений</td></tr>
<tr><td>4.4</td><td>Индивидуальные спортивные соревнования (вид спорта – по выбору разработчика)</td><td>– несколько спортивных лиг по одному виду спорта;
– один спортивный сезон;
– регистрация спортсменов – участников спортивных соревнований;
– рейтинги спортсменов с учетом их личных достижений</td></tr>
</table>

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 спорт. залов; – формирование групп; – контроль посещения занятий и прием платежей от клиентов; – финансовая отчетность
5.4	Горно-лыжный 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

<table>
<tr><td colspan="3">Тема №6. ТРАНСПОРТ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– гости;
– клиенты;
– руководители;
– менеджеры</td><td>– формирование прайс-листа;
– кадровый учет;
– регистрация клиентов;
– оперативный учет;
– анализ</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>6.1</td><td>Автосалон</td><td>– поиск автомобиля по маркам, моделям, комплектациям, ценам;
– управление продажами:
• продажи автомобилей с пробегом;
• подбор комплектации и продажа новых автомобилей;
• оформление договоров с клиентами;
– Trade IN (продажа нового автомобиля с одновременной покупкой автомобиля клиента)</td></tr>
<tr><td>6.2</td><td>Прокат автомобилей</td><td>– поиск автомобиля по категориям, маркам, моделям, ценам, типам двигателя и кузова;
– регистрация клиентов;
– регистрация парка автомобилей;
– выдача/возврат;
– финансовый анализ.</td></tr>
<tr><td>6.3</td><td>Прокат велосипедов</td><td>– размещение велопарковок (в черте города);
– выбор велосипеда по типам, моделям;
– наличие велосипедов на велопарковках;
– выдача/возврат;
– финансовый анализ</td></tr>
<tr><td>6.4</td><td>Агрегатор такси</td><td>– адресный справочник;
– регистрация водителей и автомобилей;
– прием заказов;
– расчет стоимости поездки;
– учет исполнения заказов;
– финансовый учет и анализ</td></tr>
<tr><td>6.5</td><td>Автовокзал</td><td>– автопарк;
– междугородние маршруты;
– расписание рейсов;
– продажа билетов</td></tr>
</table>

<table>
<tr><td colspan="3">Тема №7. ТОРГОВО-СКЛАДСКОЙ УЧЕТ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– гости;
– клиенты;
– кладовщики;
– продавцы;
– менеджеры</td><td>– классификация товаров по категориям;
– формирование прайс-листа;
– кадровый учет;
– регистрация клиентов;
– учет поставок и отпуска товаров</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>7.1</td><td>Оптовый склад продуктов питания</td><td>– учет предельных сроков реализации товаров;
– списание просроченных товаров;
– персональные скидки постоянным клиентам</td></tr>
<tr><td>7.2</td><td>Универсальный интернет-магазин</td><td>– формирование и контроль исполнения заказов;
– доставка товаров покупателям</td></tr>
<tr><td>7.3</td><td>Магазин по продаже компьютерной и оргтехники</td><td>– контроль совместимости комплектующих;
– комплектование товаров по заявкам покупателей</td></tr>
<tr><td>7.4</td><td>Мебельный магазин</td><td>– комплектование товаров;
– выполнение дизайн-проектов;
– доставка товаров покупателям;
– сборка товаров у покупателей</td></tr>
<tr><td colspan="3">Тема №8. ОБЩЕСТВЕННОЕ ПИТАНИЕ</td></tr>
<tr><td colspan="2">Пользователи</td><td>Автоматизируемые бизнес-процессы</td></tr>
<tr><td colspan="2">– гости;
– клиенты;
– официанты;
– менеджеры</td><td>– классификация блюд и напитков по категориям;
– формирование меню, состав блюд;
– кадровый учет;
– прием и учет исполнения заказов</td></tr>
<tr><td colspan="2">Варианты заданий</td><td>Дополнительные требования</td></tr>
<tr><td>8.1</td><td>Ресторан</td><td>– специализация ресторана («кухня»);
– предварительные заказы;
– банкеты</td></tr>
<tr><td>8.2</td><td>Диетическое питание</td><td>– контроль состава блюд;
– рецепты приготовления блюд;
– расчет калорийности блюд</td></tr>
</table>

Глава 4
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 во второй части данного учебного пособия.

4.1 Язык SQL – базовые синтаксические конструкции

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.

4.1.1 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 (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 4.1.

```
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 4.1 – 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.

4.1.2 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 DDL-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 во второй части данного учебного пособия, листинг 4.2 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 4.2 – 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 распространяется (CASCADE) на все субъекты доступа, получившие данные права от пользователя BigClerk;

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

4.1.3 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 её 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 которых 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 4.3-4.6), использующими учебную базу данных, схема которой представлена на рисунке 4.2.

4.1.3.1 Простейшие SQL-запросы

Листинг 4.3 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 4.3 – Примеры простейших «однотабличных» SQL-запросов

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

1 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.

2 Раздел WHERE реализует редяционно-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.

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

4 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 со следующей таблицей истинности (таблица 4.1).

Таблица 4.1 – 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**

5 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.

6 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.

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

8 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.

4.1.3.2 SQL-запросы с соединением (JOIN) таблиц

Листинг 4.4 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 4.4 – Примеры SQL-запросов с соединениями таблиц

Комментарии к примерам, приведенным в листинге 4.4:

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 соответствующие 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.

4.1.3.3 SQL-запросы с объединением 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 – 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 прини-

маются решения «по умолчанию» (например, имена столбцов объединенной таблицы наследуются от имен столбцов первой из объединяемых).

Листинг 4.5 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 4.5 – Пример использования оператора UNION

4.1.3.4 SQL-операторы INSERT, DELETE и UPDATE

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.

Синтаксические правила использования этих операторов интуитивно понятны и иллюстрируются приведенными ниже примерами (листинг 4.6):

```
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 4.6 – Примеры использования операторов INSERT, DELETE и UPDATE

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 помеща-

ется результат выборки информации о товарах, включенных в заказы 2018 года;

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 завершению 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. 4.1.3.6 (листинги 4.9 и 4.10).

4.1.3.5 Использование хранимых представлений

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 базовых используемых в представлении таблиц, в котором они находились в момент выполнения представления.

Ссылки на имена представлений и на 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 4.7 содержит пример использования хранимых представлений.

```
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 4.7 – Пример использования представлений в SQL-запросах

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.

4.1.3.6 Использование подчиненных запросов

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 4.8 – 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 4.8 – 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*.

Примеры *а)* и *б)* в листинге 4.9 представляют два синтаксически различных, но семантически эквивалентных SQL-запроса, производящих выборку из таблицы Сотрудники имен всех сотрудников, не оформивших ни одного заказа в 2017 году.

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*.

```
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 4.9 – Примеры использования подчиненных запросов в предикатах сравнения EXISTS и IN

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

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 4.10 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.

```
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  За-
казы
    WHERE ДатаИсполнения >
    DateAdd("m",1,ДатаРазмещения));
г)  DELETE   FROM Поставщики
    WHERE Код_Поставщика = ALL (
          SELECT Код_Поставщика FROM Склад
          WHERE ПоставкиПрекращены=True);
```

Листинг 4.10 – Примеры использования подчиненных запросов в предикатах сравнения ANY|SOME и ALL

4.1.3.7 SQL-средства групповой обработки данных

Рассмотренные выше примеры использования DML-операторов демонстрируют возможности построчной обработки табличных данных: со-

единение строк таблиц, выборка отдельных строк по заданным критериям и их последующая обработка.

Язык SQL содержит также 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 используются разделы GROUP BY и HAVING оператора SELECT, а для вычисления их статистических характеристик – так называемые *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 осмысленно 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 *усло-*

вие 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 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*) – количество строк в группе;
- MAX(*p*) – максимальное значение параметра *p*;
- MIN(*p*) – минимальное значение параметра *p*;
- SUM(*p*) – суммарное значение параметра *p*;
- AVG(*p*) – 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.

Агрегатные функции могут использоваться в разделах SELECT и/или HAVING оператора SELECT, а также в подчиненных запросах, как это показано в примерах листинга 4.11.

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.

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

```
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 4.11 – Примеры групповой обработки данных

В запросе *а)* из исходной таблицы производится *выборка строк* (WHERE), соответствующих заказам 2017 года, 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 года, и вычисляется количество строк для каждой из этих групп.

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.

4.2 Стандарты и диалекты языка SQL

4.2.1 Этапы стандартизации языка SQL

Язык SQL (под названием SEQUEL – *Structured English QUEry Language*) был разработан корпорацией IBM в середине 70-х годов 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 80-х годов 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 году 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 объ-

ектных 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 года 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 лю-

бого 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 формируется 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 года является улучшение и развитие средств работы с временными (*temporal*) базами данных, в кото-

рых хранится информация, актуальная для определенных временных периодов.

В *битемпоральных* базах данных могут быть определены два таких 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.

4.2.2 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 4.2 приведены фирменные наименования некоторых из широко распространенных диалектов SQL.

Таблица 4.2 – 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.

4.2.3 Microsoft Jet SQL

Основная причина рассмотрения в учебном пособии особенностей именно этого диалекта языка SQL – его относительная простота, что не маловажно 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 дополнительные 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 Различные наборы ***типов данных***

В таблице 4.3 перечислены типы данных SQL-89, эквивалентные им типы данных языка Microsoft Jet SQL и допустимые синонимы для их именования.

Таблица 4.3 – 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

3 ***Подстановочные символы*** предиката **LIKE** (таблица 4.4)

Таблица 4.4 – Подстановочные символы предиката сравнения LIKE

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

4 Оператор ***TRANSFORM***

Оператор TRANSFORM предназначен для создания так называемых ***перекрестных запросов***, результат выполнения которых представляется пользователю 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 условий 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 4.12 содержит пример перекрестного запроса, представляющего информацию о суммах выручки, полученных от реализации товаров, в координатах СОТРУДНИК – ТОВАР, а рисунок 4.1 – 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 4.12 – Пример перекрестного запроса

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

Рисунок 4.1 – Результат выполнения перекрестного запроса, представленного на листинге 4.12

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

Дополнительно к пяти стандартным агрегатным функциям язык Microsoft Jet SQL включает функции StDev(*выражение*) и StDevP(*выражение*), возвращающие, соответственно, смещенное и несмещенное значения среднеквадратичного отклонения, вычисляемого по набору значений, содержащихся в *выражении*.

Аргумент *выражение* может быть именем столбца, содержащего обрабатываемые данные числового типа, или выражением, операндами которого могут быть имена столбцов, числовые константы или (не статистические) 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 (таблица 4.2).

В следующем примере (листинг 4.13) рассчитывается сумма выручки от заказов, исполненных сотрудниками филиалов, у пользователя запра-

шиваются значения трех 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 4.13 – Пример использования параметризованного SQL-запроса

4.3 Практикум по SQL-программированию

4.3.1 Общие методические указания

Основные задачи проведения практических занятий – 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 (п. 4.3.3), каждое из которых связано с написанием и отладкой некоторого SQL-запроса.

Все задания выполняются в учебной базе данных, схема которой приведена на рисунке 4.2. 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 изучить материал, изложенный в п. 4.1.3 и п. 4.2.3 учебного пособия, а также проанализировать и программно реализовать SQL-запросы, примеры которых приведены на листингах 4.3 – 4.13.

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

4.3.2 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 4.2.

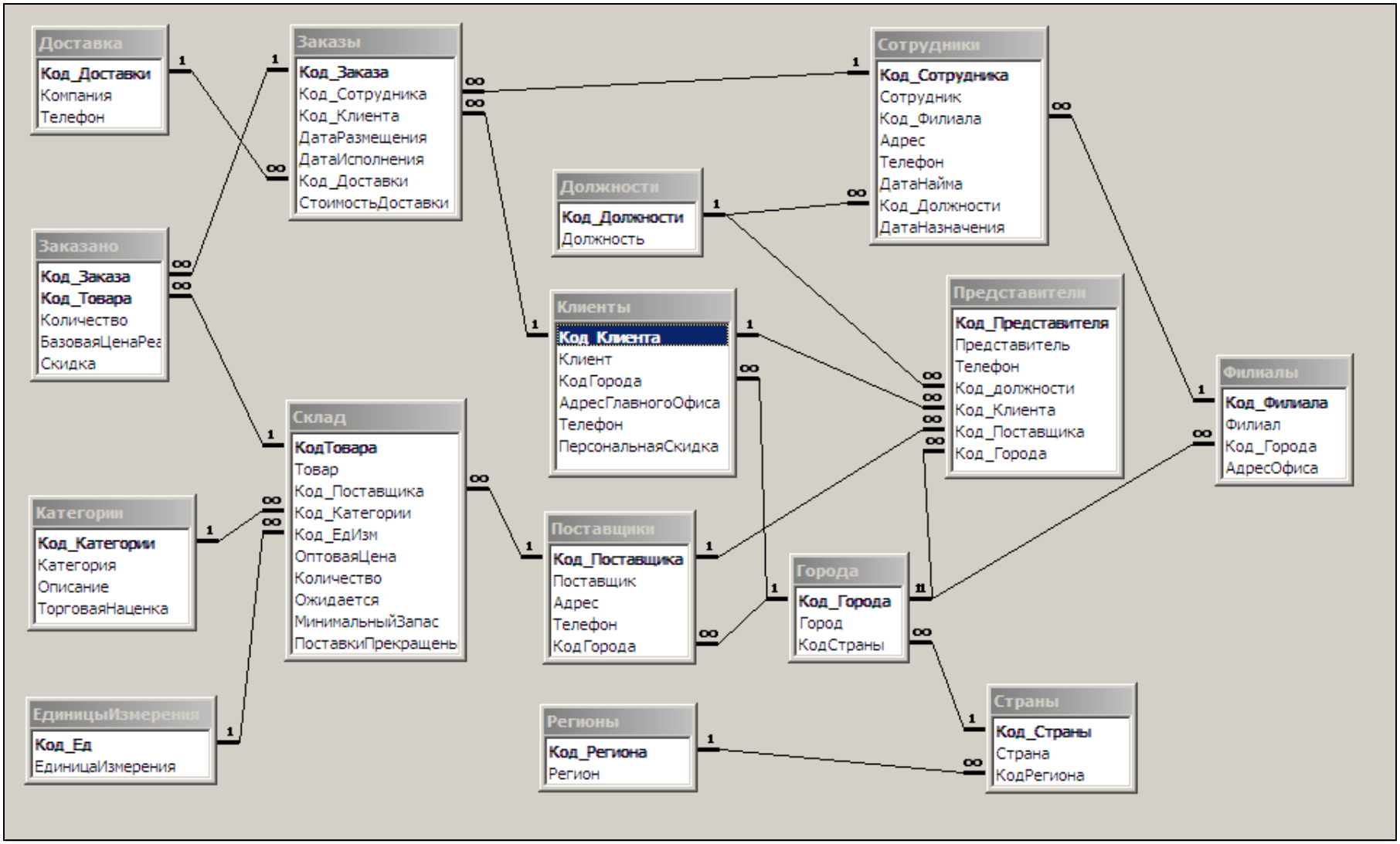


Рисунок 4.2 – Схема учебной базы данных

4.3.3 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 Прокомментируйте следующие SQL-запросы и оцените результаты их выполнения (листинг 4.14):

```
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 4.14 – 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 Прокомментируйте следующие SQL-запросы и оцените результаты их выполнения (листинг 4.15):

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.

```
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 4.15 – Примеры SQL-запросов с групповой обработкой данных

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 года.

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).

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 Объедините 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.

Глава 5
УПРАВЛЕНИЕ БАЗАМИ ДАННЫХ

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 (рисунок 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 этого учебного пособия – 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 (в СУБД MS SQL Server) функций поддержки физической модели данных, трансляции SQL-запросов и управления производительностью, а также функции защиты от несанкционированного доступа к данным, рассмотрены во второй части данного учебного пособия.

5.1 Обзор функций СУБД

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 называе-

мые *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 функционирования 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.

5.2 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.

5.2.1 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 будет детально рассмотрено в разделах 5.2.2 – 5.2.4 учебного пособия.

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.

5.2.2 Конфликты между транзакциями

Поддержка требования *согласованности* транзакций допускает рассогласованное состояние базы данных в процессе их выполнения, а реализация требований *атомарности* и *долговременности* должна обеспечить возможность отката (ROLLBACK) частично выполненных транзакций и, с другой стороны, должна 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 изменения, сделанные той транзакцией, которая завершилась последней, а все изменения в таблице, сделанные первой 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 ко-

торых 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.

5.2.3 Уровни изолированности транзакций

Стандарт SQL-92 определяет 4 уровня изолированности транзакций – от самого слабого нулевого уровня до самого сильного – третьего (таблица 5.1).

Таблица 5.1 – 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'а 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 далее в п. 5.2.5.

5.2.4 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 важ-

нейшим 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. 5.2.4.3) 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. 5.2.4.2) объекта в соответствии с требуемым уровнем изолированности транзакции;

- выбирает оптимальный *уровень блокируемого объекта* (п. 5.2.4.1) 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. 5.2.4.3), которые не могут быть разрешены естественным путем при освобождении завершенными транзакциями заблокированных ими объектов.

5.2.4.1 Уровни блокирования ресурсов

СУБД может блокировать объекты как логической, так и физической[5] моделей данных различных иерархических уровней, типичный набор которых приведен в таблице 5.2.

Таблица 5.2 – 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

[5] Объекты физической модели данных (файлы, экстенты, страницы, строки), а также индексы различных типов детально рассмотрены во второй части данного учебного пособия.

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.

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. 5.2.4.2 при рассмотрении специальных режимов блокирования – так называемых *блокировок с намерениями*.

5.2.4.2 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 (раздел 5.2.3). 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 не совместим 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 предварительно чтения данных из таблиц, связанных с модифицируемой таблицей (например, для анализа условий выборки модифицируемых строк), поэтому транзакция-писатель, кроме монопольной блокировки обновляемого объекта, 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. 5.2.4.4.

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 (таблица 5.3) 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 используют 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**.

5.2.4.3 Совместимость режимов блокирования

СУБД, принимая решения об установке или отклонении блокировок, запрашиваемых конкурирующими транзакциями, использует информацию о совместимости режимов блокирования (таблица 5.3). 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 5.3 – Совместимость режимов блокирования объектов

Запрашиваемый режим блокирования объекта / Установленный на объект режим блокирования		**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».

5.2.4.4 Тупиковые блокировки – прогнозирование и разрушение

Тупиковыми блокировками (или просто *тупиками* – *deadlock*) называют ситуации, когда две или более конкурирующие транзакции устанав-

ливают такие взаимные блокировки объекта, при которых ни одна из транзакций не может завершиться, пока другая не снимет с объекта своей блокировки (рисунок 5.1).

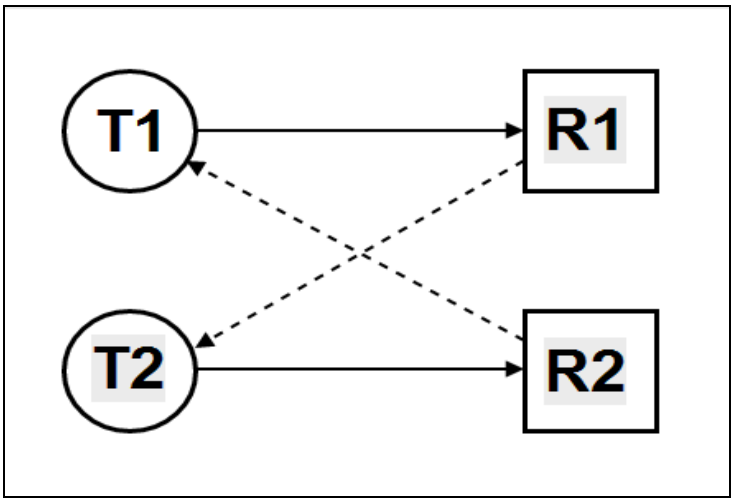


Рисунок 5.1 – Иллюстрация тупиковой блокировки

Установленные блокировки обозначены на рисунке сплошными линиями – дугами графа, направленными от узлов-транзакций к узлам-объектам, а противоположно направленные (пунктирные) дуги соответствуют блокировкам, 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 (таблица 5.3) 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 5.2.

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 – 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.

Попробуем применить этот алгоритм к графу ожидания транзакций, представленному на рисунке 5.1.

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 5.2 – три транзакции конкурируют в доступе к четырем объектам базы данных.

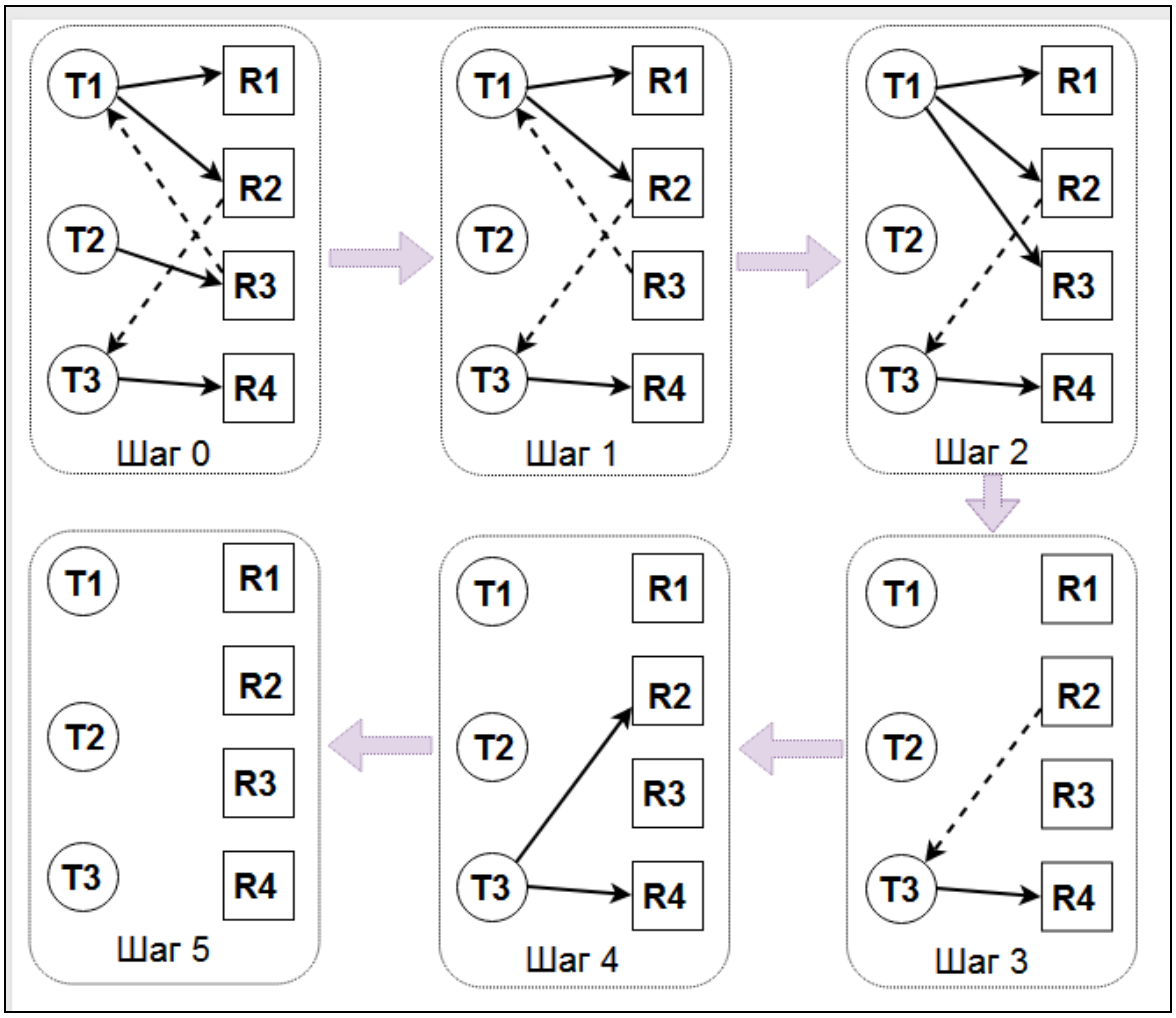


Рисунок 5.2 – Пример реализации
алгоритма редукции графа ожидания транзакций

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.

5.2.5 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).

5.2.5.1 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 5.1.

Единственный параметр этой инструкции – требуемый уровень изолированности (обзор уровней изолированности транзакций, за исключением нестандартного уровня SNAPSHOT, приведен в разделе 5.2.3).

```
SET TRANSACTION ISOLATION LEVEL
  {READ UNCOMMITTED
  | READ COMMITTED
  | REPEATABLE READ
  | SNAPSHOT
  | SERIALIZABLE
  }
```

Листинг 5.1 – Формат SQL-инструкции, используемой для установки уровня изолированности транзакций

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 5.2.

Перечни хинтов и комментарии к их использованию для локального управления уровнями изолированности транзакций и уровнями блокирования объектов БД приведены в таблицах 5.4 и 5.5.

```
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 5.2 – Пример использования *хинтов*

Таблица 5.4 – 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 5.5 – 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

5.2.5.2 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 5.3 – 5.5.

Начало транзакции

```
BEGIN { TRAN | TRANSACTION }
  [ { transaction_name |
    @tran_name_variable }
  [ WITH MARK [ 'description' ] ] ]
```

Листинг 5.3 – 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 5.8).

Фиксация транзакции

```
COMMIT [ { TRAN | TRANSACTION }
[ transaction_name |
@tran_name_variable ]]
[ WITH ( DELAYED_DURABILITY = { OFF | ON }
) ]
```

Листинг 5.4 – 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 5.8).

Точки сохранения

Инструкция SAVE TRANSACTION (листинг 5.5) устанавливает внутри транзакции *точку сохранения* – именованный маркер, к которому можно выполнить частичный откат транзакции инструкцией ROLLBACK TRANSACTION (листинг 5.6). Таких именованных точек внутри транзакции может быть несколько.

```
SAVE { TRAN | TRANSACTION }
{ savepoint_name |
@savepoint_variable }
```

Листинг 5.5 – 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 5.6) 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 5.6 – 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.

5.2.5.3 Примеры программирования транзакций

Демонстрационный пример (листинг 5.7) 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 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 5.7 – Пример отката именованной транзакции

Следующий пример (листинг 5.8) 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.

```
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 5.8 – Пример фиксации и отката вложенных транзакций с контролем счетчика открытых транзакций

Листинг 5.9 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 5.9 – Пример использования транзакции в хранимой процедуре

Процедура NewPersonalDisconts обновляет значение персональной скидки клиентам торговой компании (рисунок 4.2) пропорционально суммарной стоимости заказанных ими товаров в течение месяца.

Процедура содержит одну транзакцию, для которой уставлен 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* базы данных 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 уровнях 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?

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Бейли Л. Изучаем SQL. – Санкт-Петербург, 2012. – 592 с.

2 Волк В. К. Базы данных. Часть 2. Администрирование : учебное пособие. – Курган : Изд-во Курганского гос. ун-та, 2018. – 127 с.

3 Вьейра Р. Программирование баз данных Microsoft SQL Server 2008. Базовый курс. – Санкт-Петербург : Изд-во Диалектика-Вильямс, 2010. – 816 с.

4 Дейт К., Дарвен Х. Основы будущих систем баз данных. Третий манифест / под ред. С. Д. Кузнецова. – 2-е изд. – Москва : Янус-К, 2004.

5 Документация PostgreSQL и Postgres Pro. URL: https:// postgrespro. ru/docs/

6 Ицик Бен-Ган. Microsoft SQL Server 2008. Основы T-SQL. – Санкт-Петербург : БХВ-Петербург, 2009. – 430 с.

7 Кузнецов С. Базы данных. Вводный курс. URL: http://citforum.ru/ database/advanced_intro/

8 Мейер М. Теория реляционных баз данных. – Москва : Мир, 1987. – 608 с.

9 Риккарди Г. Системы баз данных. Теория и практика использования в Internet и среде Java. – Москва : Издательский дом «Вильямс», 2001. – 480 с.

10 Справочник по Transact-SQL. URL: http://msdn.microsoft.com/ru-ru/library/bb510741.aspx.

11 Тарасов С. В. СУБД для программиста. Базы данных изнутри. – Москва : СОЛОН-Пресс. – 320 с.: ил.

12 Туманов В. Основы проектирования реляционных баз данных. URL: https://www.intuit.ru/studies/courses/1095/191

13 Цикритизис Д., Лоховски Ф. Модели данных. – Москва : Финансы и статистика, 1985. – 344 с.

14 Чен П. Модель «сущность-связь» – шаг к единому представлению данных / пер. с англ. С. Кузнецова. URL: http://citforum.ru/ database/ classics/chen/

15 Четвериков В.Н. и др. Базы и банки данных : учеб. для вузов. – Москва : Высш. шк., 1987. – 248 с.

16 Bachman C. W. Data Structure Diagrams. New York: ACM SIGMIS Database: the DATABASE for Advances in Information Systems. 1969. – vol. 1, issue 2. p. 4-10.

17 Brown A.P.G. Modelling a Real-orld System and Designing a Schema to Represent It. Douque and Nijssen (eds.), Data Base Description, 1975.

18 Burleson D.K. The CODASYL Network Model. URL: http:// www. remote-dba. net/t_object_codasyl_network.htm

19 CA IDMS (Integrated Database Management System). URL: https://ru.bmstu. wiki/CA_IDMS.

20 Cachè. High performance multi-model database. URL: 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, No. 4, 1970.

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., Date C.J. The Second Life of Relational Model. // DB/M Magazine, No. 1, 1995; DB/M Magazine, No. 2, 1995.

26 Darwen H. , Date C. J. The Third Manifesto. //ACM SIGMOD Record 24, No. 1, 1995.

27 Date C.J. Notes Toward a Reconstitued Definition of the Relational Model Version 1 (RM/V1). In: Date C. J.(with Darwen H.) Relational Databases: Selected Writings 1989-1991. Reading, Mass.: Addison-Wesley, 1992.

28 Date C.J. The Relational Model. In: Date C. J. An Introduction to Database Systems (7th Ed.), Reading, Mass.: Addison-Wesley, 2000.

29 Date C.J., Darwen H., Lorentzos N. A. Temporal Data and The Relational Model. Reading, Mass.: Addison-Wesley, 2003.

30 A History and Evaluation of System R. IBM Research Laboratory San Jose, California, 1981.

31 URL: https://people.eecs.berkeley.edu/~brewer/cs262/SystemR.pdf

32 Kempe S.A. Short History of the ER Diagram and Information Modeling. URL: http://www.dataversity.net.

33 Long R., Harrington M., Hain R., Nicholls G. IMS Primer. – IBM, 2000. – 300 c. – (IBM Redbook). URL: http://www.redbooks.ibm.com/ redbooks/pdfs/sg245352.pdf

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 учебного пособия.

```
[ WITH <com
mon_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

Учебное издание

Волк Владимир Константинович

БАЗЫ ДАННЫХ

Часть 1

ПРОЕКТИРОВАНИЕ И ПРОГРАММИРОВАНИЕ

Учебное пособие

В авторской редакции

Подписано в печать 24.12.18	Формат 60×84 1/16	Бумага 80 г/м^2
Печать цифровая	Усл. печ. л. 11,12	Уч.-изд. л.11,12
Заказ № 26	Тираж 100	

Библиотечно-издательский центр КГУ
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.