В. В. Кириллов,
Г. Ю. Громов

Введение в реляционные базы данных

- Базы данных и управление ими
- Реляционная модель данных
- SQL – стандартный язык для работы с реляционными базами данных
- Основы проектирования реляционных баз данных
- Создание приложений на языке SQL и его процедурных расширениях
- Хранимые процедуры
- Примеры создания баз данных
- Инструментарий для создания баз данных и приложений к ним

+CD

bhv

Владимир Кириллов

Геннадий Громов

Введение в реляционные базы данных

Санкт-Петербург

«БХВ-Петербург»

2009

УДК 681.3.06
ББК 32.973.26-018.2
К43

Кириллов, В. В.

К43 Введение в реляционные базы данных / В. В. Кириллов, Г. Ю. Громов. — СПб.: БХВ-Петербург, 2009. — 464 с.: ил. + CD-ROM — (Учебная литература для вузов)

ISBN 978-5-94157-770-5

В книге рассматриваются основные понятия баз данных и систем управления ими (СУБД), моделей данных, положенных в основу баз данных и методов проектирования реляционных баз данных. Обсуждаются реляционные операции и основы теории нормализации отношений и приводятся примеры проектирования баз данных. Большое место уделено подробному описанию языка SQL — международного стандарта языка реляционных баз данных. Рассматриваются основные понятия, необходимые для изучения SQL и применения его на практике. Подробно рассмотрено манипулирование данными в интерактивном режиме, затронуты вопросы обеспечения безопасности хранимых данных, средств оптимизации запросов и создания прикладных программ. На прилагаемом к книге компакт-диске содержатся дистрибутивы СУБД OracleXE, SqlDeveloper, учебные базы данных и дополнительные материалы.

Для студентов и начинающих программистов

УДК 681.3.06
ББК 32.973.26-018.2

Рецензент — А. А. Бобцов, профессор, д. т. н., декан факультета компьютерных технологий и управления СПбГУ ИТМО

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Фото	*Кирилла Сергеева*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 01.09.08.
Формат $70\times100^{1}/_{16}$. Печать офсетная. Усл. печ. л. 37,41.
Тираж 2000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-94157-770-5

Оглавление

Введение

Основные идеи современной информационной технологии базируются на концепции баз данных. Согласно этой концепции, основой информационной технологии являются данные, которые должны быть организованы в базах данных с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей.

Увеличение объема и структурной сложности хранимых данных, расширение круга пользователей информационных систем выдвинуло требование создания удобных общесистемных средств интеграции хранимых данных и управления ими. Это и привело к появлению в конце 1960-х годов первых промышленных систем управления базами данных (СУБД) — специализированных программных средств, предназначенных для организации и ведения баз данных. Сначала это были системы с инвертированными списками, иерархические и сетевые системы. В 1969 году была предложена реляционная модель данных *(см. главу 3)*, а в конце 1970-х и начале 1980-х годов стали появляться первые промышленные реляционные СУБД. Сейчас преобладающее большинство СУБД являются реляционными, несмотря на появление объектно-ориентированных СУБД. Это не в последнюю очередь связано с тем, что в конце 1990-х годов большинство ведущих производителей реляционных СУБД создали объектные надстройки к реляционной схеме, что привело к появлению объектно-реляционных СУБД, поддерживающих некоторые технологии, реализующие объектно-ориентированный подход.

Поэтому в данной книге рассматривается реляционная модель данных, реляционные СУБД и основной язык общения с этими СУБД — SQL.

Несмотря на совпадающую модель данных, положенную в основу таких СУБД, и использование ими языка, в основном поддерживающего стандарт SQL:2003 *(см. разд. 4.1)*, все они, в той или иной мере, отличаются друг от друга. Поэтому в иллюстрационных примерах часто будет использоваться реализация SQL:2003 Oracle Database 10g.

Книга адресована широкому кругу пользователей. Для освоения материала книги необходимы минимальные знания о компьютерах. Те же читатели, которые установят на своем персональном компьютере СУБД, базы данных и средства для работы с ними, размещенные на приложенном к книге компакт-диске, смогут достаточно детально изучить материал, связанный с реляционными базами данных и познакомиться с одной из лучших профессиональных СУБД.

Книга ориентирована на студентов высших учебных заведений, изучающих дисциплины "Базы данных", "Информационные системы", "Проектирование информационных систем", а также будет полезна специалистам в области информационных технологий. Это подтверждается отзывами о частично включенных в материал книги учебных пособиях [4, 5], получаемыми авторами уже более десяти лет, а также отзывами студентов направлений "Информатика и вычислительная техника", "Информационные системы" и "Прикладная математика и информатика", изучающих перечисленные ранее дисциплины в СПбГУ ИТМО.

Книга содержит 23 главы и 2 приложения. Главы сгруппированы в семь частей.

Первая часть знакомит с историей появления и основными понятиями баз данных, с моделями данных и реляционным подходом. В *главе 1* дается общее представление о возникновении концепции баз данных и их архитектуре. В *главе 2* рассматриваются основные понятия информационно-логического (инфологического) моделирования, язык моделирования, необходимость введения ключей и подходы к построению моделей. В *главе 3* разбираются основы реляционного подхода и манипулирования реляционными данными.

Вторая часть знакомит с основами языка SQL и его применением для получения информации из реляционных баз данных. В *главе 4* рассматриваются стандарты языка, его синтаксические конструкции, типы данных и разнообразные функции. *Глава 5* описывает предложение `SELECT`, выборку данных без использования и с использованием фразы для отбора данных, упорядочение и агрегирование данных. В *главе 6* продолжается изучение способов выборки данных, но уже из нескольких таблиц. Рассматриваются средства одновременной работы с множеством таблиц, соединения, вложенные подзапросы и объединение нескольких запросов.

В *третьей части* анализируются средства защиты данных, средства изменения содержимого базы данных, управления транзакциями и обеспечения параллельной работы. *Глава 7* описывает средства языка SQL, предназначенные для защиты данных. Здесь рассматривается создание и использование представлений. В *главе 8* обсуждаются особенности синтаксиса и применения предложений модификации данных. В *главе 9* дается определение тран-

закции и объясняется необходимость ее использования. Рассматриваются проблемы, возникающие в многопользовательском режиме работы, и их решение.

Четвертая часть описывает цели и процедуры проектирования, основы нормализации и пример проектирования конкретной базы данных. В *главе 10* обсуждаются цели проектирования, возможные ошибки в процессе проектирования и этапы проектирования. *Глава 11* посвящена функциональным и многозначным зависимостям, нормальным формам, процедурам нормализации. Здесь даются рекомендации по построению даталогической модели. В *главе 12* приводится подробный пример проектирования базы данных "Библиотека".

В *пятой части* рассматривается язык создания основных объектов базы данных, системный каталог и способы оптимизации запросов. В *главе 13* дается обзор языка определения данных, сведения о создании и изменении таблиц, а также последовательностей. *Глава 14* объясняет, зачем нужен системный каталог, описывает его структуру, приводит примеры использования. В *главе 15* обсуждаются вопросы, связанные с оптимизацией выполнения запросов и способов, позволяющих увеличить их производительность.

В *шестой части* рассматриваются различные варианты и средства создания приложений с использованием языка SQL и его процедурных расширений. *Глава 16* описывает статический и динамический SQL, а также интерфейс программирования приложений. В *главе 17* подробно анализируется одно из таких расширений — PL/SQL. Даются основы PL/SQL: достаточно подробно рассматриваются команды, обработка ошибок, курсоры и динамический SQL в PL/SQL. В *главе 18* приводятся синтаксис и примеры создания хранимых процедур, функций, триггеров и пакетов (модулей). Рассказывается о встроенных пакетах PL/SQL.

Седьмая часть достаточно подробно описывает инфологические модели двух связанных предметных областей из интегрированной информационной системы управления университетом. Вырезка "UCHEB" из базы данных этой информационной системы, относящаяся к рассматриваемым моделям, размещена на приложенном к книге компакт-диске. Она может использоваться в процессе изучения материала книги. В *главе 19* рассказывается о Государственных образовательных стандартах и основных образовательных программах. На основании этого материала создается инфологическая модель учебных планов, процедура построения которой рассматривается в следующей главе. *Глава 20* подробно анализирует подходы к построению инфологической модели, здесь приводится диаграмма сущность-связь "Учебные планы". В *главе 21* дается описание предметной области, связанной с успеваемостью

студентов, рассматривается процедура построения инфологической модели и приводится диаграмма сущность-связь "Итоговая успеваемость". *Глава 22* наполнена запросами, которые надо "перевести" на язык SQL и реализовать во время изучения книги. Здесь также даны возможные ответы к некоторым из этих запросов. В *главе 23* приводятся листинги и описания ряда приложений (функции, процедур и пакетов) базы данных "UCHEB".

В приложениях даны полезные справочные сведения о PL/SQL. *Приложение А* представляет собой краткое описание СУБД Oracle 10g Express Edition (XE). Здесь представлены инструкция по ее установке, краткое описание некоторых инструментов для работы с базами данных, расположенными на Oracle 10g XE, и инструкция по установке Oracle SQL Developer. В *приложении Б* приводится перечень инструментальных средств, инструкций и листингов, размещенных на компакт-диске к книге.

ЧАСТЬ I

ЧТО ТАКОЕ БАЗА ДАННЫХ И СУБД

Глава 1

Зачем нужны базы данных

1.1. Данные и ЭВМ

Восприятие реального мира можно соотнести с последовательностью разных, хотя иногда и взаимосвязанных, явлений. С давних времен люди пытались описать эти явления (даже тогда, когда не могли их понять). Такое описание называют *данными*.

Традиционно фиксация данных осуществляется с помощью конкретного средства общения (например, с помощью естественного языка или изображений) на конкретном носителе (например, камне или бумаге). Обычно данные (факты, явления, события, идеи или предметы) и их интерпретация (семантика) фиксируются совместно, так как естественный язык достаточно гибок для представления того и другого. Примером может служить утверждение "Стоимость авиабилета 115". Здесь "115" — данное, а "Стоимость авиабилета" — его семантика.

Нередко данные и интерпретация разделены. Например, "Расписание движения самолетов" может быть представлено в виде таблицы (рис. 1.1), в верхней части которой (отдельно от данных) приводится их интерпретация. Такое разделение затрудняет работу с данными (попробуйте быстро получить сведения из нижней части таблицы, если в ней будет намного больше строк).

Применение ЭВМ для ведения* и обработки данных обычно приводит к еще большему разделению данных и интерпретации. ЭВМ имеет дело главным образом с данными как таковыми. Большая часть интерпретирующей информации вообще не фиксируется в явной форме (ЭВМ не "знает", является ли "21.50" стоимостью авиабилета или временем вылета). Почему же это произошло?

---

* *Ведение* (сопровождение, поддержка) *данных* — термин, объединяющий действия по добавлению, удалению или изменению хранимых данных.

Интерпретация

Номер рейса	Дни недели	Пункт отправления	Время вылета	Пункт назначения	Время прибытия	Тип самолета	Стоимость билета

Данные

138	.2.4..7	Баку	21.12	Москва	0.52	ИЛ-86	115.00
57	..3..6.	Ереван	7.20	Киев	9.25	ТУ-154	92.00
1234	.2…6.	Казань	22.40	Баку	23.50	ТУ-134	73.50
242	1234567	Киев	14.10	Москва	16.15	Б-737	57.00
86	.23.5..	Минск	10.50	Сочи	13.06	ИЛ-86	78.50
137	1.3..6.	Москва	15.17	Баку	18.44	ИЛ-86	115.00
241	1234567	Москва	9.05	Киев	11.05	Б-737	57.00
577	1.3.5..	Рига	21.53	Таллинн	22.57	ЯК 42	21.50
78	..3..6.	Сочи	18.25	Баку	20.12	ТУ-134	44.00
578	.2.4.6.	Таллинн	6.30	Рига	7.37	ЯК 42	21.50

Рис. 1.1. К разделению данных и их интерпретации

Существует, по крайней мере, две исторические причины, по которым применение ЭВМ привело к отделению данных от интерпретации. Во-первых, ЭВМ не обладали достаточными возможностями для обработки текстов на естественном языке — основном языке интерпретации данных. Во-вторых, стоимость памяти ЭВМ была первоначально весьма велика. Память использовалась для хранения самих данных, а интерпретация традиционно возлагалась на пользователя. Пользователь закладывал интерпретацию данных в свою программу, которая "знала", например, что шестое вводимое значение связано со временем прибытия самолета, а четвертое — со временем его вылета. Это существенно повышало роль программы, так как вне интерпретации данные представляют собой не более чем совокупность битов на запоминающем устройстве.

Жесткая зависимость между данными и использующими их программами создает серьезные проблемы в ведении данных и делает их использование менее гибким.

Нередки случаи, когда пользователи одной и той же ЭВМ создают и используют в своих программах разные наборы данных, содержащие сходную информацию. Иногда это связано с тем, что пользователь не знает (либо не захотел узнать), что в соседней комнате или за соседним столом сидит сотрудник, который уже давно ввел в ЭВМ нужные данные. Чаще же потому,

что при совместном использовании одних и тех же данных возникает масса проблем.

Разработчики прикладных программ (написанных, например, на Бейсике, Паскале или Си) размещают нужные им данные в файлах, организуя их наиболее удобным для себя образом. При этом одни и те же данные могут иметь в разных приложениях совершенно разную организацию (разную последовательность размещения в записи, разные форматы одних и тех же полей и т. п.). Обобществить такие данные чрезвычайно трудно: например, любое изменение структуры записи файла, производимое одним из разработчиков, приводит к необходимости изменения другими разработчиками тех программ, которые используют записи этого файла.

Для иллюстрации обратимся к примеру, приведенному в книге У. Девиса "Операционные системы" (М., Мир, 1980):

> "Несколько лет назад почтовое ведомство (из лучших побуждений) пришло к решению, что все адреса должны обязательно включать почтовый индекс. Во многих вычислительных центрах это, казалось бы, незначительное изменение привело к ужасным последствиям. Добавление к адресу нового поля, содержащего шесть символов, означало необходимость внесения изменений в каждую программу, использующую данные этой задачи в соответствии с изменившейся суммарной длиной полей. Тот факт, что какой-то программе для выполнения ее функций не требуется знания почтового индекса, во внимание не принимался: если в некоторой программе содержалось обращение к новой, более длинной записи, то в такую программу вносились изменения, обеспечивающие дополнительное место в памяти.
>
> В условиях автоматизированного управления централизованной базой данных все такие изменения связаны с функциями управляющей программы базы данных. Программы, не использующие значения почтового индекса, не нуждаются в модификации — в них, как и прежде, в соответствии с запросами посылаются те же элементы данных. В таких случаях внесенное изменение неощутимо. Модифицировать необходимо только те программы, которые пользуются новым элементом данных".

1.2. Концепция баз данных

Активная деятельность по отысканию приемлемых способов обобществления непрерывно растущего объема информации привела к созданию в начале 60-х годов XX в. специальных программных комплексов, называемых *системами управления базами данных (СУБД)*.

Основная особенность СУБД — это наличие процедур для ввода и хранения не только самих данных, но и описаний их структуры. Файлы, снабженные описанием хранимых в них данных и находящиеся под управлением СУБД, стали называть банки данных, а затем *базы данных (БД)*.

Пусть, например, требуется сохранить расписание движения самолетов (см. рис. 1.1) и ряд других данных, связанных с организацией работы аэропорта (БД "Аэропорт"). Используя для этого одну из "русифицированных" реляционных СУБД, можно подготовить следующее описание расписания:

```
СОЗДАТЬ ТАБЛИЦУ Расписание
  (Номер_рейса         Целое
   Дни_недели          Текст (8)
   Пункт_отправления   Текст (24)
   Время_вылета        Время
   Пункт_назначения    Текст (24)
   Время_прибытия      Время
   Тип_самолета        Текст (8)
   Стоимость_билета    Валюта);
```

и ввести его вместе с данными в БД "Аэропорт".

Язык запросов СУБД позволяет обращаться за данными как из программ, так и с терминалов (рис. 1.2).

Сформировав запрос:

```
ВЫБРАТЬ  Номер_рейса, Дни_недели, Время_вылета
ИЗ ТАБЛИЦЫ  Расписание
ГДЕ  Пункт_отправления = 'Москва'
  И  Пункт_назначения  = 'Киев'
  И  Время_вылета > 17;
```

получим расписание "Москва—Киев" на вечернее время, а по запросу:

```
ВЫБРАТЬ  КОЛИЧЕСТВО(Номер_рейса)
ИЗ ТАБЛИЦЫ  Расписание
ГДЕ  Пункт_отправления = 'Москва'
  И  Пункт_назначения  = 'Минск';
```

получим количество рейсов "Москва—Минск".

Эти запросы не потеряют актуальности и при расширении таблицы:

```
ДОБАВИТЬ В ТАБЛИЦУ Расписание
   Длительность_полета  Целое;
```

как это было с программами обработки почтовых адресов при введении почтового индекса.

Рис. 1.2. Связь программ и данных при использовании СУБД

Однако за все надо расплачиваться: на обмен данными через СУБД требуется большее время, чем на обмен аналогичными данными прямо из файлов, специально созданных для того или иного приложения.

1.3. Архитектура СУБД

СУБД должна предоставлять доступ к данным любым пользователям, включая и тех, кто практически не имеет и (или) не хочет иметь представления о:

- физическом размещении в памяти компьютера данных и их описаний;
- механизмах поиска запрашиваемых данных;
- проблемах, возникающих при одновременном запросе одних и тех же данных многими пользователями (прикладными программами);
- способах обеспечения защиты данных от некорректных обновлений и (или) несанкционированного доступа;
- поддержании баз данных в актуальном состоянии
- и множестве других функций современных СУБД.

При выполнении основных из этих функций СУБД должна использовать различные описания данных. А что это за описания?

В 1975 году подкомитет SPARC (Standards Planning and Requirements Committee) американского национального института стандартов (American National Standards Institute, ANSI) выдвинул проект трехуровневой архитектуры СУБД (рис. 1.3):

1. *Внешний* (пользовательский).
2. Промежуточный (*концептуальный*).
3. *Внутренний* (физический).

В основе архитектуры ANSI/SPARC лежит концептуальный уровень. Этот уровень описывает данные и их взаимосвязи с наиболее общей точки зрения — концепции архитекторов (разработчиков) базы данных.

Внутренний уровень позволяет скрыть подробности физического хранения данных (носители, файлы ...) от концептуального уровня. Отделение внутреннего уровня от концептуального уровня обеспечивает, так называемую, *физическую независимость* данных.

На внешнем уровне описываются различные подмножества элементов концептуального уровня для представления данных различными пользователями и (или) их программами. Каждый пользователь получает в свое распоряжение часть представлений о данных, но полная концепция скрыта. Отделение внешнего уровня от концептуального уровня обеспечивает *логическую независимость* данных.

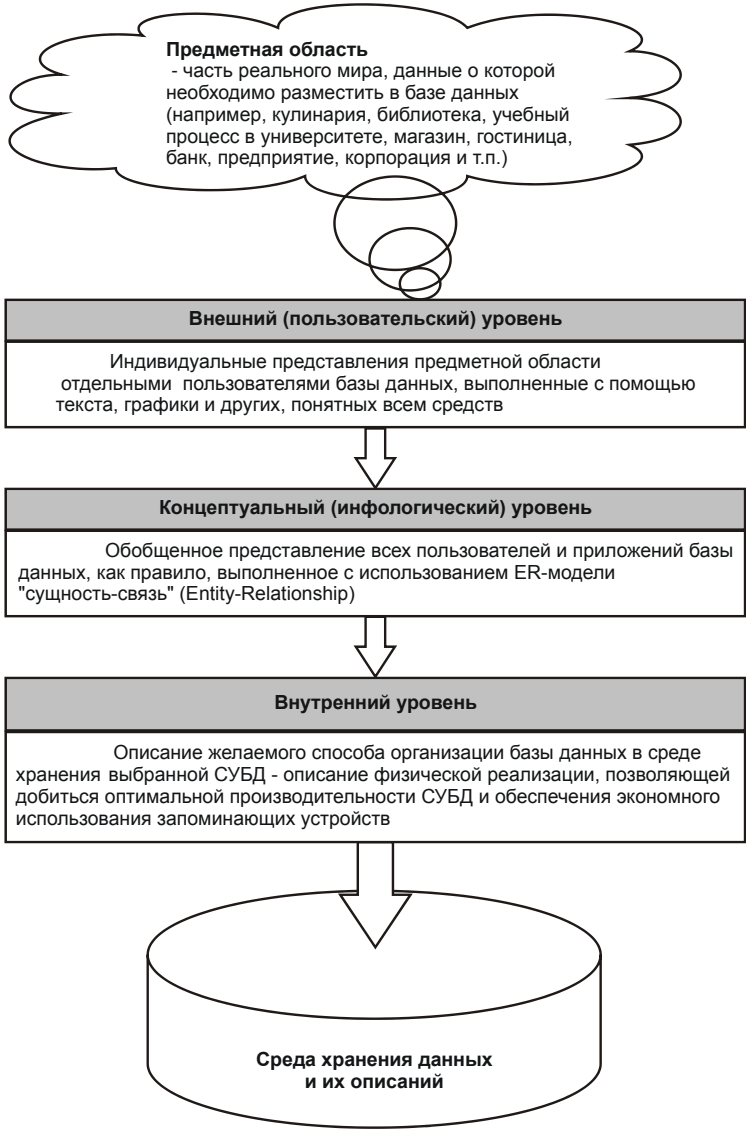


Рис. 1.3. Уровни моделей данных

В дальнейшем эта архитектура была включена в стандарты международной организации по стандартизации (International Organization for Standardization, ISO), членом которой с момента ее основания является Россия.

Естественно, что проект базы данных надо начинать с выбора *предметной области* (той части реального мира, данные о которой надо отразить в базе

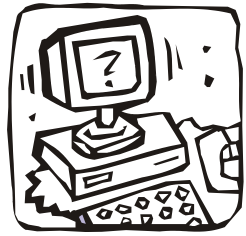
данных) и выявления требований к ней отдельных пользователей (сотрудников организации, для которых создается база данных). Подробнее этот процесс будет рассмотрен далее, а здесь отметим, что проектирование обычно поручается человеку (группе лиц) — *администратору данных (АД)*. Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей и (или) анализа их технических заданий, и свои представления о данных, которые могут потребоваться в будущих приложениях, АД сначала создает обобщенное неформальное описание создаваемой базы данных. Это описание, выполненное с использованием текста (на естественном языке), образцов входных и выходных документов, математических формул, таблиц, графики и других средств, понятных потенциальным пользователям и всем людям, работающим над проектированием базы данных, и есть концептуальная модель данных, которую часто называют логической или *информационно-логической (инфологической)* моделью.

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных и от той СУБД, которая будет использоваться для построения и ведения базы данных. В конце концов, этой средой может быть память человека, а не ЭВМ. Поэтому концептуальная модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторых определений, чтобы эта модель продолжала отражать предметную область.

Если АД — это человек, отвечающий за стратегию и политику принятия решений, связанных с данными базы данных, то человек, обеспечивающий необходимую техническую поддержку для реализации принятых решений, называется *администратором базы данных* (АБД).

АБД выбирает конкретную СУБД и решает, как данные будут представлены в хранимой базе данных, т. е. осуществляет физическое проектирование базы данных — преобразование концептуальной модели в физическую модель.

Трехуровневая архитектура позволяет обеспечить *независимость хранимых данных* от использующих их программ. АБД может при необходимости переписать хранимые данные на другие носители информации и (или) реорганизовать их физическую структуру, изменив лишь физическую модель данных. АД может подключить к системе любое число новых пользователей (новых приложений), дополнив, если надо, концептуальную модель. Указанные изменения физической и концептуальной моделей не будут замечены существующими пользователями системы (окажутся "прозрачными" для них), так же как не будут замечены и новые пользователи. Следовательно, независимость данных обеспечивает возможность развития системы баз данных без разрушения существующих приложений.



Глава 2

Инфологическая модель данных "сущность-связь"

2.1. Основные понятия

Цель инфологического моделирования — обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Поэтому инфологическую модель данных пытаются строить по аналогии с естественным языком (последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественного языка). Основными конструктивными элементами инфологических моделей являются сущности, связи между ними и их свойства (атрибуты).

Сущность — любой различимый объект, факт, явление, событие, идея или предмет, информацию о котором необходимо хранить в базе данных. Сущностями могут быть люди, места, самолеты, рейсы, вкус, цвет, женитьба, гроза, изобретение, боль и т. п. Необходимо различать такие понятия, как *тип сущности* и *экземпляр сущности*. Понятие тип сущности относится к набору однородных личностей, предметов, событий или идей, выступающих как целое. Экземпляр сущности относится к конкретной вещи в наборе. Например, типом сущности может быть `ГОРОД`, а экземпляром — `Москва`, `Киев` и т. д.

Атрибут — поименованная характеристика (свойство) сущности. Это любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Наименование атрибута должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей, например, `ЦВЕТ` может быть определен для многих сущностей: `СОБАКА`, `АВТОМОБИЛЬ`, `ДЫМ` и т. д. Атрибуты используются для определения того, какая информация должна быть собрана о сущности. Примерами атрибутов для сущности `АВТОМОБИЛЬ` являются `ТИП`, `МАРКА`, `НОМЕРНОЙ ЗНАК`, `ЦВЕТ` и т. д. Здесь также существует различие

между типом и экземпляром. Тип атрибута `ЦВЕТ` имеет много экземпляров или значений (`Красный`, `Синий`, `Банановый`, `Белая ночь` и т. д.), однако каждому экземпляру сущности присваивается только одно значение атрибута.

Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, в расписании движения самолетов (см. рис. 1.1) город — это атрибут расписания, а в кодификаторе адресов город — тип сущности.

Ключ — минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Для сущности `Расписание` (см. *разд. 1.2*) ключом является атрибут `Номер_рейса` или набор: `Пункт_отправления`, `Время_вылета` и `Пункт_назначения` (при условии, что из пункта в пункт вылетает в каждый момент времени один самолет).

Связь — ассоциирование двух или более сущностей. Абсолютное различие между типами сущностей и связями отсутствует. Один и тот же факт может совершенно обоснованно рассматриваться или как сущность, или как связь. Например, в запросе — "с кем вступила в брак Алла Пугачева" брак — связь, а в запросе — "сколько браков было зарегистрировано в этом ЗАГСе в прошлом году" брак — сущность.

Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных — это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей. Наличие такого множества связей и определяет сложность инфологических моделей.

2.2. Характеристика связей и язык моделирования

При построении инфологических моделей можно использовать язык *ER-диаграмм* — от англ. Entity-Relationship, т. е. сущность-связь. Этот язык был предложен в 1976 году сотрудником корпорации IBM Питером Ченом [7].

В нем предлагалось изображать сущности помеченными прямоугольниками, связи — помеченными ромбами, атрибуты — помеченными овалами. Над линиями, соединяющими прямоугольники, может проставляться степень связи (1 или буква M, заменяющая слово "много") и необходимое пояснение (рис. 2.1).

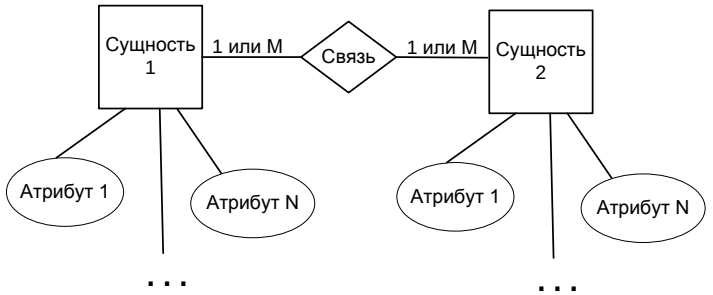


Рис. 2.1. Элементы ER-диаграмм языка, предложенного Ченом

Так, между двумя сущностями, например, А и Б возможны четыре вида связей.

Первый тип — связь *"один-к-одному" (1:1)*: в каждый момент времени каждому представителю (экземпляру) сущности А соответствует 1 или 0 представителей сущности Б. Например, студент может не "заработать" стипендию, получить обычную или одну из повышенных стипендий (рис. 2.2, *а*).

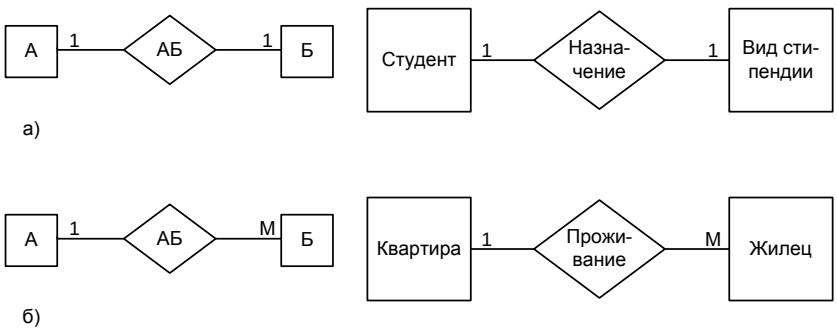


Рис. 2.2. Примеры простейших ER-моделей

Второй тип — связь *"один-ко-многим" (1:M)*: одному представителю сущности А соответствуют 0, 1 или несколько представителей сущности Б. Напри-

мер, квартира может пустовать, в ней может жить один или несколько жильцов (рис. 2.2, *б*).

Так как между двумя сущностями возможны связи в обоих направлениях, то существует еще два типа связи *"многие-к-одному" (M:1)* и *"многие-ко-многим" (M:M)*.

Пример 2.1. Если связь между сущностями МУЖЧИНЫ и ЖЕНЩИНЫ называется БРАК, то существует четыре возможных представления такой связи (рис. 2.3).

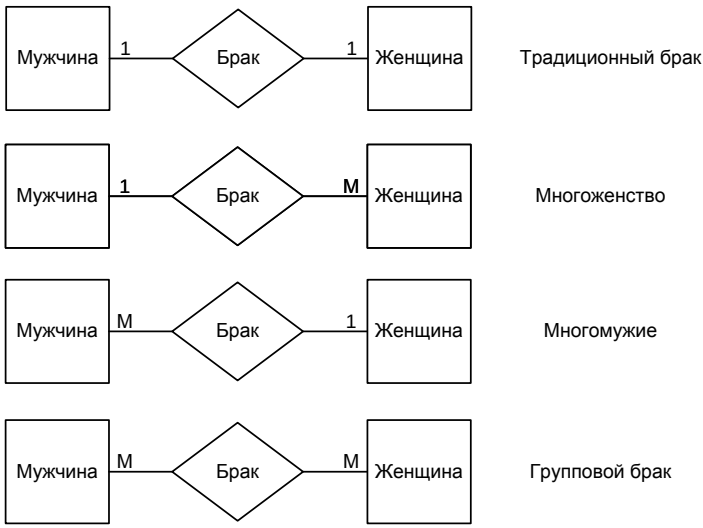


Рис. 2.3. Возможные связи между двумя сущностями

Характер связей между сущностями не ограничивается перечисленными. Существуют и более сложные связи:

- множество связей между одними и теми же сущностями (рис. 2.4, *а*): пациент, имея одного лечащего врача, может иметь также несколько врачей-консультантов; врач может быть лечащим врачом нескольких пациентов и может одновременно консультировать несколько других пациентов;
- тренарные связи (рис. 2.4, *б*): врач может назначить несколько пациентов на несколько анализов, анализ может быть назначен несколькими врачами нескольким пациентам и пациент может быть назначен на несколько анализов несколькими врачами;
- связи более высоких порядков, семантика (смысл) которых иногда очень сложна.

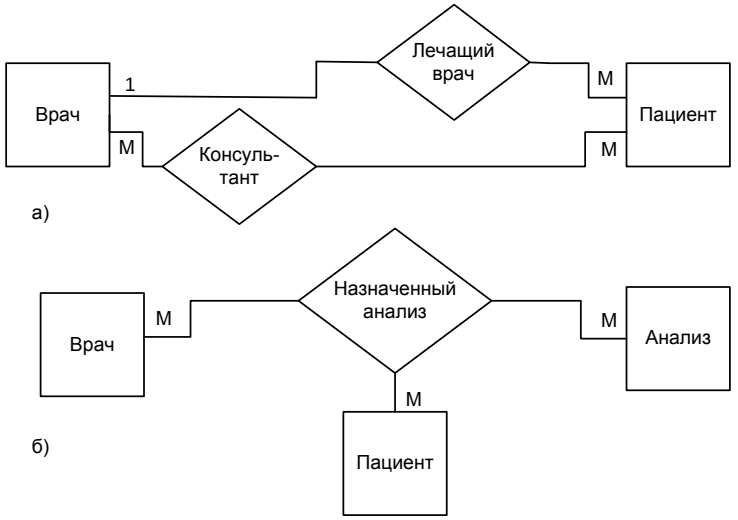


Рис. 2.4. Примеры сложных связей между сущностями

В приведенных примерах для повышения иллюстративности рассматриваемых связей не показаны атрибуты сущностей ER-диаграмм. Так, ввод атрибутов в описание брачных связей:

- фамилии, имена и отчества мужа и жены;
- даты и места рождения мужа и жены;
- даты и места регистрации брака;
- фамилий, присвоенных мужу и жене после регистрации;
- даты выдачи и номера свидетельства о браке

значительно усложнит ER-диаграмму и затруднит ее понимание.

Поэтому к настоящему времени появилось множество более удобных графических нотаций описания ER-диаграмм, поддержанных CASE-средствами (от Computer Aided Software/System Engineering) разработки информационных систем и редакторами деловой графики. В данной книге мы будем пользоваться нотациями системы построения диаграмм Microsoft Visio.

В этих нотациях сущность изображается в виде прямоугольника, в верхней части которого располагается имя сущности (рис. 2.5, *а*). В прямоугольнике перечислены атрибуты сущности. Атрибуты, расположенные сверху и отделенные от остальных горизонтальной линией, являются ключевыми.

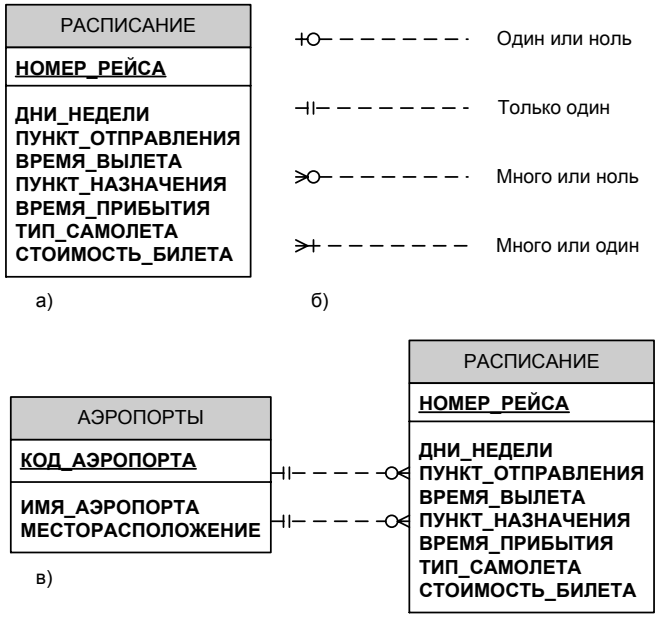


Рис. 2.5. Сущность (*а*), виды связей (*б*) и пример ER-диаграммы (*в*)

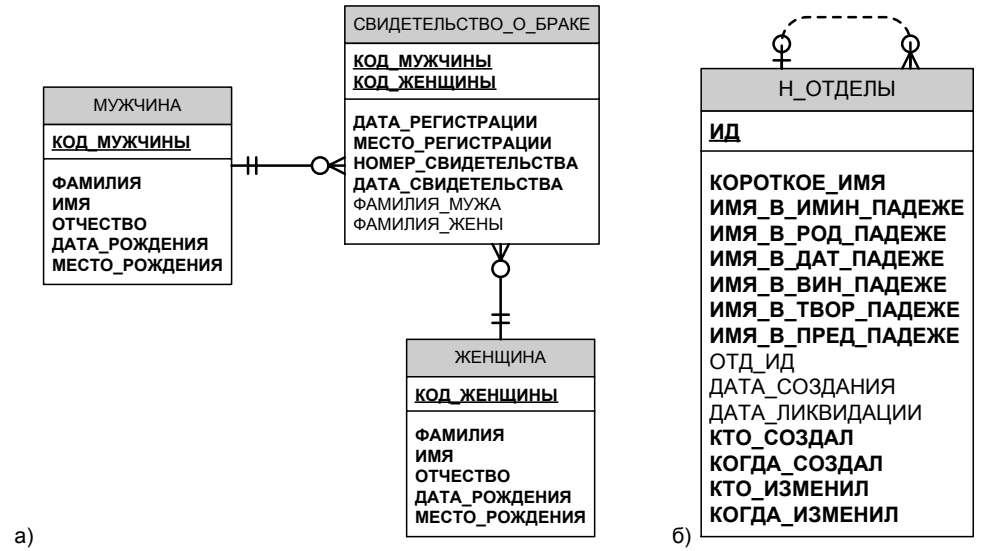


Рис. 2.6. Примеры связей в ER-диаграммах

Связь изображается пунктирной линией между двумя сущностями (рис. 2.5, *б*). На концах линий проставляются условные графические обозначения: вертикальная черта (один), кружок (ноль или "необязательно"), "воронья лапа" (много).

Если в ключ какой-либо сущности входит ключ другой сущности, то связь между такими сущностями изображается не пунктирной, а сплошной линией (рис. 2.6, *а*).

Достаточно часто встречается еще один специфический вид связи: рекурсивная связь между атрибутами одной сущности "Один-ко-многим" ("Свиное ухо"), используемая для описания иерархий с любым числом уровней. На рис. 2.6, *б* приведено изображение такой связи для сущности `ОТДЕЛЫ` (см. *разд. 20.3*). Иллюстрацию такой связи легко проследить по табл. 2.1, где представлены некоторые столбцы и строки таблицы `Н_ОТДЕЛЫ`.

Таблица 2.1. *Идентификаторы, подчиненность и названия ряда отделов СПбГУ ИТМО*

ИД	ОТД_ИД	ИМЯ_В_ИМИН_ПАДЕЖЕ
101	703	кафедра систем управления и информатики
102	703	кафедра вычислительной техники
103	705	кафедра измерительных технологии и компьютерной томографии
105	705	кафедра мехатроники
106	704	кафедра теоретической и прикладной механики
107	705	кафедра инженерной и компьютерной графики
108	703	кафедра проектирования компьютерных систем
109	703	кафедра информационно-навигационных систем
110	706	кафедра иностранных языков
111	703	кафедра информатики и прикладной математики
	…	
701	777	факультет оптико-информационных систем и технологий
702	777	факультет инженерно-физический
703	777	факультет компьютерных технологий и управления

Таблица 2.1 *(окончание)*

ИД	ОТД_ИД	ИМЯ_В_ИМИН_ПАДЕЖЕ
704	777	факультет естественнонаучный
705	777	факультет точной механики и технологий
706	777	факультет гуманитарный
707	777	факультет послевузовского профессионального обучения
	…	
777		Санкт-Петербургский государственный университет информационных технологий, механики и оптики

В строке табл. 2.1 кроме идентификатора (ИД) и имени (ИМЯ_В_ИМИН_ПАДЕЖЕ) отдела хранится идентификатор "начальника" (ОТД_ИД), т. е. того подразделения, в состав которого он входит. Так кафедра вычислительной техники (ИД=102) входит в состав факультета компьютерных технологий и управления (ИД=703), который, в свою очередь, входит в состав университета (ИД=777). В данном случае рассмотрен пример с тремя уровнями иерархии, но ясно, что при использовании этого описания количество этих уровней ограничивается только предметной областью (здесь можно было бы представить, например, любые секции кафедры, подсекции и т. п.).

Об организации иерархических запросов рассказано в *разд. 5.6*.

2.3. Классификация сущностей

Настал момент разобраться в терминологии. Основоположник реляционной модели баз данных Эдгар Кодд [8] определяет три основных класса сущностей: *стержневые*, *ассоциативные* и *характеристические*.

Стержневая сущность (стержень) — это независимая сущность, которая не является ни характеристикой, ни ассоциацией (см. далее).

В рассмотренных ранее примерах стержни — это Студент, Квартира, Мужчины, Врач и другие, названия которых помещены в прямоугольники.

Ассоциативная сущность (ассоциация) — это связь вида "многие-ко-многим" ("*-ко-многим" и т. д.) между двумя или более сущностями или эк-

земплярами сущности. Ассоциации рассматриваются как полноправные сущности:

- они могут участвовать в других ассоциациях точно так же, как стержневые сущности;
- могут обладать свойствами, т. е. иметь не только набор ключевых атрибутов, необходимых для указания связей, но и любое число других атрибутов, характеризующих связь. Например, ассоциация СВИДЕТЕЛЬСТВО_О_БРАКЕ (см. рис. 2.6) содержит ключевые атрибуты КОД_МУЖЧИНЫ и КОД_ЖЕНЩИНЫ, а также уточняющие атрибуты ДАТА_РЕГИСТРАЦИИ, МЕСТО_РЕГИСТРАЦИИ, НОМЕР_СВИДЕТЕЛЬСТВА и т. д.

Характеристическая сущность (характеристика) — это связь вида "многие-к-одной" или "одна-к-одной" между двумя сущностями (частный случай ассоциации). Единственная цель характеристики в рамках рассматриваемой предметной области состоит в описании или уточнении некоторой другой сущности. Необходимость в них возникает в связи с тем, что сущности реального мира имеют иногда многозначные свойства. Муж может иметь несколько жен; книга — несколько характеристик переиздания (исправленное, дополненное, переработанное и пр.) и т. д. Существование характеристики полностью зависит от характеризуемой сущности: женщины лишаются статуса жен, если умирает их муж.

Пример 2.2. В заключение рассмотрим пример построения инфологической модели базы данных "COOK", предназначенной для использования в пансионатах, санаториях и других организациях, предоставляющих услуги по обеспечению отдыха.

Информация из такой базы данных будет использоваться шеф-поваром пансионата для составления меню, учитывающего примерную стоимость и необходимую калорийности суточного рациона отдыхающих. В меню предлагается по несколько альтернативных блюд каждого вида (закуска, горячее, суп и т. п.) для каждой трапезы (завтрак, обед, ужин).

Перед завтраком каждый отдыхающий выбирает в информационной системе номер закрепленного за ним места в столовой пансионата и желаемый набор блюд для каждой из трапез следующего дня (желаемый набор строк меню). Это позволяет определить, сколько порций того или иного блюда надо приготовить для каждой трапезы, а также набор и количество необходимых продуктов.

Завхоз, связанный с поставщиками продуктов, определяет, что необходимо заказать для обеспечения работы столовой.

При составлении меню шеф-повар использует кулинарную книгу, где размещена информация о рецептах (рис. 2.7) и таблицы с химическим составом и пищевой ценностью различных продуктов.

127 Лобио по грузински (закуска)

Ломаную очищенную фасоль, нашинкованный лук посолить, посыпать перцем и припустить в масле с небольшим количеством бульона; добавить кинзу, зелень петрушки, реган (базилик) и довести до готовности. Затем запечь в духовке.

Фасоль стручковая (свежая или консервированная) 200,

Лук зеленый 40, Масло сливочное 30, Зелень 10.

Выход 210. Калорий 725.

Рис. 2.7. Пример кулинарного рецепта

Исходя из потребностей указанных ранее лиц, можно определить объекты, необходимые для выявления сущностей и атрибутов проектируемой базы:

1. Блюда, для описания которых нужны данные, входящие в их кулинарные рецепты:
   - код (номер) блюда (например, из книги кулинарных рецептов);
   - название блюда;
   - вид блюда (закуска, суп, горячее и т. п.);
   - рецепт (технология приготовления блюда);
   - выход (вес порции);
   - калорийность блюда;
   - название, вес и основные вещества (белки, жиры, углеводы, витамины и др.) каждого продукта, входящего в блюдо;
   - приведенная стоимость приготовления одной порции блюда (трудоемкость).
2. Для каждого поставщика продуктов:
   - код (номер) поставщика продукта;
   - название поставщика и его статус (рынок, ферма, универсам и т. п.);
   - данные о поставщике (город, адрес, телефон);
   - название поставляемого продукта;
   - дата поставки и цена на момент поставки.

3. Ежедневное потребление блюд (расход): блюдо, количество порций, дата.
4. Меню на следующий день, где на каждую из трапез (завтрак, обед, ужин) для каждого из видов блюд приводится несколько различных блюд.
5. Выбор каждым из отдыхающих конкретных блюд из меню (для упрощения схемы опущены сведения об отдыхающих и, следовательно, привязка их к местам в столовой пансионата).

Анализ объектов позволяет выделить:

- Стержни: ВИДЫ_БЛЮД, ТРАПЕЗЫ, ПРОДУКТЫ и ПОСТАВЩИКИ;

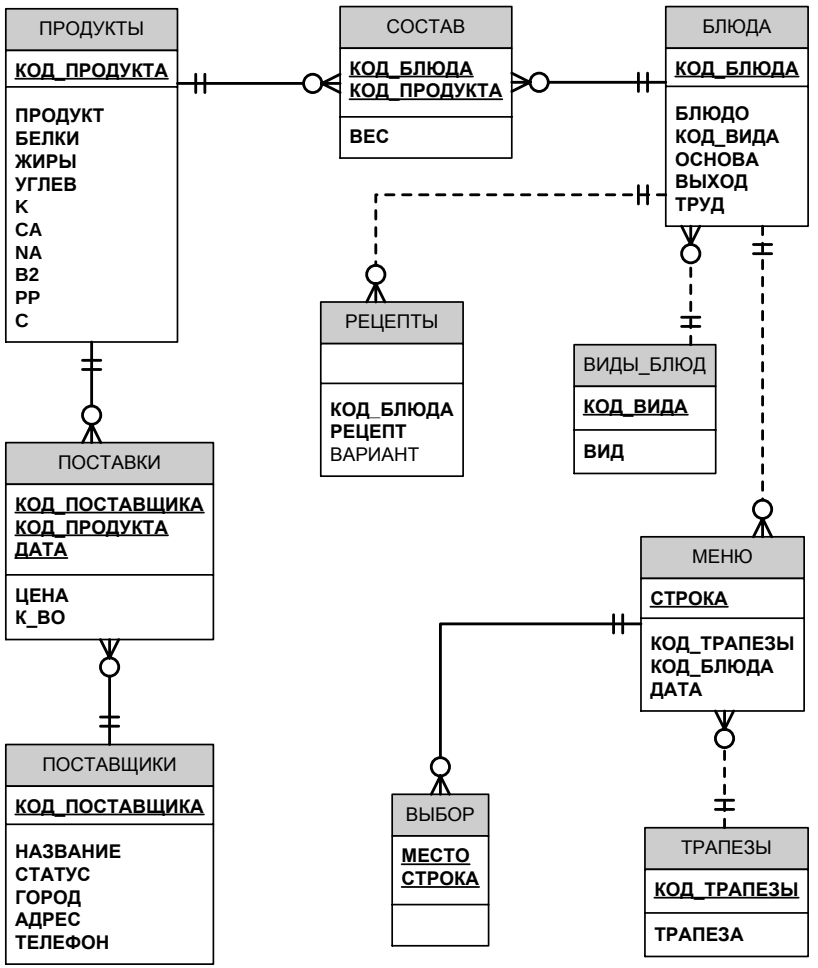


Рис. 2.8. Инфологическая модель базы данных "COOK"

- Ассоциации: СОСТАВ (связывает БЛЮДА с ПРОДУКТАМИ), ПОСТАВКИ (связывает ПОСТАВЩИКОВ с ПРОДУКТАМИ), МЕНЮ (связывает ТРАПЕЗЫ с БЛЮДАМИ) и ВЫБОР (связывает МЕНЮ с МЕСТОМ в столовой) и, наконец, частный случай ассоциации — БЛЮДА, зависящие от единственной стержневой сущности ВИДЫ_БЛЮД;
- Характеристика: РЕЦЕПТЫ (характеризует БЛЮДА).

ER-диаграмма модели показана на рис. 2.8.

2.4. О первичных и внешних ключах

Напомним, что *ключ* или *возможный ключ* — это минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Каждая сущность должна обладать хотя бы одним возможным ключом. Если же возникает ситуация, когда из состава атрибутов сущности не удается создать возможного ключа (*естественного ключа*), то создают, так называемый, *суррогатный ключ* — автоматически сгенерированное значение, никак не связанное с информационным содержанием сущности.

Один из возможных естественных ключей или суррогатный ключ принимается за *первичный ключ*. При выборе первичного ключа следует отдавать предпочтение несоставным ключам или ключам, составленным из минимального числа атрибутов. Нецелесообразно также использовать ключи с длинными текстовыми значениями (предпочтительнее использовать целочисленные атрибуты). Так, для идентификации студента можно использовать либо уникальный номер зачетной книжки, либо набор из фамилии, имени, отчества, номера группы и может быть дополнительных атрибутов, так как не исключено появление в группе двух студентов (а чаще студенток) с одинаковыми фамилиями, именами и отчествами. Плохо также использовать в качестве ключа не номер блюда, а его название, например, "Закуска из плавленых сырков "Дружба" с ветчиной и соленым огурцом" или "Заяц в сметане с картофельными крокетами и салатом из красной капусты".

Не допускается, чтобы первичный ключ стержневой сущности (любой атрибут, участвующий в первичном ключе) принимал неопределенное значение. Иначе возникнет противоречивая ситуация: появится не обладающий индивидуальностью и, следовательно, не существующий экземпляр стержневой сущности. По тем же причинам необходимо обеспечить уникальность первичного ключа.

Следует также обсудить ситуацию, когда по тем или иным причинам приходится или целесообразнее использовать суррогатный ключ. Например, на рис. 2.8, в сущности РЕЦЕПТ в качестве возможных первичных ключей можно было бы использовать пары атрибутов: (КОД_БЛЮДА, РЕЦЕПТ) или (КОД_БЛЮДА, ВАРИАНТ), где вариант — номер технологии приготовления блюда, например, кофе черного или салата "Оливье". Однако значение первого из этих ключей: (КОД_БЛЮДА, РЕЦЕПТ) — чересчур громоздко, а второго: (КОД_БЛЮДА, ВАРИАНТ) — требует обязательного ввода номера варианта, даже в том случае, когда существует всего один вариант рецепта. Поэтому введем в состав атрибутов сущности РЕЦЕПТ атрибут ИД (рис. 2.9), который будет автоматически генерироваться СУБД во время ввода рецептов и использоваться в качестве суррогатного первичного ключа.

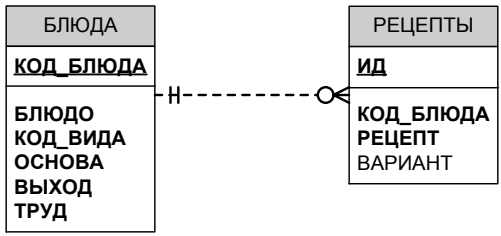


Рис. 2.9. Ввод суррогатного ключа ИД в сущность РЕЦЕПТЫ

Теперь о внешних ключах:

- Если сущность В связывает сущности А и Б, то она должна включать внешние ключи, соответствующие первичным ключам сущностей А и Б. Например, ассоциативная сущность СОСТАВ (см. рис. 2.8) включает внешние ключи КОД_БЛЮДА и КОД_ПРОДУКТА, соответствующие первичным ключам связываемых стержневых сущностей БЛЮДА и ПРОДУКТЫ.
- Если сущность Б характеризует сущность А, то она должна включать внешний ключ, соответствующий первичному ключу сущности А. Например, характеристическая сущность РЕЦЕПТ (см. рис. 2.8 или 2.9) включает внешний ключ КОД_БЛЮДА, соответствующий первичному ключу характеризуемой сущности БЛЮДА.

Таким образом, при рассмотрении выбора способа представления ассоциаций и характеристик в базе данных основной вопрос, на который следует получить ответ: "Каковы внешние ключи?" И далее, для каждого внешнего ключа необходимо решить три вопроса.

Первый вопрос — может ли данный внешний ключ принимать неопределенные значения (NULL-значения)? Иначе говоря, может ли существовать некоторый экземпляр сущности данного типа, для которого неизвестна целевая сущность, указываемая внешним ключом?

Ответ на данный вопрос не зависит от прихоти проектировщика базы данных, а определяется фактическим образом действий, принятым в той части реального мира, которая должна быть представлена в рассматриваемой базе данных. Подобные замечания имеют отношение и к вопросам (пунктам), обсуждаемым далее.

Так, в случае поставок это, вероятно, невозможно — поставка, осуществляемая неизвестным поставщиком, или поставка неизвестного продукта не имеют смысла. Но вполне возможно существование блюда с неизвестным значением его вида (суп, горячее и пр.).

Второй вопрос — что должно случиться при попытке УДАЛЕНИЯ экземпляра сущности, на первичный ключ которой ссылается внешний ключ? Например, при удалении поставщика, который осуществил, по крайней мере, одну поставку. Для данной операции существует три возможности, она:

- КАСКАДИРУЕТСЯ — операция удаления "каскадируется" с тем, чтобы удалить также все поставки удаляемого поставщика;
- ОГРАНИЧИВАЕТСЯ — удаляются лишь те поставщики, которые еще не осуществляли поставок. Иначе операция удаления отвергается;
- УСТАНАВЛИВАЕТСЯ — для всех поставок удаляемого поставщика внешний ключ устанавливается в неопределенное (NULL) значение, а затем этот поставщик удаляется. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

Третий вопрос — что должно происходить при попытке ОБНОВЛЕНИЯ первичного ключа сущности, на которую ссылается некоторый внешний ключ? Например, может быть предпринята попытка обновить номер такого поставщика, для которого имеется, по крайней мере, одна соответствующая поставка. Этот случай для определенности снова рассмотрим подробнее. Имеются те же три возможности, как и при удалении:

- КАСКАДИРУЕТСЯ — операция обновления "каскадируется" с тем, чтобы обновить также и внешний ключ в поставках этого поставщика;
- ОГРАНИЧИВАЕТСЯ — обновляются первичные ключи лишь тех поставщиков, которые еще не осуществляли поставок. Иначе операция обновления отвергается;
- УСТАНАВЛИВАЕТСЯ — для всех поставок обновляемого поставщика внешний ключ устанавливается в неопределенное значение, а затем об-

новляется первичный ключ поставщика. Такая возможность, конечно, неприменима, если данный внешний ключ не должен содержать NULL-значений.

Таким образом, для каждого внешнего ключа в проекте проектировщик базы данных должен специфицировать не только поле или комбинацию полей, составляющих этот внешний ключ и сущность, которая идентифицируется этим ключом, но также и ответы на указанные ранее вопросы (три ограничения, которые относятся к этому внешнему ключу).

Теперь о характеристиках, существование которых зависит от характеризуемых сущностей. Для них три рассмотренные ранее ограничения на внешний ключ должны специфицироваться следующим образом:

- NULL-значения не допустимы,
- УДАЛЕНИЕ ИЗ (характеризуемой сущности) КАСКАДИРУЕТСЯ,
- ОБНОВЛЕНИЕ (первичный ключ характеризуемой сущности) КАСКАДИРУЕТСЯ.

Указанные спецификации представляют зависимость по существованию характеристических сущностей.

2.5. Ограничения целостности

Целостность (от англ. integrity — нетронутость, неприкосновенность, сохранность, целостность) — понимается как правильность данных в любой момент времени. Но эта цель может быть достигнута лишь в определенных пределах: СУБД не может контролировать правильность каждого отдельного значения, вводимого в базу данных (хотя каждое значение можно проверить на правдоподобность). Например, нельзя обнаружить, что вводимое значение 5 (представляющее номер дня недели) в действительности должно быть равно 3. С другой стороны, значение 9 явно будет ошибочным и СУБД должна его отвергнуть. Однако для этого ей следует сообщить, что номера дней недели должны принадлежать набору (1,2,3,4,5,6,7).

Поддержание целостности базы данных может рассматриваться как защита данных от неверных изменений или разрушений (не путать с незаконными изменениями и разрушениями, являющимися проблемой безопасности). Современные СУБД имеют ряд средств для обеспечения поддержания целостности (так же, как и средств обеспечения поддержания безопасности).

Выделяют три группы правил целостности.

- Целостность по сущностям.

  Не допускается, чтобы какой-либо атрибут, участвующий в первичном ключе, принимал неопределенное значение.

- Целостность по ссылкам.

  Значение внешнего ключа должно либо:

  - быть равным значению первичного ключа ассоциируемой (характеризуемой) сущности;
  - быть полностью неопределенным, т. е. каждое значение атрибута, участвующего во внешнем ключе, должно быть неопределенным.

- Целостность, определяемая пользователем.

  Для любой конкретной базы данных существует ряд дополнительных специфических правил, которые относятся к ней одной и определяются разработчиком. Чаще всего контролируется:

  - уникальность тех или иных атрибутов;
  - диапазон значений (экзаменационная оценка от 2 до 5);
  - принадлежность набору значений (пол "М" или "Ж").

2.6. О построении инфологической модели

Читатель, познакомившийся лишь с материалом данной главы, не сможет правильно воспринять и оценить тех советов и рекомендаций по построению хорошей инфологической модели, которые десятилетиями формировались крупнейшими специалистами в области обработки данных. Для этого надо, по крайней мере, изучить последующие материалы. В идеале же необходимо, чтобы читатель предварительно реализовал хотя бы один проект информационной системы, предложил его реальным пользователям и побыл администратором базы данных и приложений столь долго, чтобы осознать хотя бы небольшую толику проблем, возникающих из-за недостаточно продуманного проекта. Опыт авторов и всех знакомых им специалистов по информационным системам показывает, что любые теоретические рекомендации воспринимаются всерьез лишь после нескольких безрезультатных попыток оживления неудачно спроектированных систем. (Хотя есть и такие проектировщики, которые продолжают верить, что смогут реанимировать умирающий проект с помощью изменения программ, а не инфологической модели базы данных.)

Основная сложность восприятия рекомендаций, приведенных в последующих главах, чисто психологического плана.

Действительно, для определения перечня и структуры хранимых данных надо собрать информацию о реальных и потенциальных приложениях, а также о пользователях базы данных, а при построении инфологической модели следует заботиться лишь о надежности хранения этих данных, напрочь забывая о приложениях и пользователях, для которых создается база данных.

Это связано с абсолютно различающимися требованиями к базе данных прикладных программистов и администратора базы данных. Первые хотели бы иметь в одном месте (например, в одной таблице) все данные, необходимые им для реализации запроса из прикладной программы или с терминала. Вторые же заботятся об исключении возможных искажений хранимых данных при вводе в базу данных новой информации и обновлении или удалении существующей. Для этого они удаляют из базы данных дубликаты и нежелательные функциональные связи между атрибутами, разбивая базу данных на множество маленьких таблиц. Так как многолетний мировой опыт использования информационных систем, построенных на основе баз данных, показывает, что недостатки проекта невозможно устранить любыми ухищрениями в программах приложений, то опытные проектировщики не позволяют себе идти навстречу прикладным программистам (даже тогда, когда они сами являются таковыми).

И хотя авторы осознают, что большинство людей предпочитает учиться на собственных ошибках, они все же еще раз советуют неопытным проектировщикам баз данных:

- ❒ четко разграничивать такие понятия как запрос на данные и ведение данных (ввод, изменение и удаление);
- ❒ помнить, что, как правило, база данных является информационной основой не одного, а нескольких приложений, часть из которых появится в будущем;
- ❒ плохой проект базы данных не может быть исправлен с помощью любых (даже самых изощренных) приложений.

Глава 3

Реляционный подход

3.1. Реляционная структура данных

В конце 60-х годов появились работы, в которых обсуждались возможности применения различных табличных даталогических моделей данных, т. е. возможности использования привычных и естественных способов представления данных. Наиболее значительной из них была статья исследователя фирмы IBM д-ра Эдгара Кодда [9], где, вероятно, впервые был применен термин "реляционная модель данных".

Будучи математиком по образованию Эдгар Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как *отношение* — relation (англ.) [1, 4, 9].

Наименьшая единица данных реляционной модели — это отдельное *атомарное* (неразложимое) для данной модели значение данных. Так, в одной предметной области фамилия, имя и отчество могут рассматриваться как единое значение, а в другой — как три различных значения.

Доменом называется множество атомарных значений одного и того же типа. Так, на рис. 1.1 домен пунктов отправления (назначения) — множество названий населенных пунктов, а домен номеров рейса — множество целых положительных чисел.

Смысл доменов состоит в следующем. Если значения двух атрибутов берутся из одного и того же домена, то, вероятно, имеют смысл сравнения, использующие эти два атрибута (например, для организации транзитного рейса можно дать запрос "Выдать рейсы, в которых время вылета из Москвы в Сочи больше времени прибытия из Архангельска в Москву"). Если же значения

двух атрибутов берутся из различных доменов, то их сравнение, вероятно, лишено смысла: стоит ли сравнивать номер рейса со стоимостью билета?

Отношение на доменах D1, D2, ..., Dn (не обязательно, чтобы все они были различны) состоит из заголовка и тела. На рис. 3.1 приведен пример отношения для расписания движения самолетов (см. рис. 1.1).

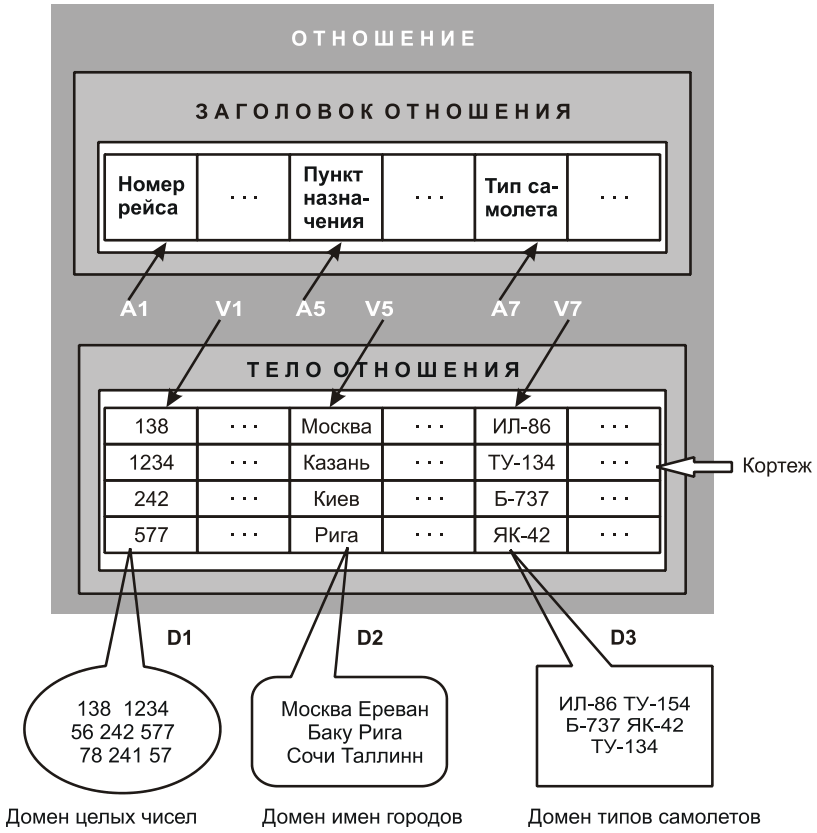


Рис. 3.1. Отношение с математической точки зрения (Ai — атрибуты, Vi — значения атрибутов)

Заголовок (на рис. 1.1 он назывался интерпретацией) состоит из такого фиксированного множества атрибутов A1, A2, ..., An, что существует взаимно однозначное соответствие между этими атрибутами Ai и определяющими их доменами Di (i = 1, 2, ..., n).

Тело состоит из меняющегося во времени множества *кортежей*, где каждый кортеж состоит в свою очередь из множества пар атрибут-значение (Ai:Vi),

(i =1, 2, ..., n), по одной такой паре для каждого атрибута Ai в заголовке. Для любой заданной пары атрибут-значение (Ai:Vi) Vi является значением из единственного домена Di, который связан с атрибутом Ai.

Степень отношения — это число его атрибутов. Отношение степени один называют унарным, степени два — бинарным, степени три — тернарным, а степени n — n-арным. Степень отношения РЕЙС (см. рис. 1.1) — 8.

Кардинальное число или *мощность отношения* — это число его кортежей. Мощность отношения РЕЙС равна 10. Кардинальное число отношения изменяется во времени в отличие от его степени.

Поскольку отношение — это множество, а множества по определению не содержат совпадающих элементов, то никакие два кортежа отношения не могут быть дубликатами друг друга в любой произвольно-заданный момент времени.

Пусть R — отношение с атрибутами A1, A2, ..., An. Говорят, что множество атрибутов K = (Ai, Aj, ..., Ak) отношения R является возможным ключом R тогда и только тогда, когда удовлетворяются два независимых от времени условия:

- уникальность — в произвольный заданный момент времени никакие два различных кортежа R не имеют одного и того же значения для Ai, Aj, ..., Ak;
- минимальность — ни один из атрибутов Ai, Aj, ..., Ak не может быть исключен из K без нарушения уникальности.

Каждое отношение обладает хотя бы одним возможным ключом, поскольку по меньшей мере комбинация всех его атрибутов удовлетворяет условию уникальности. Один из возможных ключей (выбранный произвольным образом) принимается за его первичный ключ. Остальные возможные ключи, если они есть, называются альтернативными ключами.

Упомянутые ранее и некоторые другие математические понятия явились теоретической базой для создания реляционных СУБД, разработки соответствующих языковых средств и программных систем, обеспечивающих их высокую производительность, и создания основ теории проектирования баз данных. Однако для массового пользователя реляционных СУБД можно с успехом использовать неформальные эквиваленты этих понятий:

- отношение — *таблица* (иногда *файл*);
- кортеж — *строка* (иногда *запись*);
- атрибут — *столбец*, *поле*.

При этом принимается, что *запись* означает *экземпляр записи*, а *поле* означает *имя и тип поля*.

3.2. Реляционная база данных

Реляционная база данных — это совокупность отношений, содержащих всю информацию, которая должна храниться в БД. Однако пользователи могут воспринимать такую базу данных как совокупность таблиц. Так в табл. 3.1—3.10 показаны таблицы базы данных, построенные по инфологической модели базы данных "COOK" (рис. 2.8 и 2.9).

Отметим, что большинство примеров, связанных с изучением основ языка SQL (см. *главы 5—7*), используют информацию из базы данных "COOK". Поэтому представлялось целесообразным не только привести данные этой базы на прилагаемом к книге компакт-диске, но и в табл. 3.1—3.10, потратив шесть страниц текста. Урезаны лишь тексты рецептов (табл. 3.8) и число строк табл. 3.10.

Так как иллюстративная база данных создавалась для лекционного курса в 1989 году, когда существовали "смешные" цены, а также исчезнувшие названия статусов (коопторг) и городов (Ленинград), то авторы пытались несколько раз ее модифицировать. Однако поняв, что изменение цен, статусов и названий идет быстрее, чем подготовка и, тем более, выпуск издания, они решили сохранить в книге старые цены и названия.

Заметим также, что химический состав продуктов приведен для 1 кг их съедобной части: основные пищевые вещества (белки, жиры и углеводы) даны в граммах, а минеральные вещества (калий, кальций, натрий) и витамины (B2, PP, C) — в миллиграммах. Калорийность блюд определяется по массе и калорийности каждого из продуктов, входящих в это блюдо: для получения значения калорийности продукта исходят из того, что при окислении 1 г углеводов или белков в организме освобождается в среднем 4,1 ккал, а при окислении 1 г жиров — 9,3 ккал. Стоимость продуктов дана в рублях, а трудоемкость приготовления блюда — в копейках.

Таблица 3.1. БЛЮДА

КОД_БЛЮДА	БЛЮДО	КОД_ВИДА	ОСНОВА	ВЫХОД	ТРУД
1	Салат летний	1	Овощи	200,0	3
2	Салат мясной	1	Мясо	200,0	4
3	Салат витаминный	1	Овощи	200,0	4
4	Салат рыбный	1	Рыба	200,0	4
5	Паштет из рыбы	1	Рыба	120,0	5
6	Мясо с гарниром	1	Мясо	250,0	3

7	Сметана	1	Молоко	140,0	1
8	Творог	1	Молоко	140,0	2
9	Суп харчо	2	Мясо	500,0	5
10	Суп-пюре из рыбы	2	Рыба	500,0	6
11	Уха из судака	2	Рыба	500,0	5
12	Суп молочный	2	Молоко	500,0	3
13	Бастурма	3	Мясо	300,0	5
14	Бефстроганов	3	Мясо	210,0	6
15	Судак по-польски	3	Рыба	160,0	5
16	Драчена	3	Яйца	180,0	4
17	Морковь с рисом	3	Овощи	260,0	3
18	Сырники	3	Молоко	220,0	4
19	Омлет с луком	3	Яйца	200,0	5
20	Каша рисовая	3	Крупа	210,0	4
21	Пудинг рисовый	3	Крупа	160,0	6
22	Вареники ленивые	3	Молоко	220,0	4
23	Помидоры с луком	3	Овощи	260,0	4
24	Суфле из творога	3	Молоко	280,0	6
25	Рулет с яблоками	4	Фрукты	200,0	5
26	Яблоки печеные	4	Фрукты	160,0	3
27	Суфле яблочное	4	Фрукты	220,0	6
28	Крем творожный	4	Молоко	160,0	4
29	"Утро"	5	Фрукты	200,0	5
30	Компот	5	Фрукты	200,0	2
31	Молочный напиток	5	Молоко	200,0	2
32	Кофе черный	5	Кофе	100,0	1
33	Кофе на молоке	5	Кофе	200,0	2

Таблица 3.2. ВИДЫ_БЛЮД

КОД_ВИДА	ВИД
1	Закуска
2	Суп
3	Горячее
4	Десерт
5	Напиток

Таблица 3.3. ПРОДУКТЫ

КОД_ПРОДУКТА	ПРОДУКТ	БЕЛКИ	ЖИРЫ	УГЛЕВ	К	СА	NA	В2	РР	С
1	Говядина	189,0	124,0	0,0	3150	90	600	1,5	28,0	0
2	Судак	190,0	80,0	0,0	1870	270	0	1,1	10,0	30
3	Масло	60,0	825,0	90,0	230	220	740	0,1	1,0	0
4	Майонез	31,0	670,0	26,0	480	280	0	0,0	0,0	0
5	Яйца	127,0	115,0	7,0	1530	550	710	4,4	1,9	0
6	Сметана	26,0	300,0	28,0	950	850	320	1,0	1,0	2
7	Молоко	28,0	32,0	47,0	1460	1210	500	1,3	1,0	10
8	Творог	167,0	90,0	13,0	1120	1640	410	2,7	4,0	5
9	Морковь	13,0	1,0	70,0	2000	510	210	0,7	9,9	50
10	Лук	17,0	0,0	95,0	1750	310	180	0,2	2,0	100
11	Помидоры	6,0	0,0	42,0	290	140	400	0,4	5,3	250
12	Зелень	9,0	0,0	20,0	340	275	75	1,2	4,0	380
13	Рис	70,0	6,0	773,0	540	240	260	0,4	16,0	0
14	Мука	106,0	13,0	732,0	1760	240	120	1,2	22,0	0
15	Яблоки	4,0	0,0	113,0	2480	160	260	0,3	3,0	130
16	Сахар	0,0	0,0	998,0	30	20	10	0,0	0,0	0
17	Кофе	127,0	36,0	9,0	9710	180	180	0,3	1,8	0

Таблица 3.4. ПОСТАВЩИКИ

КОД_ПОСТАВЩИКА	НАЗВАНИЕ	СТАТУС	ГОРОД	АДРЕС	ТЕЛЕФОН
1	СЫТНЫЙ	рынок	Ленинград	Сытнинская,3	2329916
2	ПОРТОС	кооператив	Резекне	Садовая,27	317664
3	ШУШАРЫ	совхоз	Пушкин	Новая, 17	4705038
4	ТУЛЬСКИЙ	универсам	Ленинград	Тульский,5	2710837
5	УРОЖАЙ	кооптorг	Луга	Песчаная,19	789000
6	ЛЕТО	агрофирма	Ленинград	Пулковская,8	2939729
7	ОГУРЕЧИК	ферма	Паневежис	Укмерге,15	127331
8	КОРЮШКА	кооператив	Йыхви	Нарвское ш.,64	432123

Таблица 3.5. ПОСТАВКИ

КОД_ПОСТАВЩИКА	КОД_ПРОДУКТА	ЦЕНА	К_ВО	ДАТА
1	11	1,50	50	14.05.1989
1	12	3,00	10	10.05.1989
1	15	2,00	170	10.05.1989
2	1	3,60	300	14.05.1989
2	6	3,60	80	14.05.1989
2	5	1,80	100	14.05.1989
3	7	0,40	200	14.05.1989
3	12	2,50	20	14.05.1989
3	15	1,50	200	14.05.1989
4	4	2,04	50	09.05.1989
4	13	0,88	150	09.05.1989
4	16	0,94	200	09.05.1989
4	17	4,50	50	09.05.1989
5	4	3,00	50	14.05.1989
5	10	0,50	130	14.05.1989
5	13	1,20	40	14.05.1989
5	14	0,50	70	14.05.1989
5	16	1,00	50	14.05.1989
6	10	0,70	90	10.05.1989
7	1	4,20	70	10.05.1989
7	3	4,00	250	10.05.1989
7	6	2,20	140	10.05.1989
7	8	1,00	150	14.05.1989
8	5	2,00	70	10.05.1989
8	11	1,00	100	10.05.1989

Таблица 3.6. МЕНЮ

СТРОКА	КОД_ТРАПЕЗЫ	КОД_БЛЮДА	ДАТА
1	1	3	15.05.1989
2	1	6	15.05.1989
3	1	19	15.05.1989
4	1	21	15.05.1989

```
     5           1        31 15.05.1989
     6           1        32 15.05.1989
     7           2         1 15.05.1989
     8           2         6 15.05.1989
     9           2         9 15.05.1989
    10           2        12 15.05.1989
    11           2        14 15.05.1989
    12           2        16 15.05.1989
    13           2        18 15.05.1989
    14           2        26 15.05.1989
    15           2        28 15.05.1989
    16           3         6 15.05.1989
    17           3         8 15.05.1989
    18           3        20 15.05.1989
    19           3        16 15.05.1989
    20           3        30 15.05.1989
    21           3        31 15.05.1989
```

Таблица 3.7. ТРАПЕЗЫ

```
  КОД_
КОД_ТРАПЕЗЫ ТРАПЕЗА
----------- -------
          1 Завтрак
          2 Обед
          3 Ужин
```

Таблица 3.8. РЕЦЕПТЫ

```
   КОД_                                                          ВА-
ИД БЛЮДА РЕЦЕПТ                                                  РИАНТ
-- ----- ---------------------------------------------------------- -----
 1     1 Помидоры и яблоки нарезать кружочками, положить помидоры в
 2     2 Вареное охлажденное мясо, свежую зелень и вареную очищенну
 3     3 Зелень мелко нарезать и положить горкой в салатник. Свежие
 4     4 Вареные рыбу и морковь нарезать небольшими ломтиками, а по
 5     5 Филе судака припустить до готовности. Морковь спассеровать
 6     6 Мясо вареное нарезать тонкими кусочками, гарнировать ломти
```

```
 7     7 Сметану положить в стакан конической формы. Подавать на ст
 8     8 Протертый творог положить в салатник или на мелкую тарелку
 9     9 Грудинку говядины нарезать на куски и варить до полуготовн
10    10 Филе судака припустить с маслом и веточкой сельдерея. Гото
11    11 Судак очистить, разделать на филе с кожей и ребрами, нарез
12    12 Промытый рис варить в кипящей воде 5-6 минут, откинуть на
13    13 Мясо нарезать кубиками (30 г), добавить рубленый лук, соль
14    14 Говядину нарезать на широкие ломти толщиной до 20 мм, отби
15    15 Подготовленную рыбу нарезать на порционные куски; кожу на
16    16 Сырые яйца смешать с пшеничной мукой, добавить сметану и с
17    17 Нарезать кружочками морковь, положить в посуду, добавить п
18    18 В протертый творог добавить яйца, сахарный песок, соль и п
19    19 К свежим яйцам добавляют холодное молоко, соль и тщательно
20    20 Рис сварить в воде до полуготовности, добавить кипящее мол
21    21 Готовую рисовую рассыпчатую кашу смешать с холодным молоко
22    22 В протертый творог положить яйца, соль, сахар, размягченно
23    23 Спассеровать на масле мелко нарезанный лук. В конце пассер
24    24 В протертый творог положить сметану, яичные желтки, растер
25    25 Очистить яблоки, разрезать каждое на 8 частей и каждую час
26    26 Не прорезая насквозь, удалить из яблок сердцевину и семена
27    27 Запеченные яблоки (см. БЛЮДО 26) охладить и протереть чере
28    28 Яйца размешать с сахаром и, взбивая, протереть, не доводя
29    29 Очищенную и промытую морковь натереть на терке, залить вод
30    30 Яблоки очистить от кожицы, удалить сердцевину и нарезать.
31    31 Яблоки натереть на терке и отжать из них сок. В стакан вли
32    32 Вскипятить воду в кофейнике со щепоткой соли или 1/2 чайно     1
33    32 Кофеварку или кастрюлю сполоснуть горячей водой, положить     2
34    33 Сварить черный кофе, как указано в БЛЮДЕ 32, взяв меньшее
```

Таблица 3.9. СОСТАВ

КОД_ БЛЮДА	КОД_ ПРОДУКТА	ВЕС	КОД_ БЛЮДА	КОД_ ПРОДУКТА	ВЕС	КОД_ БЛЮДА	КОД_ ПРОДУКТА	ВЕС
1	11	100	12	7	350	21	13	70
1	15	80	12	13	35	21	6	30
1	12	5	12	3	5	21	3	20
1	4	15	12	16	5	21	5	20
2	1	65	13	1	180	21	16	15
2	9	40	13	11	100	22	8	140
2	11	35	13	10	40	22	6	30

2	12	20	13	12	20	22	14	20	
2	5	20	13	3	5	22	16	15	
2	4	20	14	1	90	22	5	8	
3	11	55	14	7	50	23	11	250	
3	15	55	14	6	20	23	10	65	
3	6	50	14	10	10	23	3	20	
3	12	20	14	3	5	24	8	80	
3	10	15	14	12	5	24	7	100	
3	16	5	14	14	3	24	5	40	
4	2	50	15	2	100	24	6	30	
4	11	50	15	9	20	24	16	20	
4	4	40	15	5	20	24	3	10	
4	9	35	15	3	20	24	14	10	
4	5	20	15	10	10	25	15	120	
4	12	5	15	12	5	25	16	35	
5	2	80	16	5	120	25	14	30	
5	9	40	16	7	35	25	8	20	
5	3	25	16	6	15	25	3	20	
5	12	5	16	14	9	26	15	150	
6	1	80	16	3	5	26	16	20	
6	11	150	17	9	150	26	3	2	
6	4	30	17	7	50	27	15	50	
6	12	10	17	13	25	27	7	150	
7	6	125	17	3	20	27	5	80	
7	16	15	17	12	10	27	16	35	
8	8	75	17	14	5	27	3	2	
8	6	50	18	8	140	28	8	100	
8	16	15	18	6	30	28	5	20	
9	1	80	18	14	15	28	6	20	
9	10	30	18	5	10	28	16	15	
9	11	25	18	16	15	28	3	10	
9	13	35	19	5	120	29	15	150	
9	12	15	19	7	45	29	9	200	
9	3	15	19	10	20	29	16	15	
10	2	70	19	3	15	30	15	70	
10	7	250	20	13	50	30	16	10	
10	3	20	20	7	75	31	7	150	
10	14	15	20	15	75	31	15	150	
10	12	5	20	16	10	31	16	25	
11	2	100	20	3	5	32	17	8	
11	9	20				33	17	8	
11	10	20				33	16	25	
11	3	5				33	7	75	
11	12	2							

Таблица 3.10. ВЫБОР

МЕСТО	СТРОКА	МЕСТО	СТРОКА	МЕСТО	СТРОКА	МЕСТО	СТРОКА	МЕСТО	СТРОКА
1	1	6	1	11	1	16	2	21	1
1	3	6	4	11	3	16	3	21	3
1	5	6	6	11	5	16	6	21	6
1	8	6	7	11	8	16	7	21	7

1	10	6	10	11	10	16	9	21	10
1	11	6	13	11	11	16	11	21	13
1	14	6	15	11	14	16	14	21	14
1	17	6	16	11	17	16	17	21	16
1	18	6	19	11	18	16	18	21	18
1	20	6	20	11	20	16	21	21	20
2	2	7	2	12	1	17	1	22	1
2	4	7	3	12	3	17	4	22	4
2	6	7	5	12	5	17	5	22	6
2	7	7	8	12	8	17	7	22	7
2	9	7	10	12	10	17	9	22	10
2	11	7	11	12	13	17	11	22	13
2	15	7	14	12	14	17	15	22	14
2	16	7	17	12	16	17	16	22	16
2	19	7	18	12	18	17	19	22	19
2	21	7	21	12	20	17	21	22	20
3	2	8	2	13	2	18	1	23	1
3	3	8	4	13	4	18	4	23	4
3	6	8	5	13	6	18	5	23	6
3	7	8	8	13	7	18	7	23	7
3	9	8	10	13	9	18	9	23	10
3	11	8	11	13	11	18	13	23	13
3	14	8	14	13	15	18	15	23	14
3	17	8	17	13	16	18	16	23	16
3	18	8	19	13	19	18	19	23	19
3	21	8	20	13	21	18	21	23	20
4	1	9	1	14	2	19	1	24	1
4	4	9	3	14	4	19	4	24	4
4	5	9	5	14	6	19	5	24	6
4	7	9	8	14	7	19	7	24	7
4	9	9	10	14	9	19	9	24	10
4	12	9	13	14	11	19	12	24	12
4	15	9	14	14	15	19	15	24	15
4	16	9	16	14	16	19	16	24	16
4	18	9	18	14	19	19	18	24	18
4	21	9	20	14	21	19	21	24	20
5	1	10	1	15	2	20	1	25	2
5	3	10	3	15	3	20	4	25	3
5	6	10	5	15	6	20	5	25	5
5	7	10	8	15	7	20	7	25	8
5	10	10	10	15	9	20	9	25	10
5	13	10	13	15	11	20	13	25	11
5	14	10	14	15	14	20	14	25	14
5	16	10	16	15	17	20	16	25	17
5	19	10	18	15	18	20	19	25	18
5	20	10	20	15	21	20	21	25	21

В рассмотренной базе данных, как и в любой другой реляционной базе данных:

- каждая таблица состоит из однотипных строк и имеет уникальное имя;
- строки имеют фиксированное число полей (столбцов) и значений (множественные поля и повторяющиеся группы недопустимы). Иначе говоря,

в каждой позиции таблицы на пересечении строки и столбца всегда имеется в точности одно значение или ничего;

- строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку такой таблицы;
- столбцам таблицы однозначно присваиваются имена, и в каждом из них размещаются однородные значения данных (даты, фамилии, целые числа или денежные суммы);
- полное информационное содержание базы данных представляется в виде явных значений данных, и такой метод представления является единственным. В частности, не существует каких-либо специальных "связей" или указателей, соединяющих одну таблицу с другой. Так, связи между строкой с `КОД_БЛЮДА=9` табл. 3.1 (`БЛЮДА`) и строкой с `КОД_ПРОДУКТА=13` табл. 3.3 (`ПРОДУКТЫ`) (для приготовления харчо нужен рис), представляются не с помощью указателей, а благодаря существованию в табл. 3.9 (`СОСТАВ`) строки, в которой код блюда равен 9, а код продукта — 13;
- при выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию. Этому способствует наличие имен таблиц и их столбцов, а также возможность выделения любой строки или любого набора строк с указанными признаками (например, продуктов, не содержащих углеводов, но имеющих в своем составе витамин С).

3.3. Манипулирование реляционными данными

В *главе 11* будет показано, что стремление к минимизации числа таблиц для хранения данных может привести к возникновению различных проблем при их обновлении, и будут даны рекомендации по разбиению некоторых больших таблиц на несколько маленьких. Но как сформировать требуемый ответ, если нужные для него данные хранятся в разных таблицах?

Предложив реляционную модель данных, Эдгар Кодд создал и инструмент для удобной работы с отношениями — реляционную алгебру. Каждая операция этой алгебры использует одну или несколько таблиц (отношений) в качестве ее операндов и продуцирует в результате новую таблицу, т. е. позволяет "разрезать" или "склеивать" таблицы (рис. 3.2).

Созданы языки манипулирования данными, позволяющие реализовать все операции реляционной алгебры и практически любые их сочетания. Среди

них наиболее распространены SQL (Structured Query Language — *структуризованный язык запросов*) и QBE (Quere-By-Example — *запросы по образцу*) [5, 6]. Оба относятся к языкам очень высокого уровня, с помощью которых пользователь указывает, какие данные необходимо получить, не уточняя процедуру их получения.

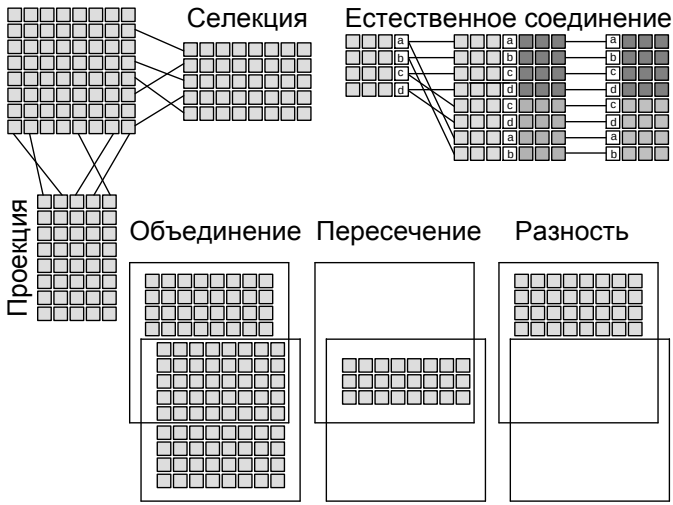


Рис. 3.2. Некоторые операции реляционной алгебры

С помощью единственного запроса на любом из этих языков можно соединить несколько таблиц во временную таблицу и вырезать из нее требуемые строки и столбцы (селекция и проекция).

3.3.1. Обновление отношений

Отношения (например, отношение R, с атрибутами A1, A2, ..., An) могут дополняться, удаляться или изменяться.

Добавление

Если эту операцию обозначить ADD и применить к отношению R, то можно записать:

ADD (R; $A_1 = d_1$, $A_2 = d_2$, ..., $A_n = d_n$)

или при фиксированном порядке имен атрибутов

ADD (R; d_1, d_2, ..., d_n),

где di — значение i-го атрибута добавляемого кортежа.

Пример 3.1. Добавить в таблицу БЛЮДА (табл. 3.1) блюдо Шашлык:

```
ADD(Блюда; Код_блюда=35, Блюдо='Шашлык', Код_вида=3, Основа='Мясо',
Выход=200, Труд=6)
```

или

```
ADD (Блюда; 35, 'Шашлык', 3, 'Мясо', 200, 6)
```

Цель операции — добавить указанный кортеж в определенное отношение. Результат операции может быть не согласован с целями операции по следующим причинам:

- добавляемый кортеж не соответствует описанию (схеме) определенного отношения;
- некоторые значения кортежа не принадлежат соответствующим доменам (например, по описанию значения атрибута должны быть целыми числами, а в операции указано текстовое значение);
- описанный кортеж совпадает по ключу с кортежем, уже находящимся в отношении.

В каждом из этих случаев операция ADD (R; d_1, d_2, ..., d_n) оставляет отношение R неизменным и некоторым образом сообщает об ошибке.

Пример 3.2. Добавить в таблицу ПРОДУКТЫ (табл. 3.3) Шпик:

```
ADD (Продукты; Код_продукта=8, Продукт='Шпик', Белки=110, Жиры=850,
Углеводы='Отсутствуют', K=2700, CA=120, NA=710, B2=1,2, PP=32, C=0)
```

Эта операция не допускается по всем из перечисленных причин (продукт с номером Код_продукта=8 уже существует, Углеводы именуется Углев и оцениваются в граммах, а не текстовой величиной).

Удаление

Эта операция, которую мы обозначим DEL, вводится для уничтожения сделанного. Для приведенного ранее отношения R она записывается в виде

DEL (R; A_1= d_1, A_2= d_2, ..., A_n= d_n)

или при фиксированном порядке имен атрибутов

DEL (R; d_1, d_2, ..., d_n).

Пример 3.3. Из таблицы ПОСТАВЩИКИ (табл. 3.5) удалить поставщика ПОРТОС:

```
DEL (Поставщики; Код_поставщика=3, Название='ПОРТОС', Город='Резекне')
```

или

```
DEL (Поставщики; 3, 'ПОРТОС', 'Резекне')
```

В действительности нет необходимости задавать так много информации, чтобы однозначно определить кортеж, который надо удалить. Достаточно определить значения некоторого ключа. Если К = {Bp, Bj, ..., Bn} является ключом, то можно использовать следующую форму записи:

DEL (R; B_1= e_1, B_2= e_2, ..., B_n= e_n).

Краткая форма записи операции удаления:

```
DEL (Поставщики; 3)
```

Операция удаления не выполняется лишь в тех случаях, когда заданный кортеж отсутствует в отношении. Тогда отношение остается неизменным и сообщается об ошибке условия. Ограничения на удаление последнего кортежа из отношения не накладывается; пустое отношение допускается.

Изменение

Вместо того чтобы добавлять или удалять целый кортеж отношения, можно изменить лишь часть кортежа. Для R при {C1, C2, ..., Cp} из {A1, A2.....An} операция изменения имеет вид:

CH (R; $A_1 = d_1$ $A_2 = d_2$, ..., $A_n = d_n$; $C_1 = e_1$ $C_2 = e_2$..... $C_p = e_p$).

Или если K = {B_1= e_1, B_2= e_2, ..., B_n= e_n} является ключом, то

$C_1 = e_1$ $C_2 = e_2$..... $C_p = e_p$.

Пример 3.4. Уменьшить количество лука в бастурме до 30 г.

```
CH (Состав; Код_блюда=13, Код_продукта=10, Вес=40; Вес=30)
```

или

```
CH (Состав; Код_блюда=13; Вес=30)
```

Операция изменения является наиболее удобной. Тот же результат может быть получен с помощью операции добавления, следующей за операцией удаления. Таким образом, все возможные ошибки операции добавления и удаления присущи и операции изменения: указанный в операции кортеж не существует, изменения имеют неправильный формат или используемые значения не принадлежат существующему домену или измененный кортеж имеет тот же ключ, что и кортеж, уже принадлежащий отношению.

3.3.2. Реляционные операции

Операции обновления — это операции над кортежами. В данном разделе мы будем рассматривать операторы, которые включают в себя целое отношение, т. е. позволяют "разрезать" и "склеивать" таблицы.

Унарные операции

Эти операции служат для преобразования одного отношения в другое, являющееся подмножеством первого.

Селекция (горизонтальное подмножество) R создается из тех строк R, которые удовлетворяют заданным условиям, соединяемым логическими операторами AND, OR, NOT AND и NOT OR. Каждое условие может состоять из имени какого-либо столбца R, оператора сравнения (< , <= , = , <> , > , >=) и константы или имени другого столбца R. При этом сравниваемые столбцы должны быть определены на одном и том же домене.

R				R[B=1]				R[D= ε OR B>1]			
A	B	C	D	A	B	C	D	A	B	C	D
—	—	—	—	—	—	—	—	—	—	—	—
a	1	Б	ε	a	1	Б	ε	a	1	Б	ε
b	7	Г	ω	c	1	Я	ε	b	7	Г	ω
c	1	Я	ε	a	1	Г	ω	c	1	Я	ε
a	1	Г	ω					b	7	Б	ε
b	7	Б	ε								

Пример 3.5. Найти блюда, в состав которых входит более 15 г зелени (`Код_продукта=12`):

```
Состав [Код_продукта=12 AND Вес > 15]:

КОД_БЛЮДА КОД_ПРОДУКТА  ВЕС
--------- ------------ ----
        2           12   20
        3           12   20
       13           12   20
```

Проекция (вертикальное подмножество) R создается из указанных столбцов R (в заданном порядке) с последующим исключением избыточных дубликатов строк.

R				R[AB1]		R[DCA]		
A	B	C	D	A	B	D	C	A
—	—	—	—	—	—	—	—	—
a	1	Б	ε	a	1	ε	Б	a
b	7	Г	ω	b	7	ω	Г	b
c	1	Я	ε	c	1	ε	Я	c
a	1	Г	ω			ω	Г	a
b	7	Б	ε			ε	Б	b

Пример 3.6. Найти всех поставщиков продуктов и перечень всех городов, в которых они расположены:

```
Поставщики [Название Город]

НАЗВАНИЕ ГОРОД
-------- ---------
СЫТНЫЙ   Ленинград
ПОРТОС   Резекне
ШУШАРЫ   Пушкин
ТУЛЬСКИЙ Ленинград
УРОЖАЙ   Луга
ЛЕТО     Ленинград
ОГУРЕЧИК Паневежис
КОРЮШКА  Йыхви
```

Бинарные операции

К двум отношениям с одной и той же схемой могут быть применены операции над множествами.

Если R и S — два отношения со схемой ABCD, то отношения R U S, R ∩ H и R – S также имеют схему ABCD. При этом S и, например, S1 рассматриваются как одинаковые таблицы, так как в реляционных моделях схемы оцениваются по именам, а не по последовательности столбцов.

R				S				S1			
A	B	C	D	A	B	C	D	C	B	D	A
—	—	—	—	—	—	—	—	—	—	—	—
a	1	Б	ε	c	4	Г	&	Г	4	&	c
b	7	Г	ω	a	1	Г	ε	Г	1	ε	a
c	1	Я	ε	b	7	Г	ω	Г	7	ω	b
a	1	Г	ω	c	1	Я	ω	Я	1	ω	c
b	7	Б	ε								

Объединение R U S содержит те строки, которые есть либо в R, либо в S, либо в обеих таблицах.

R				S				R U S			
A	B	C	D	A	B	C	D	A	B	C	D
—	—	—	—	—	—	—	—	—	—	—	—
a	1	Б	ω	c	4	Г	&	a	1	Б	ω
b	7	Г	ε	a	1	Г	ε	b	7	Г	ε
c	1	Я	ω	b	7	Г	ω	c	1	Я	ω
a	1	Г	ε	c	1	Я	ω	a	1	Г	ε
b	7	Б	ω					b	7	Б	ω
								c	1	Я	&
								b	7	Г	ω

Если, например, разные пользователи базы данных "COOK" вводят во временные таблицы информацию о новых блюдах, то с помощью операции объединения ее легко добавить в таблицы БЛЮДА и СОСТАВ, не заботясь о возможном появлении нескольких одинаковых рецептов. Об этом же можно не заботиться при дополнении базы данных информацией из аналогичной базы данных другой организации, так как при объединении в таблицы добавляются только уникальные строки.

Разность R – S содержит только те строки, которые есть в R, но отсутствуют в S. Соответственно, S – R содержит те строки, которые есть в S, но отсутствуют в R.

R				S				R – S			
A	B	C	D	A	B	C	D	A	B	C	D
—	—	—	—	—	—	—	—	—	—	—	—
a	1	Б	ω	c	4	Г	&	a	1	Б	ω
b	7	Г	ε	a	1	Г	ε	b	7	Г	ε
c	1	Я	ω	b	7	Г	ω	b	7	Б	ω
a	1	Г	ε	c	1	Я	ω				
b	7	Б	ω								

Если, например, администратор базы данных "COOK" "раздобыл" аналогичную базу данных в другой организации, то путем вычитания однотипных таблиц легко выявить те сведения, которые можно было бы добавить в базу данных.

Пересечение R ∩ S содержит только те строки, которые есть и в R, и в S. Заметим, что пересечение может быть определено как разность отношений: R ∩ S = R – (R – S).

С помощью этой операции можно, например, выделить из таблиц базы данных "COOK" поставщиков, которые поставляют продукты, используемые в существующих рецептах:

```
Продукты [Код_продукта] ∩ Состав [Код_продукта]
```

и т. п.

Соединения — это операторы, позволяющие получить декартово произведение двух отношений, а затем выделить его определенное подмножество.

Декартово произведение двух отношений — это множество упорядоченных пар из кортежей: оно содержит все возможные строки, составленные сцеплением строки из первого отношения (начало создаваемой строки) со строкой из второго отношения. Следовательно, число строк произведения равно произведению числа строк сомножителей, а их длина — сумме длин строк сомножителей. Так декартово произведение R × T содержит шесть столбцов и десять строк.

R				T		R x T					
A	B	C	D	E	F	A	B	C	D	E	F
—	—	—	—	—	—	—	—	—	—	—	—
a	1	Б	ω	1	&	a	1	Б	ω	1	&
b	7	Г	ε	7	$	a	1	Б	ω	7	$
c	1	Я	ω			b	7	Г	ε	1	&
a	1	Г	ε			b	7	Г	ε	7	$
b	7	Б	ω			c	1	Я	ω	1	&
						c	1	Я	ω	7	$
						a	1	Г	ε	1	&
						a	1	Г	ε	7	$
						b	7	Б	ω	1	&
						b	7	Б	ω	7	$

Соединение R с T по столбцам i и j (R[i Θ j]T) содержит те строки декартова произведения R × T, в которых значение из i-го столбца R находится в отношении Θ (< , <= , = , < > , >=, >) со значением из j-го столбца T. При этом сравниваемые столбцы должны быть определены на одном и том же домене. Кроме того, i и j могут рассматриваться не только как отдельные столбцы, но и как совместимые (совпадающие по числу, порядку и типам данных) множества столбцов.

Имеется несколько вариантов операции соединения:

- *эквисоединение* — это соединение, полученное при равенстве значений i и j (Θ = "=");
- *естественное соединение* — это эквисоединение, в котором исключен столбец j;
- *композиция* — это эквисоединение, где исключены столбцы, по которым производилось соединение (столбцы i и j);
- *тета-соединение* — это соединения, полученные при любом значении Θ, кроме "=".

Эквисоединение R[B=E]						Естественное соединение (без E)					Композиция (без E и B)			
A	B	C	D	E	F	A	B	C	D	F	A	C	D	F
—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
a	1	Б	ω	1	&	a	1	Б	ω	&	a	Б	ω	&
b	7	Г	ε	7	$	b	7	Г	ε	$	b	Г	ε	$
c	1	Я	ω	1	&	c	1	Я	ω	&	c	Я	ω	&
a	1	Г	ε	1	&	a	1	Г	ε	&	a	Г	ε	&
b	7	Б	ω	7	$	b	7	Б	ω	$	b	Б	ω	$

Пример 3.7. Если вам надо попасть из Санкт-Петербурга в Сочи через Москву и в вашей базе данных есть расписание полетов самолетов из Санкт-Петербурга в Москву и из Москвы в Сочи, то целесообразно выполнить тета-соединение расписаний с условием: время прибытия в Москву меньше времени вылета в Сочи. Полученное отношение "Транзит" упростит процедуру подбора подходящих рейсов.

Деление R : S. Для выполнения такой операции делитель S должен состоять из атрибутов (например, C D), являющихся подмножеством атрибутов делимого R (например, A B C D F). При этом частное R : S, если оно существует, состоит из тех атрибутов R, которых нет в S (т. е. атрибутов A B F).

В процессе формирования R : S кортеж включается в него только тогда, когда его декартово произведение с делителем S содержится в делимом R.

R					S		R : S		
A	B	C	D	F	C	D	A	B	F
—	—	—	—	—	—	—	—	—	—
b	7	Б	ω	$	Б	ω	a	1	&
a	1	Б	ω	&	Г	ε	c	7	$
b	1	Г	ω	&					
a	1	Г	ε	&					
c	7	Г	ε	$					
a	7	Б	ε	&					
c	7	Б	ω	$					

Следует отметить, что операция деления может быть выражена через другие операции:

1. Получить и запомнить проекцию делимого, состоящую из атрибутов, не принадлежащих делителю.
2. Получить декартово произведение этой проекции на делитель.
3. Вычесть из полученного произведения делимое.
4. Получить проекцию разности, состоящую из атрибутов, не принадлежащих делителю.
5. Определить частное как разность между проекциями, полученными в пп. 1 и 4.

Переименование атрибутов. Иногда появляется необходимость соединить отношение с самим собой, игнорировав при этом связи по каким-либо его атрибутам. Это можно сделать копированием (проектированием один к одному) такого отношения и переименованием указанных атрибутов. Встречаются и другие случаи, когда возникает необходимость в переименовании ряда атрибутов отношения. Для выполнения операции переименования атрибута A в атрибут B отношения R воспользуемся командой вида:

REN (R; A = B)

ЧАСТЬ II

ЯЗЫК SQL. ИЗВЛЕЧЕНИЕ ДАННЫХ

Глава 4

Основы SQL

4.1. Стандарты языка SQL

В *разд. 3.1* рассматривались революционные предложения Эдгара Кодда по созданию реляционной модели данных и инструмента для удобной работы с отношениями — реляционной алгебры. Он же инициировал разработку для этой модели языка SEQUEL (Structured English Query Language, структурированный английский язык для запросов), который впоследствии по юридическим соображениям был переименован в *SQL* (Structured Query Language, структурированный язык запросов). Официальным произношением стало [es kju:' el] — *эс-кью-эл*. Несмотря на это, даже англоязычные специалисты по-прежнему часто называют SQL *сиквел*, вместо *эс-кью-эл* (по-русски также часто говорят "эс-ку-эль"). Целью разработки было создание простого непроцедурного языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования.

К началу 1980-х годов SQL завоевал популярность как язык реляционных СУБД и привлек внимание Американского национального института по стандартизации (American National Standards Institute, ANSI), который в 1986, 1989, 1992, 1999 и 2003 годах выпустил стандарты языка SQL. В 1989 году SQL был включен в стандарты международной организации по стандартизации ISO (SQL:1989), а затем были приняты и опубликованы стандарты SQL:1992, SQL:1999 и SQL:2003. В настоящее время все производители распространенных реляционных СУБД поддерживают с различной степенью соответствия стандарт SQL:2003 [6]. Каждый из новых стандартов отличался от предыдущего улучшением синтаксиса, расширением типов данных, новыми процедурными расширениями, которые сможет применить любое приложение, имеющее доступ к базе данных, средств для работы с Java и XML-документами.

4.2. Почему SQL?

Все языки манипулирования данными (ЯМД), созданные до появления реляционных баз данных и разработанные для многих СУБД, были ориентированы на операции с данными, представленными в виде логических записей файлов. Это требовало от пользователей детального знания организации хранения данных и достаточных усилий для указания не только того, какие данные нужны, но и того, где они размещены, и как шаг за шагом получить их.

Рассматриваемый же далее непроцедурный язык SQL ориентирован на операции с данными, представленными в виде логически взаимосвязанных совокупностей таблиц. Особенность предложений этого языка состоит в том, что они ориентированы в большей степени на конечный результат обработки данных, чем на процедуру этой обработки. SQL сам определяет, где находятся данные, какие индексы и даже наиболее эффективные последовательности операций следует использовать для их получения: не надо указывать эти детали в запросе к базе данных.

Для иллюстрации различий между ЯМД рассмотрим следующую ситуацию. Пусть, например, вы собираетесь посмотреть кинофильм и хотите воспользоваться для поездки в кинотеатр услугами такси. Одному шоферу такси достаточно сказать название фильма — и он сам найдет вам кинотеатр, в котором показывают нужный фильм. (Подобным же образом, самостоятельно, отыскивает запрошенные данные SQL.)

Для другого шофера такси вам, возможно, потребуется самому узнать, где демонстрируется нужный фильм, и назвать кинотеатр. Тогда водитель должен найти адрес этого кинотеатра. Может случиться и так, что вам придется самому узнать адрес кинотеатра и предложить водителю проехать к нему по таким-то и таким-то улицам. В худшем случае вам, может быть, даже придется по дороге давать указания: "Повернуть налево... проехать пять кварталов... повернуть направо...". (Аналогично больший или меньший уровень детализации запроса приходится создавать пользователю в разных СУБД, не имеющих языка SQL.)

Появление теории реляционных баз данных и предложенного Коддом языка запросов "alpha", основанного на реляционном исчислении [1], инициировало разработку ряда языков запросов, которые можно отнести к двум классам:

1. Алгебраические языки, позволяющие выражать запросы средствами специализированных операторов, применяемых к отношениям: `JOIN` — соединить, `INTERSECT` — пересечь, `SUBTRACT` — вычесть и т. д. (см. *разд. 3.3*).
2. Языки исчисления предикатов, представляющие собой набор правил для записи выражения, определяющего новое отношение из заданной совокуп-

ности существующих отношений. Другими словами исчисление предикатов есть метод определения того отношения, которое нам желательно получить (как ответ на запрос) из отношений, уже имеющихся в базе данных.

Как указывалось в *разд. 4.1*, в начале 1980-х годов SQL "победил" другие языки запросов и стал фактическим стандартом таких языков для профессиональных реляционных СУБД. В 1989 году он стал международным стандартом языка баз данных и начал внедряться во все распространенные СУБД персональных компьютеров. Почему же это произошло?

Непрерывный рост быстродействия, а также снижение энергопотребления, размеров и стоимости компьютеров привели к резкому расширению возможных рынков их сбыта, круга пользователей, разнообразия типов и цен. Естественно, что расширился спрос на разнообразное программное обеспечение.

Борясь за покупателя, фирмы, производящие программное обеспечение, стали выпускать на рынок все более и более интеллектуальные и, следовательно, объемные программные комплексы. Приобретая (желая приобрести) такие комплексы, многие организации и отдельные пользователи часто не могли разместить их на собственных ЭВМ, однако не хотели и отказываться от нового сервиса. Для обмена информацией и ее обобществления были созданы сети ЭВМ, где обобществляемые программы и данные стали размещать на специальных обслуживающих устройствах — файловых серверах.

СУБД, работающие с файловыми серверами, позволяют множеству пользователей разных ЭВМ (иногда расположенных достаточно далеко друг от друга) получать доступ к одним и тем же базам данных. При этом упрощается разработка различных автоматизированных систем управления организациями, учебных комплексов, информационных и других систем, где множество сотрудников (учащихся) должны использовать общие данные и обмениваться создаваемыми в процессе работы (обучения). Однако при такой идеологии вся обработка запросов из программ или с терминалов пользовательских ЭВМ выполняется на этих же ЭВМ. Поэтому для реализации даже простого запроса ЭВМ часто должна считывать из файлового сервера и (или) записывать на сервер целые файлы, что ведет к конфликтным ситуациям и перегрузке сети.

Для исключения указанных и некоторых других недостатков была предложена *технология "клиент-сервер"*, по которой запросы пользовательских ЭВМ (*клиент*) обрабатываются на специальных серверах баз данных (*сервер*), а на ЭВМ возвращаются лишь результаты обработки запроса. При этом, естественно, нужен единый язык общения с сервером и в качестве такого языка выбран SQL. Поэтому все современные версии профессиональных реляционных СУБД (DB2, Oracle, SQL Server, Sybase, PostgreSQL, MySQL) и даже нереля-

ционных СУБД (например, Adabas) используют технологию "клиент-сервер" и язык SQL. К тому же приходят разработчики СУБД персональных ЭВМ, многие из которых уже сегодня снабжены языком SQL.

Бытует мнение, поскольку большая часть запросов формулируется на SQL, то практически безразлично, что это за СУБД — был бы SQL.

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим (менее 30) набором основных предложений. SQL может использоваться как интерактивный (для выполнения запросов) и как встроенный (для построения прикладных программ) язык. В нем существуют:

- предложения определения данных (определение баз данных, а также определение и уничтожение таблиц и индексов);
- запросы на выбор данных (предложение `SELECT`);
- предложения модификации данных (добавление, удаление и изменение данных);
- предложения управления данными (предоставление и отмена привилегий на доступ к данным, управление транзакциями и другие). Кроме того, он предоставляет возможность выполнять в этих предложениях:
  - арифметические вычисления (включая разнообразные функциональные преобразования), обработку текстовых строк и выполнение операций сравнения значений арифметических выражений и текстов;
  - упорядочение строк и (или) столбцов при выводе содержимого таблиц на печать или экран дисплея;
  - создание представлений (виртуальных таблиц), позволяющих пользователям иметь свой взгляд на данные без увеличения их объема в базе данных (см. *разд. 4.3* и *7.3.4*);
  - запоминание выводимого по запросу содержимого таблицы, нескольких таблиц или представления в другой таблице (реляционная операция присваивания);
  - агрегатирование данных: группирование данных и применение к этим группам таких операций, как среднее, сумма, максимум, минимум, число элементов и т. п.

Ориентированный на работу с таблицами SQL не имеет достаточных средств для создания сложных прикладных программ. Поэтому в разных СУБД он либо используется вместе с языками программирования высокого уровня (например, такими как Си, Паскаль, PHP), либо включен в состав команд

специально разработанного языка СУБД (диалекта SQL): SQLpl в DB2, PL/SQL в Oracle, PL/pgSQL в PostgreSQL, Trunsact-SQL в SQL Server и т. п. Эти диалекты включают кроме SQL средства условной обработки (например, под контролем `IF…THEN`), управляющие операторы (например, циклы `WHILE`), переменные и обработку ошибок.

4.3. Таблицы SQL

До сих пор понятие "таблица", как правило, связывалось с реальной или базовой таблицей, т. е. с таблицей, для каждой строки которой в действительности имеется некоторый двойник, хранящийся в физической памяти машины (рис. 4.1). Однако SQL использует и создает ряд виртуальных (как будто существующих) таблиц: представлений, курсоров и неименованных рабочих таблиц, в которых формируются результаты запросов на получение данных из базовых таблиц и, возможно, представлений. Это таблицы, которые не существуют в базе данных, но как бы существуют с точки зрения пользователя.

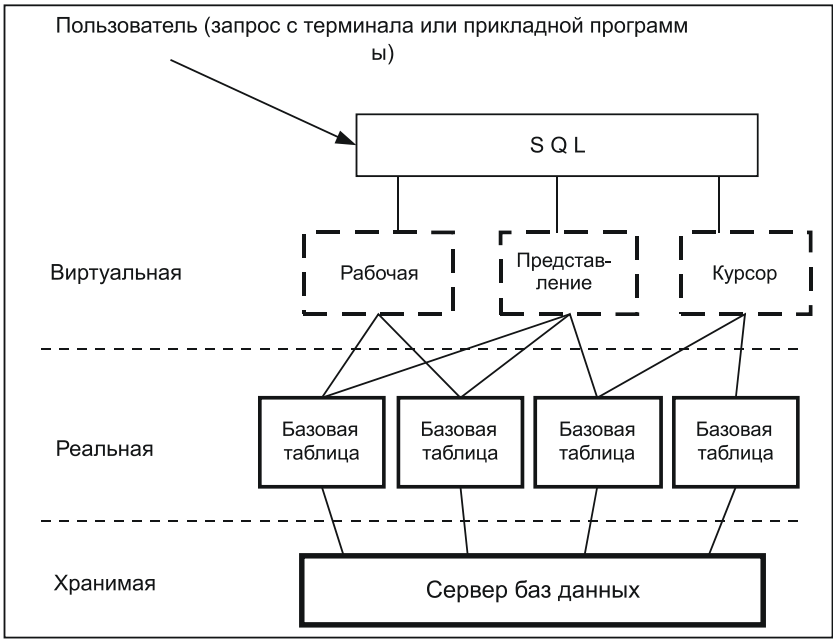


Рис. 4.1. База данных в восприятии пользователя

Базовые таблицы создаются с помощью предложения CREATE TABLE (создать таблицу), подробное описание которого приведено в *главе 13*. Здесь же приведем пример предложения для создания описания таблицы Блюда:

```
CREATE TABLE Блюда
     (
       Код_блюда  число,
       Блюдо      текст(70),
       Код_вида   число,
       Основа     текст(10),
       Выход      число,
       Труд       число
     );
```

Предложение CREATE TABLE специфицирует имя базовой таблицы, которая должна быть создана, имена ее столбцов и типы данных для этих столбцов (а также, возможно, некоторую дополнительную информацию, не иллюстрируемую данным примером). CREATE TABLE — выполняемое предложение. Если ввести его с терминала, система тотчас построит таблицу Блюда, которая сначала будет пустой: она будет содержать только строку заголовков столбцов, но не будет еще содержать никаких строк с данными. Однако можно немедленно приступить к вставке таких строк данных, возможно, с помощью предложения INSERT, и создать таблицу, аналогичную таблице Блюда (см. табл. 3.1).

Если теперь потребовалось узнать, какие овощные блюда может приготовить повар пансионата, то можно набрать на терминале следующий текст запроса:

```
SELECT Код_блюда, Блюдо
FROM   Блюда
WHERE  Основа = 'Овощи';
```

и мгновенно получить на экране следующий результат его реализации:

```
КОД_БЛЮДА БЛЮДО
--------- ----------------
        1 Салат летний
        3 Салат витаминный
       17 Морковь с рисом
       23 Помидоры с луком
```

Для выполнения этого предложения SELECT (выбрать), подробное описание которого будет дано в *главах 5 и 6*, СУБД должна сначала сформировать пустую рабочую таблицу, состоящую из столбцов Код_блюда и Блюдо, тип данных которых должен совпадать с типом данных аналогичных столбцов базовой таблицы Блюда. Затем она должна выбрать из таблицы Блюда все строки,

у которых в столбце Основа хранится слово Овощи, выделить из этих строк столбцы Код_блюда и Блюдо и загрузить укороченные строки в рабочую таблицу. Наконец, СУБД должна выполнить процедуры по организации вывода содержимого рабочей таблицы на экран терминала (при этом, если в рабочей таблице содержится более 20—24 строк, она должна использовать процедуры постраничного вывода и т. п.). После выполнения запроса СУБД должна уничтожить рабочую таблицу.

Если, например, надо получить значение калорийности всех овощей, включенных в таблицу Продукты, то можно набрать на терминале запрос

```
SELECT Продукт, Белки, Жиры, Углев, ((Белки + Углев)*4.1+Жиры*9.3)
FROM   Продукты
WHERE  Продукт IN ('Морковь', 'Лук', 'Помидоры', 'Зелень');
```

и получить на экране следующий результат его реализации:

```
ПРОДУКТ   БЕЛКИ   ЖИРЫ  УГЛЕВ ((БЕЛКИ+УГЛЕВ)*4.1+ЖИРЫ*9.3)
-------- ------ ------ ------ ----------------------------
Морковь    13,0    1,0   70,0                        349,6
Лук        17,0    0,0   95,0                        459,2
Помидоры    6,0    0,0   42,0                        196,8
Зелень      9,0    0,0   20,0                        118,9
```

В последнем столбце этой рабочей таблицы приведены данные о калорийности продуктов, отсутствующие в явном виде в базовой таблице Продукты. Эти данные вычислены по хранимым значениям основных питательных веществ продуктов, помещены в рабочую таблицу и будут существовать до момента смены изображения на экране. Однако если необходимо сохранить эти данные в какой-либо базовой таблице, то существует предложение (INSERT), позволяющее переписать содержимое рабочей таблицы в указанные столбцы базовой таблицы (реляционная операция присваивания).

Отметим, что в рабочей таблице в качестве имени последнего столбца выводится выражение для вычисления калорийности, т. е. тот текст, который приведен в запросе (другого нет). Однако можно присвоить этому тексту псевдоним, и тогда запрос и его результат будут иметь вид:

```
SELECT Продукт, Белки, Жиры, Углев, ((Белки + Углев)*4.1+Жиры*9.3) Калорийность
FROM   Продукты
WHERE  Продукт IN ('Морковь', 'Лук', 'Помидоры', 'Зелень');

ПРОДУКТ   БЕЛКИ   ЖИРЫ  УГЛЕВ КАЛОРИЙНОСТЬ
-------- ------ ------ ------ ------------
Морковь    13,0    1,0   70,0        349,6
```

```
Лук          17,0     0,0    95,0         459,2
Помидоры      6,0     0,0    42,0         196,8
Зелень        9,0     0,0    20,0         118,9
```

Часто пользователя не устраивает как способ описания нужного набора выводимых строк, так и результат выполнения запроса, сформированного из данных одной таблицы. Ему хотелось бы уточнить выводимые (запрашиваемые) данные сведениями из других таблиц.

Например, в запросе на получение состава овощных блюд

```
SELECT Код_блюда, Код_продукта, Вес
FROM   Состав
WHERE  Код_блюда IN (1, 3, 17, 23);
```

пришлось перечислять номера этих блюд, так как в таблице `Состав` нет данных об основных продуктах блюда (они есть в таблице `Блюда`). Полученный состав овощных блюд (левая колонка табл. 4.1) оказался "слепым": в нем и блюда и продукты представлены номерами, а не именами.

Удобнее и нагляднее выглядит результат запроса (правая колонка табл. 4.1), сформированного по трем таблицам:

```
SELECT Блюда.Блюдо, Продукты.Продукт, Состав.Вес
FROM   Состав, Блюда, Продукты
WHERE  Состав.Код_блюда = Блюда.Код_блюда
AND    Состав.Код_продукта = Продукты.Код_продукта
AND    Основа = 'Овощи';
```

В нем для получения рабочей таблицы выполняется естественное соединение (см. *разд. 3.3.2*) таблиц `Блюда`, `Продукты` и `Состав` (условие соединения — равенство значений кодов блюд и значений кодов продуктов). Затем выделяются строки, у которых в столбце `Основа` хранится слово `Овощи`, и из полученных строк — столбцы `Блюдо`, `Продукт` и `Вес`.

Таблица 4.1. *Результат запроса по составу овощных блюд базы данных "COOK"*

Запрос по одной таблице			Запрос по трем таблицам		
КОД_БЛЮДА	КОД_ПРОДУКТА	ВЕС	БЛЮДО	ПРОДУКТ	ВЕС
1	4	15	Салат летний	Майонез	15
1	11	100	Салат летний	Помидоры	100
1	12	5	Салат летний	Зелень	5
1	15	80	Салат летний	Яблоки	80
3	6	50	Салат витаминный	Сметана	50

***Таблица 4.1** (окончание)*

Запрос по одной таблице			Запрос по трем таблицам	
3	10	15	Салат витаминный Лук	15
3	12	20	Салат витаминный Помидоры	55
3	11	55	Салат витаминный Зелень	20
3	16	5	Салат витаминный Яблоки	55
3	15	55	Салат витаминный Сахар	5
17	3	20	Морковь с рисом Масло	20
17	7	50	Морковь с рисом Молоко	50
17	12	10	Морковь с рисом Морковь	150
17	9	150	Морковь с рисом Зелень	10
17	13	25	Морковь с рисом Рис	25
17	14	5	Морковь с рисом Мука	5
23	3	20	Помидоры с луком Масло	20
23	10	65	Помидоры с луком Лук	65
23	11	250	Помидоры с луком Помидоры	250

Если пользователи достаточно часто интересуются составом различных блюд, то для упрощения формирования запросов целесообразно создать представление.

Представление — это пустая именованная таблица, определяемая перечнем тех столбцов таблиц и признаками тех их строк, которые хотелось бы в ней увидеть. Представление является как бы "окном" в одну или несколько базовых таблиц. Оно создается с помощью предложения `CREATE VIEW` (создать представление), подробное описание которого приведено в *разд. 7.3.4*. Здесь же приведем пример предложения для создания представления `Состав_блюд`:

```
CREATE VIEW Состав_блюд
AS
SELECT   Блюдо, Продукт, Вес
FROM     Состав, Блюда, Продукты
WHERE    Состав.Код_блюда = Блюда.Код_блюда
AND      Состав.Код_продукта = Продукты.Код_продукта;
```

Оно описывает пустую таблицу, в которую при реализации запроса будут загружаться данные из столбцов `Блюдо`, `Продукт` и `Вес` таблиц `Блюда`, `Продукты` и `Состав` соответственно. Теперь для получения состава овощных блюд можно дать запрос

```
SELECT Блюдо, Продукт, Вес
FROM   Состав_блюд
WHERE  Основа = 'Овощи';
```

и получить на экране терминала данные, которые представлены в правой колонке табл. 4.1. А для получения состава супа харчо можно дать запрос:

```
SELECT Блюдо, Продукт, Вес
FROM   Состав_блюд
WHERE  Блюдо = 'Суп харчо';
```

О целесообразности создания представлений будет рассказано далее, а здесь лишь отметим, что они позволяют повысить уровень логической независимости данных, упростить их восприятие и "скрыть" от некоторых пользователей те или иные данные. Например, данные о новых ценах на продукты первой необходимости или из какой рыбы приготавливается "Судак по-польски" (тут мы вспомнили меню одной из столовых, в котором предлагалось блюдо "Судак по-польски из хека").

Наконец, еще об одних виртуальных таблицах — курсорах. *Курсор* — это пустая именованная таблица, определяемая перечнем тех столбцов базовых таблиц и признаками тех их строк, которые хотелось бы в ней увидеть. В чем же различие между курсором и представлением?

Для пользователя представления почти не отличаются от базовых таблиц (есть лишь некоторые ограничения при выполнении различных операций манипулирования данными). Они могут использоваться как в интерактивном режиме, так и в прикладных программах. Курсоры же созданы для процедурной работы с таблицей в прикладных программах. Например, после объявления курсора

```
DECLARE
Блюд_состав CURSOR
FOR
SELECT Блюдо, Продукт, Вес
FROM    Состав, Блюда, Продукты
WHERE   Состав.Код_блюда = Блюда.Код_блюда
AND     Состав.Код_продукта = Продукты.Код_продукта
AND     Блюдо = 'Суп харчо';
```

и его активизации (`OPEN Блюд_состав`) будет создана временная таблица с составом блюда "Суп харчо" и специальным указателем, определяющим в качестве текущей первую строку этой таблицы. С помощью предложения `FETCH` (выбрать), которое обычно исполняется в программном цикле, можно присвоить определенным переменным значения указанных столбцов этой строки. Одновременно курсор будет передвинут к следующей строке таблицы. После обработки в программе полученных значений переменных выполняется следующее предложение `FETCH` и т. д. до окончания перебора всех продуктов харчо.

4.4. Синтаксические конструкции SQL

Все составляющие языка SQL можно условно разделить на следующие конструкции:

- предложения;
- идентификаторы (имена);
- константы;
- операторы;
- зарезервированные и ключевые слова;
- псевдостолбцы и таблица DUAL.

В книге будут описываться и иллюстрироваться на примерах конструкции стандарта SQL:2003, а также его реализации в Oracle Database 10g — ведущей СУБД в коммерческом секторе, работающей в разнообразных операционных системах (Linux, Solaris, Unix, Windows и пр.) и являющейся основой множества информационных систем.

> Отметим, что Oracle предлагает разработчикам бесплатную версию СУБД — Oracle® Database 10g Express Edition. Эту версию (Oracle Database XE) для Linux или Windows читатель может загрузить с Web-сайта корпорации Oracle и установить на свой персональный компьютер (на компакт-диске, прилагаемом к книге, размещена версия **Oracle Database 10g Release 2 (10.2.0.1) Express Edition for Microsoft Windows**). Описание содержимого компакт-диска размещено в *приложении Б*.
>
> Установив Oracle и использовав другие материалы диска, можно реализовать все примеры и упражнения для подробного изучения разделов книги.

4.4.1. Предложения SQL

В литературе по базам данных нет однозначного названия основной конструкции языка. В англоязычной литературе она именуется как *statement*, а в русскоязычной превращается в: оператор, инструкция, команда, утверждение, предложение. В этой книге мы будем использовать понятие *предложение*, которое может состоять из фраз, включающих те или иные конструкции SQL.

Любое предложение SQL состоит из ключевых и зарезервированных слов и слов, определяемых пользователем в соответствии с синтаксическими пра-

вилами. Для записи предложений принят свободный формат, но мы рекомендуем придерживаться следующих правил:

- каждая фраза предложения должна начинаться с новой строки;
- начало фразы должно быть выровнено с началом остальных фраз предложения;
- части фразы, не помещающиеся в строку, должны начинаться с новой строки с некоторым отступом относительно начала всей фразы;
- для записи ключевых слов используйте прописные буквы;
- для записи определяемых пользователем слов используйте строчные буквы (кроме записи текстовых констант см. *разд. 4.4.3*);
- каждое предложение SQL должно заканчиваться символом точка с запятой (**;**).

В синтаксисе предложений используются условные обозначения, показанные в табл. 4.2.

Таблица 4.2. *Условные обозначения*

Обозначение	Описание
`[ ]`	В эти скобки заключаются необязательные синтаксические единицы
`{ }`	Конструкция, заключенная в эти скобки, должна рассматриваться как одна синтаксическая единица
`|`	Используется для разделения альтернативных синтаксических единиц
`...`	Указывает на то, что непосредственно предшествующая синтаксическая единица может повторяться один или несколько раз
*	Используется для замены перечня имен всех столбцов таблицы

Все предложения SQL делятся на три группы:

- предложения манипуляции данными (Data Manipulation Language, DML)
- предложения определения данных (Data Definition Language, DDL)
- предложения определения доступа к данным (Data Control Language, DCL)

Начнем изучение с предложений манипуляции данными: `SELECT` (выбрать), `INSERT` (вставить), `UPDANT` (обновить) и `DELETE` (удалить).

4.4.2. Идентификаторы (имена)

Идентификаторы — это пользовательские или системные имена объектов баз данных. Они должны или могут присваивается базам данных, таблицам, ограничениям и правилам в таблице, столбцам таблицы, индексам, курсорам, представлениям, процедурам, триггерам и т. п.

Идентификатор объекта создается при определении объекта. Например, так, как это было проиллюстрировано в *разд. 4.3*, для создания имен таблицы и ее столбцов:

```
CREATE TABLE Блюда
     (
       Код_блюда  число,
       Блюдо      текст(70),
       Код_вида   число,
       Основа     текст(10),
       Выход      число,
       Труд       число
     );
```

Затем идентификатор используется для обращения к объектам. Например:

```
SELECT Код_блюда, Блюдо
FROM   Блюда;
```

Существует два класса идентификаторов:

- обычные идентификаторы;
- идентификаторы с разделителем.

В SQL:2003 и обычные идентификаторы, и идентификаторы с разделителем могут содержать от 1 до 128 символов, а в СУБД Oracle — до 30 байт (число символов зависит от их набора).

Обычный идентификатор может содержать любые буквы, цифры и знак подчеркивания (_), а в Oracle еще и знаки диеза (#) и доллара ($). Он должен начинаться с буквы и не может содержать пробелов и специальных символов. Он не может быть зарезервированным словом (см. *разд. 4.4.5*), например, `SELECT`.

В обход некоторых из этих ограничений можно использовать идентификаторы с разделителем, т. е. имена объектов, заключенные в двойные кавычки ("). В частности, такие идентификаторы можно применять для того, чтобы давать имена с зарезервированными словами, или для того, чтобы использовать в имени обычно не употребляемые там символы (например, "Код блюда").

4.4.3. Константы и *NULL*-значения

В SQL *константами* считаются любые числовые значения, строки символов, значения, связанные с представлением времени (дата и время), и булевы значения, которые не являются идентификаторами или ключевыми словами.

Числовые константы могут выглядеть так:

```
546
-777
+36.6
5E6
-9E-2
```

В них допустимы числа со знаком и без знака, в обычной и экспоненциальной записи. В числовых константах разрешается использовать следующий набор символов:

```
0 1 2 3 4 5 6 7 8 9 + - $ . E e
```

Не следует включать в числовую константу запятую, которая обычно используется как разделитель элементов (например, константа 7,654 будет интерпретироваться как 7 и отдельно 654).

Булевы значения, строковые константы и даты выглядят примерно так:

```
TRUE
'С новым годом'
15-10-2007 15:06:54
15-ОКТЯБРЬ-2007 15:06:54
```

Строковые константы должны всегда заключаться в одинарные кавычки (`' '`) — апострофы. Символы в строковых константах не ограничиваются алфавитными символами. По сути, любой символ из набора символов можно представить в виде строковой константы:

```
'1989'
'57.321 + 12345'
'Это тоже строковая константа'
'15-ОКТЯБРЬ-2007 15:06:54'
```

Все приведенные примеры фактически являются совместимыми с типом данных `CHARACTER` (см. *разд. 4.5*). Не путайте строковую константу `'1989'` с числовой константой `1989`. Не стоит использовать строковые константы в арифметических выражениях без явного их преобразования в числовой тип. Хотя в Oracle некоторые из них и преобразуются автоматически. Например, при выполнении предложения

```
SELECT '1989' + 1234.56 FROM DUAL;
```

будет получен результат суммирования 1989 с 1234.56:

```
'1989'+1234.56
--------------
       3223,56
```

а при выполнении предложения

```
SELECT '1989' + '1234.56' FROM DUAL;
```

появится сообщение об ошибке:

```
ORA-01722: неверное число
```

При необходимости отражения в строковой константе символа одинарной кавычки, необходимо написать этот символ два раза. Например, для вывода текста:

```
- 'К нам придет Д'Артаньян ?'
```

придется написать запрос:

```
SELECT '- '' К нам придет Д''Артаньян ? ''' FROM DUAL;
```

Заметим, что все СУБД используют специальное значение — NULL (пустое или несуществующее значение), которое может быть записано в поле таблицы базы данных. NULL-значения нужно всегда рассматривать как "отсутствие информации", а не как пустые строки, пробелы или нули.

Существует несколько кратких правил о поведении NULL-значений.

- Значения типа NULL нельзя помещать в столбцы, определенные как NOT NULL.
- Значения типа NULL не равны друг другу. Распространенная ошибка сравнивать два столбца, содержащие значения NULL, и ожидать, что они совпадут. (Правильный метод идентификации значений NULL, в предложениях WHERE или в булевых выражениях — это использование таких выражений, как `value IS NULL` и `value IS NOT NULL`.)
- Столбец, содержащий значение NULL, игнорируется при вычислениях агрегатных значений, таких, как AVG, SUM или COUNT.
- Если столбцы, содержащие значения NULL, перечислены в предложении GROUP BY запроса, выходные данные запроса будут содержать для значений NULL всего одну строку. По сути дела стандарт рассматривает все найденные значения NULL как одну группу.
- В предложениях DISTINCT или ORDER BY, как и в предложении GROUP BY, значения NULL не отличаются друг от друга. Для предложения ORDER BY Oracle устанавливает значения NULL в конец получаемого набора данных.

4.4.4. Операторы

Оператор — это символ или имя, обозначающий действие, выполняемое над одним или несколькими выражениями. Они, как правило, делятся на следующие категории: арифметические операторы, оператор конкатенации, операторы присваивания, операторы сравнения, логические операторы и унарные операторы.

Арифметические операторы выполняют математические действия над двумя значениями любого типа, относящегося к числовой категории (табл. 4.3).

Таблица 4.3. *Арифметические операторы*

Оператор	Описание
+	Сложение
–	Вычитание
*	Умножение
/	Деление

Операторы + и – могут применяться для выполнения арифметических операций над датами.

Оператор конкатенации (||) соединяет две отдельные текстовые строки в одно строковое значение.

Оператор присваивания (=) присваивает значение переменной или псевдониму заголовка столбца. В Oracle используется оператор присваивания :=.

Операторы сравнения проверяют равенство или неравенство двух выражений (табл. 4.4). Результатом операции сравнения является булево значение: TRUE (верно), FALSE (неверно) или UNKNOWN (неизвестно). Если хотя бы одно из сравниваемых значений равно NULL, то результат также будет NULL.

Таблица 4.4. *Операторы сравнения*

Оператор	Описание
=	Равно
>	Больше
<	Меньше

Таблица 4.4 *(окончание)*

Оператор	Описание
>=	Больше или равно
<=	Меньше или равно
<>	Не равно
IS NULL	Имеет ли значение NULL
IS NOT NULL	Не имеет ли значение NULL

Эти операторы наиболее часто используются во фразе WHERE для отбора строк, соответствующих условиям поиска (см. примеры в *разд. 4.3*).

Логические операторы обычно применяются во фразе WHERE для проверки истинности какого-либо условия (табл. 4.5). Логические операторы возвращают булево значение TRUE или FALSE.

Таблица 4.5. *Логические операторы*

Оператор	Описание
ALL	TRUE, если весь набор сравнений даст результат TRUE
AND	TRUE, если оба булевых выражения дают результат TRUE
ANY	TRUE, если хотя бы одно сравнение из набора даст результат TRUE
BETWEEN	TRUE, если операнд находится внутри диапазона
EXISTS	TRUE, если подзапрос возвращает хотя бы одну строку
IN	TRUE, если операнд равен одному выражению из списка или одной или нескольким строкам, возвращаемым подзапросом
LIKE	TRUE, если операнд совпадает с шаблоном
NOT	Обращает значение любого другого булевого оператора
OR	TRUE, если любое булево выражение равно TRUE
SOME	TRUE, если несколько сравнений из набора дают результат TRUE

Унарные операторы выполняют операцию над одним выражением числовой категории: + (числовое значение становится положительным) и – (числовое значение становится отрицательным).

Приоритет операторов

Когда в выражении присутствуют несколько операторов, последовательность их выполнения определяет приоритет операторов. Далее приведен список операторов, в котором они расположены в порядке от самого высокого к самому низкому приоритету:

1. () (выражения, стоящие в скобках).
2. +, – (унарные операторы).
3. *, / (математические операторы).
4. +, – (арифметические операторы).
5. =, >, <, >=, <=, <> (операторы сравнения).
6. IS NULL.
7. NOT.
8. AND.
9. ALL, ANY, BETWEEN, IN, LIKE, OR, SOME.
10. = (присваивание значения переменной).

Если операторы имеют одинаковый приоритет, вычисления производятся слева направо. Для того чтобы изменить применяемый по умолчанию порядок выполнения операторов, в выражении используются скобки. Выражения, заключенные в скобки, вычисляются первыми, а уже после них — все, что находится за скобками. Если применяются вложенные скобки, то первым выполняются выражения в наиболее глубоко вложенных скобках.

4.4.5. Зарезервированные и ключевые слова

В SQL существуют некоторые слова и фразы, имеющие особую значимость. *Ключевые слова SQL* — это слова, которые настолько тесно связаны с функционированием реляционной базы данных, что их нельзя использовать ни для каких других целей.

В SQL:2003 определяется свой список зарезервированных и ключевых слов. Кроме этого, существующие СУБД имеют собственные списки подобных

слов (в том числе и Oracle). Эти слова нельзя использовать в качестве идентификаторов (символом * отмечены слова стандарта SQL:2003):

ACCESS	DEFAULT *	INTEGER *	ONLINE	START
ADD *	DELETE *	INTERSECT *	OPTION *	SUCCESSFUL
ALL *	DESC *	INTO *	OR *	SYNONYM
ALTER *	DISTINCT *	IS *	ORDER *	SYSDATE
AND *	DROP *	LEVEL *	PCTFREE	TABLE *
ANY *	ELSE *	LIKE *	PRIOR *	THEN *
AS *	EXCLUSIVE	LOCK	PRIVILEGES *	TO *
ASC *	EXISTS	LONG	PUBLIC *	TRIGGER
AUDIT	FILE	MAXEXTENTS	RAW	UID
BETWEEN *	FLOAT *	MINUS	RENAME	UNION *
BY *	FOR *	MLSLABEL	RESOURCE	UNIQUE *
CHAR *	FROM *	MODE	REVOKE *	UPDATE *
CHECK *	GRANT *	MODIFY	ROW	USER *
CLUSTER	GROUP *	NOAUDIT	ROWID	VALIDATE
COLUMN	HAVING *	NOCOMPRESS	ROWNUM	VALUES *
COMMENT	IDENTIFIED	NOT *	ROWS *	VARCHAR *
COMPRESS	IMMEDIATE *	NOWAIT	SELECT *	VARCHAR2
CONNECT *	IN *	NULL *	SESSION *	VIEW *
CREATE *	INCREMENT	NUMBER	SET *	WHENEVER *
CURRENT *	INDEX	OF *	SHARE	WHERE
DATE *	INITIAL	OFFLINE	SIZE *	WITH *
DECIMAL *	INSERT *	ON *	SMALLINT *	

4.4.6. Псевдостолбцы, таблица *DUAL* и еще о словах, которые нежелательно использовать пользователям

Кроме зарезервированных слов в Oracle существуют и другие, которые имеют специальное значение. Это имена типов данных (см. *разд. 4.5*) и имена встроенных функций (см. *разд. 4.6*). Не следует использовать имена схем,

ключевые слова DIMENSION, SEGMENT, ALLOCATE, DISABLE, имена псевдостолбцов и таблицы DUAL (табл. 4.6), а также слова, начинающиеся с SYS_ .

Таблица 4.6. *Псевдостолбцы и таблица* DUAL

Имя	Значение
Имя_последовательности.CURRVAL	Текущее значение последовательности для текущего сеанса (Имя_послед.NEXTVAL) должно быть объявлено первым
Имя_последовательности.NEXTVAL	Следующее значение последовательности для текущего сеанса
Имя_таблицы.LEVEL	1 — для корня, 2 — для дочернего уровня корня и т. д. Используется в команде SELECT...CONNECT BY, которая реализует иерархические структуры
ROWID	Уникальный идентификатор строки таблицы. Для этого идентификатора существует специальный тип данных ROWID
ROWNUM	Позиция отдельной строки среди строк, отобранных запросом. Oracle выбирает строки в произвольном порядке и оценивает ROWNUM перед сортировкой с помощью фразы ORDER BY. Однако если ORDER BY использует индексы, то порядок ROWNUM может отличаться от его порядка без индекса
DUAL	Таблица, автоматически создаваемая Oracle для каждого пользователя. В ней один столбец с именем DUMMY и типом данных VARCHAR 2(1). В единственной строке этой таблицы хранится значение 'X'. Отметим, что указанное значение и его описание не имеет большого значения, так как чаще всего эта таблица используется для вывода значения какого-либо выражения любого типа. Например, для получения текущей даты и времени можно дать запрос: SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS') FROM DUAL; который, при выполнении запроса в 18 часов 15 минут 42 секунды 27 января 2008 года, выдаст результат: 27-янв-2008 18:15:42. Можно также дать запрос SELECT SIN(3) FROM DUAL; получив 0.14112000805986722210074480280811027987 и т. п.

4.5. Типы данных SQL

Типы данных позволяют установить основные правила для данных, содержащихся в конкретном столбце таблицы, в том числе размер выделяемой для них памяти.

В языке SQL имеется несколько категорий типов данных, основные из которых приведены в табл. 4.7.

Таблица 4.7. *Категории и типы данных SQL:2003*

Категория	Пример	Описание
Символьные	`CHAR`, `VARCHAR`	Любые комбинации символов из допустимого набора. Вариабельные типы позволяют хранить строки переменной длины, тогда как остальные — только строки фиксированной. В вариабельных типах данных автоматически удаляются замыкающие пробелы, а в других типах пробелы остаются
	`NATIONAL CHARACTER`	Типы данных, связанные с национальными символами
Двоичные	`BLOB`, `CLOB`	Двоичные (символьные) большие объекты
Числовые	`NUMERIC(p, s)`	Числовые значения с регулируемой точностью (`p`) и масштабом (`s`)
Дата/время	`DATE`	Данные, связанные со временем
Связь с данными	`DATALINK`	Определяет ссылку на файл или другой внешний источник, не являющийся частью среды SQL
Интервальные	`INTERVAL`	Определяет набор временных значений или промежуток времени
XML	`XML`	Данные формата XML

СУБД Oracle поддерживает множество типов данных, в том числе и большинство стандартных типов SQL:2003.

4.5.1. Символьные

Символьные данные могут иметь фиксированную и переменную длину (табл. 4.8). Эти данные обозначаются в SQL с помощью одинарных кавычек, о чем подробно рассказывалось в *разд. 4.4.3*. Данные с символами в формате UNICODE (стандарт кодирования символов, позволяющий представить знаки практически всех письменных языков), как правило, содержат большее число байт, чем данные в стандарте ASCII (American Standard Code for Information Interchange), и это надо учитывать, устанавливая длину (n) в байтах.

Таблица 4.8. *Символьные данные в Oracle*

Oracle	SQL:2003	Описание
`CHAR(n) [BYTE | CHAR],` `CHARACTER(n) [BYTE | CHAR]`	`CHARACTER(n)`	Хранит символьные данные фиксированной длины до 2000 байт. При указании атрибута `BYTE` длина массива измеряется в байтах. При указании атрибута `CHAR` длина измеряется в символах

`NCHAR(n),`
`NATIONAL CHARACTER(n),`
`NATIONAL CHAR{n}`
	`NATIONAL CHARACTER`	Хранит данные в формате символов UNICODE длиной от 1 до 2000 байт. По умолчанию — 1 байт

`VARCHAR(n),`
`CHARACTER VARYING(n),`
`CHAR VARYING(n)`
	`CHARACTER VARYING(n)`	Хранит символьные данные размером от 1 до 4000 байт. Oracle рекомендует использовать вместо этого типа тип `VARCHAR2`

`VARCHAR2(n [BYTE | CHAR])`	`CHARACTER VARYING(n)`	Хранит символьные данные переменной длины до 4000 байт (определяется параметром `n`). *Атрибут* `BYTE` показывает, что размер измеряется в байтах. Если вы используете атрибут `CHAR`, база Oracle должна провести внутреннее преобразование в определенное количество байт, которое должно соответствовать ограничению в 4000 байт

Таблица 4.8 *(окончание)*

Oracle	SQL:2003	Описание
NVARCHAR2(n)	Отсутствует	Представляет собой рекомендуемый Oracle тип для хранения символов UNICODE переменной длины от 1 до 4000 байт

4.5.2. Двоичные

LOB-типы используются для хранения больших объектов (Large Object). Двоичные объекты BLOB используются для хранения очень больших объектов данных с неопределенным или переменным размером, таких как графика, векторная графика, звуковые файлы, фотографии, видеосегменты и другие виды мультимедийной информации. Символьные объекты CLOB используются для хранения глав книг, больших документов и т. п. В табл. 4.9 приведено описание двоичный типов данных.

Таблица 4.9. *Двоичные данные в Oracle*

Oracle	SQL:2003	Описание
BLOB	BLOB	Хранит большой двоичный объект (Binary Large Object, BLOB) размером от 8 до 128 терабайт в зависимости от размера блока в базе данных
CLOB	CLOB	Хранит большой символьный объект (Large Character Object, CLOB) размером от 8 до 128 терабайт в зависимости от размера блока в базе данных
NCLOB	NCLOB	Представляет собой CLOB с поддержкой многобайтовых символов (например, UNICODE) размером от 8 до 128 терабайт в зависимости от размера блока базы данных

В Oracle большие двоичные объекты (BLOB, CLOB и NCLOB) имеют следующие ограничения:

- их нельзя выбирать с удаленной машины;
- их нельзя сохранять в кластерах;
- они не могут быть компонентом предложений `ORDER BY` и `GROUP BY` в запросе;

- их нельзя использовать в агрегатных функциях запроса;
- на них нельзя ссылаться в запросах при помощи инструкций `DISTINCT` и `UNIQUE` или в соединениях;
- они не могут быть частью первичного ключа или ключа индекса;
- их нельзя использовать в предложении `UPDATE OF` триггера `UPDATE`.

4.5.3. Числовые

Используются для хранения нулевых, положительных и отрицательных чисел с фиксированной и плавающей запятой (точкой). Из всех, перечисленных в табл. 4.10, типов *на практике можно использовать только один* — `NUMBER(p, s)`. Остальные типы числовых данных существуют в стандарте и сохранены для поддержания ранее созданных программ.

Таблица 4.10. *Числовые данные в Oracle*

Oracle	SQL:2003	Описание
`BINARY_FLOAT`	`FLOAT`	Хранит 32-битное число с плавающей точкой
`BINARY_DOUBLE`	`FLOAT`	Хранит 64-битное число с плавающей точкой
`DECIMAL(p, s)`	`DECIMAL(p, s)`	Синоним типа `NUMBER`, принимающий в качестве аргументов точность и масштаб
`DOUBLE PRECISION`	`DOUBLE PRECISION`	Хранит значения с плавающей точкой двойной точности. То же, что `FLOAT(126)`
`FLOAT(n)`	`FLOAT(n)`	Хранит числовые данные с плавающей точкой с двоичной точностью до 126
`INTEGER(n)`	`INTEGER`	Хранит целые числа со знаком и без знака с точностью до 38. Тип `INTEGER` считается синонимом `NUMBER`
`NUMBER(p, s)`, `NUMERIC(p, s)`	`NUMERIC(p, s)`	Хранит числа с точностью в пределах от 1 до 38 и масштабом от –84 до 127

Таблица 4.10 *(окончание)*

Oracle	SQL:2003	Описание
REAL	REAL	Хранит значения с плавающей точкой с одинарной точностью. То же, что FLOAT(63)
SMALLINT	SMALLINT	То же, что INTEGER

4.5.4. Дата/время

В Oracle тип DATE хранит дату и время, т. е. заменяет стандартные типы DATE и TIMESTAMP, хотя и сохранил форматы стандарта (табл. 4.11). Для получения всех необходимых видов дата/время в Oracle используются функции TO_CHAR и (или) TO_DATE (см. *разд. 4.6*) и соответствующий формат даты (см. *разд. 4.5*).

Таблица 4.11. *Дата/время в Oracle*

Oracle	SQL:2003	Описание
DATE	DATE	Хранит дату и время от 00:00:00 01-01-4712 до н. э. до 23:59:59 31-12-9999
TIMESTAMP(n){[WITH TIME ZONE] | [WITH LOCAL TIME ZONE]}	TIMESTAMP[WITH TIME ZONE]	Значение полной даты и времени, где n — количество цифр для долей секунды в поле секунд (допустимые значения 0—9, по умолчанию — 6). При указании атрибута WITH TIME ZONE сохраняется переданный в качестве параметра часовой пояс (по умолчанию — часовой пояс текущего сеанса) и значение времени выдается с учетом этого часового пояса. При указании атрибута WITH LOCAL TIME ZONE данные хранятся с учетом часового пояса текущего сеанса и возвращаются также с учетом часового пояса текущего сеанса

4.5.5. Связь с данными

Для связи с данными в Oracle используется тип BFILE (в SQL:2003 DATALINK), который содержит указатель на объект типа BLOB, хранимый вне пределов базы данных, но находящийся на локальном сервере и имеющий размер до 4 Гбайт. База данных осуществляет потоковый доступ по чтению (но не по записи) к этому внешнему объекту. Если вы удалите строку, содержащую значение типа BFILE, будет удален только указатель (исходная структура файлов не затрагивается).

4.5.6. Интервальные

Используются для описания промежутков времени между двумя временными отсчетами, задаваемыми типом "Дата/время" (табл. 4.12).

Таблица 4.12. *Интервальные данные в Oracle*

Oracle	SQL:2003	Описание
INTERVAL DAY(n) TO SECOND (x)	INTERVAL	Хранит промежуток времени, измеряемый в днях, часах, минутах и секундах, n — число цифр в поле "день" (допустимые значения 0—9, по умолчанию — 2), x — число цифр для долей секунды в поле секунд (допустимые значения 0—9, по умолчанию — 6)
INTERVAL YEAR (n) TO MONTH (x)	INTERVAL	Хранит промежуток времени, измеряемый в годах и месяцах, где n — число цифр в поле года. Значение n может быть от 0 до 9, по умолчанию — 2

4.5.7. XML

Для хранения в базе Oracle данных формата XML используется XMLTYPE (в SQL:2003 XML). Доступ к данным XML осуществляется с помощью выражений XPath, а также нескольких встроенных XPath-функций, функций SQL и пакетов PL/SQL. Тип XMLTYPE определяется системой, поэтому его можно применять в качестве аргумента функций, а также типа данных для столбца в таблице или представлении. При использовании этого типа в таблице данные можно сохранить в форме CLOB или связанного объекта.

4.5.8. Данные, специфичные для СУБД Oracle

В Oracle есть специфичные данные, отсутствующие в стандарте SQL:2003, описание которых приведено в табл. 4.13.

Таблица 4.13. *Данные в Oracle*

Oracle	SQL:2003	Описание
`ROWID`	Нет	Представляет собой уникальный идентификатор типа base-64 для каждой строки таблицы. Часто используется с псевдостолбцом `ROWID`
`UROWID [(n)]`	Нет	Хранит значение типа base-64, показывающее логический адрес строки в таблице. По умолчанию размер составляет 4000 байт. При желании можно указать размер в пределах до 4000 байт

4.6. Функции SQL

Функции могут быть использованы везде, где используются переменные, столбцы или выражения (соответствующего типа). Их обычно подразделяют на числовые, символьные, агрегатные, функции работы с датами (дата и время) и т. п. (см. табл. 4.14—4.22).

В описаниях функций используются следующие параметры:

- `char`, `char1`, `char2`,... — константы в апострофах или выражения типа `CHAR`;
- `d`, `d1`, `d2`,... — константы в апострофах или выражения типа `DATE`;
- `expr`, `expr1`, `expr2`,... — любые выражения;
- `fmt` — формат данных;
- `k`, `m`, `n` — числовые константы или выражения типа `NUMBER`;
- `nls` — выражение вида `'NLS_SORT = name'`;
- `raw` — исходные данные;
- `rowid` — внутренний уникальный идентификатор строки;
- `set`, `set1`, `set2` — наборы символов;
- `z1`, `z2` — часовые пояса (см. табл. 4.16).

4.6.1. Числовые функции

Таблица 4.14. *Числовые функции*

Функции	Возвращаемое значение
ABS(n)	Абсолютное значение n, большее или равное n
ACOS(n)	Арккосинус n (n от –1 до 1), результат от 0 до π в радианах
ATAN(n)	Арктангенс n (n не ограничено), результат от –π/2 до π/2
ATAN2(n/m)	Арктангенс n и m (n и m не ограничены), результат от –π до π
CEIL(n)	Наименьшее целое, не меньшее n
COS(n)	Косинус n, заданного в радианах
COSH(n)	Гиперболический косинус n в радианах
EXP(n)	Возведение e (exp) в степень n (где e = 2.7182818)
FLOOR(n)	Наибольшее целое, меньшее или равное n
LN(n)	Натуральный логарифм n, где n > 0
LOG(m,n)	Основание m логарифма n
MOD(m,n)	Остаток от деления m на n
POWER(m,n)	m в степени n. Если n не целое, то оно усекается до целого
ROUND(n[,m])	n, округленное до m-го десятичного знака; если m опущено, то оно принимается равным 0. m может быть отрицательным для округления цифр левее десятичной точки
SIGN(n)	Если n < 0, то –1; если n = 0, то 0; если n > 0, то 1
SIN(n)	Синус n, заданного в радианах
SINH(n)	Гиперболический синус n в радианах
SQRT(n)	Квадратный корень из n; если n < 0, то NULL
TAN(n)	Тангенс n, заданного в радианах
TANH(n)	Гиперболический тангенс n в радианах
TRUNC(n[,m])	n, усеченное до m десятичных знаков; если m опущено, то оно принимается равным 0. m может быть отрицательным для усечения (обнуления) цифр слева от десятичной точки

4.6.2. Символьные функции

Таблица 4.15. *Символьные функции*

Функции	Возвращаемое значение
`ASCII(char)`	Код ASCII первого символа символьной переменной `char`
`CHR(n)`	Символ, код ASCII которого равен `n`
`CONCAT(char1, char2`	Соединяет (конкатенирует) строку `char1` со строкой `char2`. (Эквивалентна выражению: `char1 || char2`)
`INITCAP(char)`	Символьная переменная с первыми буквами слов, начинающихся с заглавной буквы
`INSTR(char1, char2[,n[,m]])`	Позиция `m`-го включения `char2` в `char1` при начале поиска с позиции `n`. Если `m` опущено, по умолчанию оно равно 1; аналогично для `n`. Позиции даются относительно первого знака `char1`, даже если `n` > 1
`INSTRB(char1, char2[,n[,m]])`	Эквивалентна `INSTR`, но `n` и результат возвращаются в байтах, а не в позициях символов. Эту функцию полезно использовать при работе с многобайтовыми символьными строками
`LENGTH(char)`	Длина в знаках символьной переменной `char`
`LENGTHB(char)`	Длина в байтах символьной переменной `char`
`LOWER(char)`	`char`, где все буквы преобразованы в строчные (маленькие)
`LPAD(char1,n [,char2])`	Строка `char1`, дополненная слева до длины `n` последовательностью символов из строки `char2` с повторением этой последовательности столько раз, сколько необходимо. Если `char2` опущено, то для заполнения используются пробелы
`LTRIM(char [,set])`	Удаляет из `char` начальные знаки до тех пор, пока не появится знак, отсутствующий среди знаков `set`. При отсутствии `set` из `char` удаляются все левые пробелы
`NLS_INITCAP(char [,nls])`	Аналог `INITCAP`, но необязательный аргумент `nls` позволяет задать используемый в функции национальный язык
`NLS_LOWER(char [,nls])`	Аналог `LOWER`, но необязательный аргумент `nls` позволяет задать используемый в функции национальный язык

Таблица 4.15 *(окончание)*

Функции	Возвращаемое значение
NLSSORT(char [,nls])	Байтовая строка, использованная для сортировки char на базе языка, заданного аргументом nls. Эту функцию полезно применять для сравнения строк в различных языках
NLS_UPPER(char [,nls])	Аналог UPPER, но необязательный аргумент nls позволяет задать используемый в функции национальный язык
REPLACE(char1, char2[,char3])	Строка, полученная из char1, в которой все вхождения char2 заменены на char3. Если char3 отсутствует, то все вхождения char2 в char1 удаляются
RPAD(char1,n [,char2])	Строка char1, дополненная справа символами char2, с повторением, если необходимо. Если char2 опущена, то char1 дополняется пробелами
RTRIM(char [,set])	Удаляет из char конечные знаки до тех пор, пока не появится знак, отсутствующий среди знаков set. При отсутствии set из char удаляются все правые пробелы
SOUNDEX(char)	Фонетическое представление char (четырехсимвольное представление, показывающее, как звучит начало char)
SUBSTR(char,m [,n])	Подстрока, получаемая из char, начиная с символа m. Если задано n, то подстрока ограничивается n символами. При отрицательном m символы отсчитываются с конца char
SUBSTRB(char,m [,n])	Эквивалентно SUBSTR, но аргументы m и n выражаются не в символах, а в байтах. Эту функцию полезно использовать при работе с многобайтовыми символьными строками
TRANSLATE(char, set1, set2)	Строка, полученная трансляцией char в наборе set1 в набор set2
TRIM([{{LEADING | TRAILING BOTH} char1} char1} FROM] char2)	Если задано LEADING, из char2 удаляются начальные char1. Если задано TRAILING, из char2 удаляются конечные char1. Если задано BOTH или ничего, из char2 удаляются начальные и конечные char1. По умолчанию char1 — пробел; если определено только char2, из char2 удаляются начальные и конечные пробелы; если char1 или char2 равно NULL, то возвращается NULL
UPPER(char)	Строка, полученная из char заменой ее строчных букв на заглавные

4.6.3. Даты и время

Таблица 4.16. *Функции работы с датами и временем*

Функции	Возвращаемое значение
`ADD_MONTHS(d,n)`	Дата `d` плюс `n` месяцев
`LAST_DAY(d)`	Дата последнего дня месяца, заданного датой `d`
`MONTHS_BETWEEN (d1,d2)`	Количество месяцев между датами `d1` и `d2`. Если `d1` > `d2`, то результат положителен, иначе отрицателен
`NEW_TIME(d,z1,z2)`	Преобразует дату и время, заданное `d` в часовом поясе `z1`, в дату и время в часовом поясе `z2`. Символьные значения `z1` и `z2` выбираются из следующего списка: • `AST`, `ADT` — Атлантическое стандартное и дневное время; • `BST`, `BDT` — Берингово стандартное и дневное время; • `CST`, `CDT` — Центральное стандартное и дневное время; • `EST`, `EDT` — Восточное стандартное и дневное время; • `GMT` — Среднее время по Гринвичу; • `HST`, `HDT` — Аляски-Гаваев стандартное и дневное время; • `MST`, `MDT` — Монтаны стандартное и дневное время; • `NST` — Нью-Фаунленда стандартное время; • `PST`, `PDT` — Тихоокеанское стандартное и дневное время; • `YST`, `YDT` — Юкона стандартное и дневное время
`NEXT_DAY(d,char)`	Дата первого из дней недели, обозначенной `char`, которая больше или равна `d`
`ROUND(d[,fmt])`	Значение `d`, округленное до ближайшего числа в формате, заданном `fmt` (например, год или месяц). По умолчанию `DD`
`SYSDATE`	Текущая дата и время
`TRUNC(d[,fmt])`	Значение `d`, усеченное до ближайшего числа в формате, заданном `fmt` (например, год или месяц). По умолчанию `DD`

Таблица 4.17. *Форматы функций* `ROUND` *и* `TRUNCATE`

Используемый формат	Значение
`CC	SCC`	Дата первого дня века
`YYYY	SYYYY	YEAR	SYEAR	YYY	YY	Y	IYYY	IY	I`	Дата первого дня года (при округлении: до или после 1 июля) (для года по ISO)
`Q`	Дата первого дня квартала (при округлении: до или после 16 числа второго месяца квартала)
`MONTH	MON	MM	RM`	Дата первого дня месяца (при округлении: до или после 16 числа месяца)
`WW	IW` (для года по ISO)	Дата первого дня недели, начинающейся не с воскресения, а с дня недели, определенного по первому дню *года* (при округлении: до или после 4 дня недели)
`W`	Дата первого дня недели, начинающейся не с воскресения, а с дня недели, определенного по первому дню *месяца* (при округлении: до или после 4 дня недели)
`DDD	DD	J`	Номер дня
`DAY	DY	D`	Дата первого дня недели (воскресенья)
`HH	HH12	HH24`	Час
`MI`	Минута

Таблица 4.18. *Форматы дат и времени*

Формат	Описание
`SCC` или `CC`	Век; `'S'` — префикс перед датой (до н. э.) с '-'
`YYYY	SYYYY`	Год; `'S'` — префикс перед датой (до н. э.) с '-'
`YYY	YY	Y`	Последние 3, 2 или 1 цифра(ы) года
`IYYY`	4 цифры года в стандарте ISO
`IYY	IY	I`	Последние 3, 2 или 1 цифра(ы) года в стандарте ISO
`Y,YYY`	Год с запятой в данной позиции

Таблица 4.18 *(окончание)*

Формат	Описание
SYEAR | YEAR	Год прописью. 'S' — префикс перед датой (до н. э.) с'-'
BC | AD	BC/AD — до н. э. / н. э.
B.C. | A.D.	Аналогично BC/AD
Q	Квартал (четверть) года
MM	Номер месяца
RM	Номер месяца римскими цифрами (I..XII; JAN=I)
MONTH	Имя месяца, дополненное пробелами до девяти символов
MON	Трехбуквенная аббревиатура имени месяца (JAN, FEB,...)
WW | W	Неделя года (1—52) или месяца (1—5)
IW	Неделя года в стандарте ISO (1—21 или 1—53)
DDD DD | D	Номер дня года (1—366), месяца (1—31) или недели (1—7)
DAY	Наименование дня, дополненное пробелами до 9 символов
DY	Трехбуквенная аббревиатура наименования дня
J	День по Юлианскому календарю; количество дней от 31 декабря 4713 г. до н. э.
AM | PM	Указатель часового пояса
A.M. | P.M.	Указатель часового пояса с периодами
HH | HH12	Время суток, интервал 1—12
HH24	Время суток, интервал 0—23
MI	Минуты (0—59)
SS | SSSSS	Секунды (0—59) или после полуночи (0—86399)
-/.,;:	Пунктуация, которая воспроизводится в результате
"...текст..."	Строка в кавычках, которая воспроизводится в результате

4.6.4. Преобразование данных

Таблица 4.19. *Функции преобразования*

Функция	Возвращаемое значение
`CHARTOROWID(char)`	Идентификатор строки (тип данных `ROWID`) из строки `char`
`CONVERT(char,set1 [,set2])`	Преобразованное `char` (по набору символов `set1`). Необязательный аргумент `set2` задает исходный набор символов
`HEXTORAW(char)`	Строка `char`, преобразованная из шестнадцатеричного представления в двоичное — удобное для включения в `RAW`-столбец (столбец с исходными данными)
`RAWTOHEX(raw)`	Строка шестнадцатеричных значений, получаемая из `raw` (исходные данные)
`ROWIDTOCHAR(rowid)`	Символьная строка длиной 18 символов, полученная `rowid` (идентификатор строки)
`TO_CHAR(expr[,fmt [,nls]])`	`expr` преобразуется из числового значения или даты в символьную строку по формату, заданному в `fmt`. Необязательный аргумент `nls` позволяет задать используемый в функции национальный язык. Если `fmt` опущено, то числовое `expr` преобразуется в строку такой длины, которая вмещает только значащие цифры; дата же преобразуется по формату даты согласно умолчанию: `'DD-MON-YY'`
`TO_DATE(char[,fmt [,nls]])`	Преобразование даты из символьного вида в значение даты по формату, заданному в `fmt`. Необязательный аргумент `nls` позволяет задать используемый в функции национальный язык. Если `fmt` опущена, `char` должна иметь формат даты по умолчанию: `'DD-MON-YY'`
`TO_LOB(long_column)`	При выполнении операции `INSERT...SELECT` позволяет преобразовать значения из столбца с типом данных `LONG` или `LONG RAW` в столбец с данными типа `CLOB` или `BLOB`
`TO_MULTI_BYTE(char)`	Преобразование `char` с однобайтовыми символами в многобайтовые символы

Таблица 4.19 *(окончание)*

Функция	Возвращаемое значение
`TO_NUMBER(char[,fmt [,nls]])`	Преобразование `char` в число по формату `fmt`. Необязательный аргумент `nls` позволяет задать используемый в функции национальный символ валюты
`TO_SINGLE_BYTE(char)`	Преобразование `char` с многобайтовыми символами в однобайтовые
`TRANSLATE(char USING {CHAR_CS|NCHAR_CS})`	Преобразует `char` в набор символов национального языка (`NCHAR_CS`) или наоборот (`CHAR_CS`); функция аналогична `CONVERT` и используется с типом данных `NCHAR` и `NVARCHAR2`

Таблица 4.20. *Числовые форматы*

Элемент	Пример	Описание
`9`	`9999`	Количество цифр, определяющих ширину вывода
`0`	`0999`	Вывод ведущих нулей
`$`	`$9999`	Вывод перед значением знака доллара
`B`	`B9999`	Вывод пробелов вместо ведущих нулей
`FM`	`FM90.9`	Вывод без начальных и конечных пробелов
`MI`	`9999MI`	Вывод знака "–" после отрицательных чисел
`S`	`S9999`	Вывод "+" для положительных чисел и "–" для отрицательных
`PR`	`9999PR`	Вывод отрицательных чисел в <угловых скобках>
`D`	`99D99`	Вывод десятичного разделителя
`G`	`9G999`	Вывод разделителя групп (например, триад в денежных данных)
`C`	`C999`	Вывод символа интернациональной денежной единицы
`L`	`L999`	Вывод конкретного денежного символа
`,`	`9,999`	Вывод запятой в указанной позиции

Таблица 4.20 *(окончание)*

Элемент	Пример	Описание
.	99.99	Вывод точки в указанной позиции
V	999V99	Умножение на 10 в N-ой степени, где N задается в виде количества девяток после 'V'
EEEE	9&999EEEE	Вывод в экспоненциальной форме
RN rn	RN rn	Возвращает значения прописными или строчными римскими цифрами (преобразуемое значение между 1 и 3999)

4.6.5. Различные функции для работы с одиночной строкой

Таблица 4.21. *Различные функции*

Функция	Возвращаемое значение
BFILENAME ('directory', 'filename')	Возвращает BFILE (указатель доступа к файлам ОС, где хранятся данные типа LOB)
DUMP(expr[,k[,m[,n]]])	Строка символов, содержащая код типа данных, длину в байтах и внутреннее представление expr. Необязательный аргумент k позволяет задать представление возвращаемого значения: 8 — восьмеричное, 10 — десятичное, 16 — шестнадцатеричное, 17 — одиночные символы. Необязательный аргумент m задает начальную позицию в expr, а необязательный аргумент n — длину возвращаемого значения, начиная с m
GREATEST(expr1, expr2,...)	Наибольшее значение из перечня. Перед сравнением все выражения преобразуются к типу первого выражения
LEAST(expr1, expr2,...)	Наименьшее значение из перечня. Перед сравнением все выражения преобразуются к типу первого выражения
NLS_CHARSET_DECL _LEN (width,char)	Возвращает ширину (width) объявления NCHAR столбца с именем (char)
NLS_CHARSET_ID (char)	Возвращает идентификатор NLS набора символов, соответствующий имени набора (char)

Таблица 4.21 *(окончание)*

Функция	Возвращаемое значение
NLS_CHARSET_NAME (n)	Возвращает имя NLS набора символов, соответствующего номеру идентификатора набора (n)
NVL(expr1,expr2)	Если expr1 равно NULL, возвращает expr2, иначе возвращает expr1. Expr1 и expr2 могут быть любого типа. Тип возвращаемой величины такой же, как для expr1
NVL2 (expr1, expr2, expr3)	Если expr1 равно NULL, возвращает expr3, иначе возвращает expr2. Expr1 имеет любой тип данных, expr2 и expr3 не могут быть типа LONG. Тип возвращаемой величины такой же, как для expr2 или expr3
SYS_CONTEXT ('namespace', 'attribute' [,n])	Возвращает значение атрибута, связанного с контекстом namespace. Значение n позволяет изменять заданную по умолчанию максимальную длину возвращаемого значения (256), доводя ее до 4000
SYS_GUID()	Возвращает глобально уникальный идентификатор
UID	Уникальный целочисленный идентификатор текущего пользователя
USER	Имя текущего пользователя
USERNV(char)	Информация о среде текущего сеанса. Если char равен: 'ISDBA' — возвращает 'TRUE' если вы имеете ISDBA-роль; 'LANGUAGE' — возвращает используемый язык; 'TERMINAL' — возвращается идентификатор терминала пользователя (в терминах операционной системы); 'SESSIONID' — возвращается идентификатор сеанса пользователя; 'ENTRYID' — возвращается доступный идентификатор элемента, за которым идет слежение; 'LANG' — возвращает ISO-аббревиатуру текущего языка; 'INSTANCE' — возвращает номер текущего экземпляра (INSTANCE); 'CLIENT_INFO' — возвращает до 64 байтов информации сеанса; она может быть сохранена прикладной программой, использующей пакет DBMS_APPLICATION_INFO
VSIZE(expr)	Число байтов во внутреннем представлении expr

4.6.6. Агрегатные функции

Агрегатные функции возвращают результат обработки группы строк (табл. 4.22). Они имеют смысл в конструкциях SELECT, ORDER BY, HAVING и в аналитических функциях.

Использование DISTINCT позволяет учитывать только различающиеся значения аргумента expr. При указании ALL (или по умолчанию) учитываются все значения expr. Например, DISTINCT при нахождении среднего значения из 1,1,1 и 3 дает результат 2, тогда как ALL при этой же операции дает результат 1.5.

Таблица 4.22. *Агрегатные функции*

Функция	Возвращаемое значение
AVG([DISTINCT | ALL] expr)	Среднее значение expr, с игнорированием пустых (NULL) значений
CORR(expr1,expr2)	Возвращает коэффициент корреляции набора пар значений expr1, expr2; для его определения используется формула: COVAR_POP(expr1,expr2)/ (STDDEV_POP(expr1)* STDDEV_POP(expr2))
COUNT({[DISTINCT | ALL] expr |*})	Количество строк, в которых expr не является пустым (NULL) значением. Установка * позволяет подсчитать все выбранные строки, включая строки с NULL-значениями
COVAR_POP(expr1, expr2)	Значение (СУММА(expr1*expr2) - СУММА(expr2)* СУММА(expr1)/n)/n, где n — номер пары (expr1,expr2), в которой ни expr1, ни expr2 не содержат NULL
COVAR_SAMP(expr1, expr2)	Значение (СУММА(expr1*expr2) - СУММА(expr1)*СУММА(expr2)/n)/(n-1), где n — номер пары (expr1,expr2), в которой ни expr1, ни expr2 не содержат NULL
MAX([DISTINCT | ALL] expr)	Максимальное значение expr
MIN([DISTINCT | ALL] expr)	Минимальное значение expr

Таблица 4.22 *(окончание)*

Функция	Возвращаемое значение
STDDEV([DISTINCT | ALL] expr)	Среднеквадратичное (стандартное) отклонение от expr с игнорированием пустых (NULL) значений
SUM([DISTINCT | ALL] expr)	Сумма значений expr
VARIANCE([DISTINCT | ALL] expr)	Дисперсия expr, с игнорированием пустых значений

4.6.7. Функции *CASE*, *CAST* и *DECODE*

Функция CASE позволяет использовать выражения IF-THEN-ELSE в инструкциях SELECT и UPDATE. Эта функция проверяет список условий и возвращает одно из нескольких возможных значений.

Инструкция CASE имеет два способа применения: простой и поисковый.

```
-- Простая операция сравнения
CASE выражение
WHEN искомое_значение THEN результат [...]
[ELSE альтернативный_результат]
END
```

В простой функции CASE входное значение (выражение) проверяется на соответствие каждому предложению WHEN. Результирующее значение (результат) возвращается при первой истинности:

```
выражение = искомое_значение
```

Если ни одно из условий when не оказывается истинным, то возвращается альтернативный_результат. Если это альтернативное значение не указано, то возвращается NULL.

Рассмотрим пример преобразования оценок, хранимых в таблице Ведомости базы данных "UCHEB", в официальные текстовые значения. В примере использованы синтаксические конструкции, которые подробно рассматриваются в *главе 5*.

```
SELECT оценка,
       CASE оценка
           WHEN '5'      THEN 'отлично'
```

```
            WHEN '4'      THEN 'хорошо'
            WHEN '3'      THEN 'удовлетворительно'
            WHEN '2'      THEN 'неудовлетворительно'
            WHEN 'зачет'  THEN 'зачет'
            WHEN 'незач'  THEN 'незачет'
            WHEN 'осв'    THEN 'освобождение'
            WHEN 'неявка' THEN 'неявка'
            ELSE 'отсутствие данных'
        END официальное_значение, дата, состояние
FROM н_ведомости
WHERE члвк_ид = 116483;
```

Часть оценок студента с номером 116483 имеет следующий вид:

```
ОЦЕНКА   ОФИЦИАЛЬНОЕ_ЗНАЧЕНИЕ ДАТА       СОСТОЯНИЕ
-------- -------------------- ---------- ------------
5        отлично              24.12.1998 актуальна
4        хорошо               29.12.1998 актуальна
зачет    зачет                05.01.1999 актуальна
3        удовлетворительно    16.06.1999 актуальна
.        отсутствие данных    05.01.2001 актуальна
неявка   неявка               28.12.2001 неактуальна
...
зачет    зачет                05.06.2007 актуальна
неявка   неявка               19.06.2007 актуальна
2        неудовлетворительно  19.06.2007 актуальна
...
```

Поисковые выражения `CASE` позволяют проанализировать несколько логических условий, и как только одно из них оказывается истинным, выражение возвращает соответствующий этому условию результат.

```
-- Булева поисковая операция
CASE
WHEN логическое_условие THEN результат [...]
[ELSE альтернативный_результат]
END
```

Структура более сложной логической поисковой операции, аналогична структуре простой операции сравнения за исключением того, что для каждого предложения `WHEN` существует своя логическая операция сравнения (не обязательно "=").

Так, предыдущий пример можно записать и в таком виде:

```
SELECT оценка,
       CASE
           WHEN оценка = '5'      THEN 'отлично'
           WHEN оценка = '4'      THEN 'хорошо'
           WHEN оценка = '3'      THEN 'удовлетворительно'
           WHEN оценка = '2'      THEN 'неудовлетворительно'
           WHEN оценка = 'зачет'  THEN 'зачет'
           WHEN оценка = 'незач'  THEN 'незачет'
           WHEN оценка = 'осв'    THEN 'освобождение'
           WHEN оценка = 'неявка' THEN 'неявка'
           ELSE 'отсутствие данных'
       END официальное_значение, дата, состояние
FROM н_ведомости
WHERE члвк_ид = 116483:
```

Приведем более сложный пример с подсчетом количества различных оценок, полученных студентом за время обучения в университете.

```
SELECT
       CASE
           WHEN оценка = '5'      THEN 'отлично'
           WHEN оценка = '4'      THEN 'хорошо'
           WHEN оценка = '3'      THEN 'удовлетворительно'
           WHEN оценка = '2'      THEN 'неудовлетворительно'
           WHEN оценка = 'зачет'  THEN 'зачет'
           WHEN оценка = 'незач'  THEN 'незачет'
           WHEN оценка = 'осв'    THEN 'освобождение'
           WHEN оценка = 'неявка' THEN 'неявка'
           ELSE 'отсутствие данных'
       END официальное_значение, оценка, COUNT(*) количество
FROM н_ведомости
WHERE члвк_ид = 116483
GROUP BY оценка;
```

Результат выполнения запроса:

```
ОФИЦИАЛЬНОЕ_ЗНАЧЕНИЕ ОЦЕНКА   КОЛИЧЕСТВО
-------------------- -------- ----------
отсутствие данных    .                 6
```

```
неудовлетворительно  2              4
удовлетворительно    3             66
хорошо               4             26
отлично              5              2
зачет                зачет         91
неявка               неявка         2
```

Функция CAST неявно преобразует выражение с одним типом данных в другой тип и имеет синтаксис:

```
CAST (выражение AS тип_данных [(длина)])
```

Функция CAST преобразует любое выражение, например значение в столбце или переменную, в другой, указанный тип данных. Для тех типов, которые поддерживают указание длины (например, CHAR или VARCHAR), можно указать длину.

Отметим, что некоторые преобразования, например такие, как преобразование значений DECIMAL к INTEGER, приводят к операциям округления. Кроме того, некоторые преобразования могут завершиться ошибкой, если у нового типа данных не достаточно места для отображения преобразованного значения.

Функция DECODE имеет синтаксис

```
DECODE(выражение, искомое_значение1, результат1 [,искомое_значение2, результат2 [,…]], [, по_умолчанию])
```

Функция сравнивает выражение с искомым_значением1. Если выражение равно ему, то возвращается результат1. В противном случае выражение сравнивается с искомым_значением2 и т. д. Если совпадение не обнаруживается, функция возвращает значение по_умолчанию или NULL, если это значение опущено.

Реализуем с помощью функции DECODE рассмотренный ранее пример преобразования оценок:

```
SELECT оценка,
     (DECODE(оценка,'5','отлично',
                    '4','хорошо',
                    '3','удовлетворительно',
                    '2','неудовлетворительно',
                    'зачет', 'зачет',
                    'незач', 'незачет',
                    'осв', 'освобождение',
                    'неявка', 'неявка',
                    'отсутствие данных')) эквивалент, дата, состояние
FROM н_ведомости
WHERE члвк_ид = 116483;
```

Глава 5

Запросы с использованием единственной таблицы

5.1. О предложениях *SELECT* и *SUBQUERY*

SELECT — предложение языка SQL извлекает строки, столбцы и производные значения (например, вычисляемые выражения) из одной или нескольких таблиц базы данных.

В большинстве приложений SELECT является наиболее часто используемым оператором DML. При формировании запроса SELECT пользователь описывает ожидаемый набор данных, но не указывает, какие физические операции должны быть произведены для получения этого набора. Определение для запроса оптимального плана является задачей СУБД, а точнее — оптимизатора.

Полный синтаксис стандартного предложения SELECT очень мощный и сложный, но его можно разделить на следующие основные фразы.

```
SELECT [{ALL | DISTINCT}] отбираемый_элемент [AS псевдоним] [,...]
FROM [ONLY | OUTER]
   { имя_таблицы [[AS] псевдоним] |
    имя_представления [[AS] псевдоним] } [,...]
   [ [тип_соединения] JOIN условие_соединения ]
[WHERE условие_поиска [ {AND | OR | NOT} условие_поиска [...]]
[[ START WITH условие ] CONNECT BY [PRIOR] условие ]
[GROUP BY группировка_по_выражению { группировка_по_столбцам |
     ROLLUP группировка_по_столбцам | CUBE группировка_по_столбцам |
     GROUPING SETS ( список_наборов_группировок ) | (  ) |
      набор_группировок, список__наборов_ группировок }
   [HAVING условие_поиска] ]
[ORDER BY {выражение_для_сортировки [ASC | DESC]} [,...] ];
```

Расшифровка сокращений *отбираемый_элемент*, *псевдоним*, *имя_таблицы*, *имя_представления*, *условие_соединения*, *условие_поиска* и других, входящих в этот синтаксис, намного его усложнит. Еще более сложным и громоздким будет выглядеть синтаксис SELECT для Oracle Database 10g, в котором существует очень большое количество расширений стандарта SQL:2003 [6].

Поэтому приведем здесь лишь краткое назначение основных фраз предложения SELECT, перенеся их подробное изучение в следующие разделы данной главы и *главы 6*:

- FROM — содержит перечень таблиц, из которых должна выводиться запрашиваемая информация;
- SELECT — содержит список столбцов из таблиц, перечисленных во фразе FROM, а также выражений, использующих значения из этих таблиц;
- WHERE — включает набор условий для отбора определенных строк таблиц, из которых должна выводиться запрашиваемая информация;
- CONNECT BY — определяет условие соотношения между родительскими строками и их строками-потомками в иерархических запросах (см. *разд. 5.6*);
- GROUP BY — разбивает выводимые строки на логические группы, применяя к ним указанные в SELECT агрегатные функции (COUNT, SUM, MAX и др.), создавая для каждой из групп итоговую строку с количеством, суммой, максимумом и др. (см. *разд. 4.6*);
- HAVING — включает набор условий для отбора определенных итоговых строк из тех, которые получаются при выполнении фразы GROUP BY;
- ORDER BY — сортирует полученный набор строк или итоговых строк.

Напомним, что SELECT, как и любое другое предложение языка SQL, должно заканчиваться символом точка с запятой.

Предложение SUBQUERY (подзапрос) — это предложение SELECT, заключенное в круглые скобки и встроенное в тело другого запроса. Подзапросы (вложенные запросы) позволяют получать одно или несколько значений и помещать их в предложения SELECT, INSERN, UPDATE или DELETE или в другой подзапрос. Подробно о них будет рассказано в *главе 6*.

5.2. Выборка без использования фразы *WHERE*

В этом разделе мы будем рассматривать простейшую форму предложения SELECT:

```
SELECT ALL | DISTINCT}] отбираемый_элемент [[AS] псевдоним] [,...]
FROM имя_таблицы [[AS] псевдоним];
```

Опишем кратко ключевые слова этого предложения.

`[{ALL | DISTINCT}] отбираемый_элемент`

Используется для описания результирующего набора значений, извлекаемых из базы данных при выполнении предложения `SELECT`.

`Отбираемый_элемент` может представлять собой константу, агрегат или скалярную функцию, математическое выражение, параметры или переменную или же подзапрос, но наиболее часто *`отбираемый_элемент`* — это столбец таблицы или представления. Несколько таких элементов должны разделяться запятыми.

Если данные извлекаются из контекста, отличного от контекста текущего пользователя, перед столбцом нужно указывать префикс в виде схемы или имени владельца. Если таблица принадлежит другому пользователю, то в ссылке на столбец указывать имя этого пользователя. Чтобы извлечь все столбцы таблицы, указанной в предложении `FROM`, можно использовать краткое написание в виде звездочки (`*`).

Ключевое слово `ALL`, заданное по умолчанию, приводит к извлечению всех записей, удовлетворяющих критерию отбора. Ключевое же слово `DISTINCT` заставляет базу данных отфильтровывать все дублирующиеся записи и возвращать экземпляр из нескольких идентичных записей.

`AS` *`псевдоним`* заменяет имя столбца более кратким именем. Это предложение особенно полезно для замены непонятных или длинных имен и выражений понятными обозначениями.

`FROM` *`имя_таблицы`* `[[AS]` *`псевдоним`*`]`

Это предложение служит для указания таблицы, из которой запрос получает данные. Предложение `FROM` также позволяет назначать псевдонимы длинным именам таблиц.

Отметим, что псевдонимы в этой и предыдущей фразе действуют только в пределах одного запроса, а также, что слово `[AS]` можно опустить.

Здесь и далее мы будем иллюстрировать результаты выполнения предложений SQL на примерах работы с базой данных "COOK", содержимое которой приведено в табл. 3.1—3.10 данной книги. Иногда (для более сложных примеров) мы воспользуемся достаточно содержательной базой данных "UCHEB" (см. *главу 19*), содержимое которой приведено на компакт-диске, прилагаемом к книге. Последнее будет каждый раз специально оговариваться.

5.2.1. Простая выборка

Сформулируем запрос на получения названия, статуса и месторасположения каждого из поставщиков:

```
SELECT название, статус, город
FROM поставщики;

НАЗВАНИЕ СТАТУС     ГОРОД
-------- ---------- ---------
СЫТНЫЙ   рынок      Ленинград
ПОРТОС   кооператив Резекне
ШУШАРЫ   совхоз     Пушкин
ТУЛЬСКИЙ универсам  Ленинград
УРОЖАЙ   кооптorг   Луга
ЛЕТО     агрофирма  Ленинград
ОГУРЕЧИК ферма      Паневежис
КОРЮШКА  кооператив Йыхви
```

При необходимости получения полной информации о поставщиках, можно было бы дать запрос

```
SELECT код_поставщика, название, статус, город, адрес, телефон
FROM Поставщики;
```

или более краткую нотацию, где вместо перечня всех имен таблицы `Поставщики` используется рассмотренный в предыдущем разделе символ (`*`):

```
SELECT *
FROM Поставщики;
```

Результат любого из этих запросов имеет вид:

```
КОД_ПОСТАВЩИКА НАЗВАНИЕ СТАТУС     ГОРОД     АДРЕС          ТЕЛЕФОН
-------------- -------- ---------- --------- -------------- -------
             1 СЫТНЫЙ   рынок      Ленинград Сытнинская,3   2329916
             2 ПОРТОС   кооператив Резекне   Садовая,27      317664
             3 ШУШАРЫ   совхоз     Пушкин    Новая, 17      4705038
             4 ТУЛЬСКИЙ универсам  Ленинград Тульский,5     2710837
             5 УРОЖАЙ   кооптorг   Луга      Песчаная,19     789000
             6 ЛЕТО     агрофирма  Ленинград Пулковская,8   2939729
             7 ОГУРЕЧИК ферма      Паневежис Укмерге,15      127331
             8 КОРЮШКА  кооператив Йыхви     Нарвское ш.,64  432123
```

Еще один пример. Выдать основу всех блюд:

```
SELECT Основа
FROM Блюда;
```

дает результат, в котором среди десяти первых значений всего лишь четыре уникальных: овощи, мясо, рыба и молоко.

```
ОСНОВА
------
Овощи
Мясо
Овощи
Рыба
Рыба
Мясо
Молоко
Молоко
Мясо
Рыба
...
```

5.2.2. Исключение дубликатов

В предыдущем примере был выдан правильный, но не совсем удачный перечень основных продуктов: из него не были исключены дубликаты. Это можно сделать, дополнив запрос ключевым словом `DISTINCT` (различный, различные), что позволит получить следующий результат:

```
SELECT DISTINCT Основа
FROM Блюда;
```

```
ОСНОВА
------
Кофе
Крупа
Молоко
Мясо
Овощи
Рыба
Фрукты
Яйца
```

Здесь всего восемь строк, в которых содержатся все уникальные значения столбца `Основа` таблицы `Блюда`.

5.2.3. Выборка вычисляемых значений

Из описания синтаксиса предложения SELECT известно, что в нем может содержаться не только перечень столбцов таблицы или символ *, но и выражения.

Например, если нужно получить значение калорийности всех продуктов, то можно учесть, что при окислении 1 г углеводов или белков в организме освобождается в среднем 4.1 ккал, а при окислении 1 г жиров — 9.3 ккал, и выдать запрос:

```
SELECT Продукт, ((Белки + Углев)*4.1 + Жиры*9.3)
FROM   Продукты;
```

позволяющий получить следующий результат:

```
ПРОДУКТ  (БЕЛКИ+УГЛЕВ)*4.1+ЖИРЫ*9.3
-------- --------------------------
Говядина                     1928,1
Судак                          1523
Масло                        8287,5
Майонез                      6464,7
Яйца                         1618,9
Сметана                      3011,4
Молоко                        605,1
Творог                         1575
Морковь                       349,6
Лук                           459,2
Помидоры                      196,8
Зелень                        118,9
Рис                          3512,1
Мука                         3556,7
Яблоки                        479,7
Сахар                        4091,8
Кофе                          892,4
```

Фраза SELECT может включать не только выражения для вычисления числовых значений, но и текстовые выражения, а также числовые или текстовые константы.

В качестве примера дополним предыдущий запрос константой 'Калорий =', расположенной перед вычисляемым значением, и дадим псевдоним Калор выражению для вычисления калорийности продуктов:

```
SELECT Продукт, 'Калорий =',
           ((Белки + Углев)*4.1 + Жиры *9.3) Калор
FROM   Продукты;
```

Результат его выполнения будет иметь вид:

```
ПРОДУКТ  'КАЛОРИЙ='      КАЛОР
-------- ---------- ----------
Говядина Калорий =      1928,1
Судак    Калорий =        1523
. . .
```

Попробуем теперь создать текстовое выражение, воспользовавшись для этого оператором конкатенации (см. *разд. 4.4.4*). Создадим набор строк, каждая из которых содержит все данные о поставщике кроме его кода:

```
SELECT статус||' '||название||' - адрес: '
       ||город||', '||адрес Поставщики
FROM Поставщики;
```

Результат выполнения такого запроса имеет вид:

```
ПОСТАВЩИКИ
--------------------------------------------------
рынок СЫТНЫЙ - адрес: Ленинград, Сытнинская,3
кооператив ПОРТОС - адрес: Резекне, Садовая,27
совхоз ШУШАРЫ - адрес: Пушкин, Новая, 17
универсам ТУЛЬСКИЙ - адрес: Ленинград, Тульский,5
кооптторг УРОЖАЙ - адрес: Луга, Песчаная,19
агрофирма ЛЕТО - адрес: Ленинград, Пулковская,8
ферма ОГУРЕЧИК - адрес: Паневежис, Укмерге,15
кооператив КОРЮШКА - адрес: Йыхви, Нарвское ш.,64
```

Используя разнообразные функции работы со строками (см. *разд. 4.6*) можно существенно "разукрасить" этот результат. Воспользуемся только одной из них — `RPAD`, позволяющей дополнить строку справа до указанной длины пробелами (по умолчанию) или любым набором символов:

```
SELECT RPAD(статус||' '||название,20)||' - адрес: '
||город||', '||адрес Поставщики
FROM Поставщики;

ПОСТАВЩИКИ
-----------------------------------------------------
рынок СЫТНЫЙ          - адрес: Ленинград, Сытнинская,3
кооператив ПОРТОС     - адрес: Резекне, Садовая,27
совхоз ШУШАРЫ         - адрес: Пушкин, Новая, 17
универсам ТУЛЬСКИЙ    - адрес: Ленинград, Тульский,5
кооптторг УРОЖАЙ      - адрес: Луга, Песчаная,19
```

```
агрофирма ЛЕТО       - адрес: Ленинград, Пулковская,8
ферма ОГУРЕЧИК       - адрес: Паневежис, Укмерге,15
кооператив КОРЮШКА   - адрес: Йыхви, Нарвское ш.,64
```

5.3. Выборка с использованием фразы *WHERE*

Даже в такой микроскопической базе данных, какой является рассматриваемая нами база данных "COOK", нас часто интересуют только определенные строки таблиц. Для этого в SQL существует фраза WHERE, позволяющая указать критерии для отбора нужных строк.

Синтаксис этой фразы имеет вид:

```
WHERE условие_поиска [ {AND | OR | NOT} условие_поиска [...],
```

где *условие_поиска* одна из следующих конструкций:

- `значение { = | <> | < | <= | > | >= } { значение | (подзапрос) }`
- `значение_1 [NOT] BETWEEN значение_2 AND значение_3`
- `значение [NOT] IN { (константа [,константа]...) | (подзапрос) }`
- `значение IS [NOT] NULL`
- `[таблица.]столбец [NOT] LIKE 'строка_символов' [ESCAPE 'символ']`
- `EXISTS (подзапрос)`

Кроме традиционных операторов сравнения (= | <> | < | <= | > | >=) в WHERE-фразе используются условия BETWEEN (между), LIKE (похоже на), IN (принадлежит), IS NULL (не определено) и EXISTS (существует), которые могут предваряться оператором NOT (не). Критерий отбора строк формируется из одного или нескольких условий, соединенных логическими операторами:

- AND — когда должны удовлетворяться оба разделяемых с помощью AND условия;
- OR — когда должно удовлетворяться одно из разделяемых с помощью OR условий;
- AND NOT — когда должно удовлетворяться первое условие и не должно второе;
- OR NOT — когда или должно удовлетворяться первое условие или не должно удовлетворяться второе.

При этом существует приоритет AND над OR (сначала выполняются все операции AND и только после этого операции OR). Для получения желаемого

результата WHERE-условия должны быть введены в правильном порядке, который можно организовать с помощью скобок.

При обработке условия числа сравниваются алгебраически — отрицательные числа считаются меньшими, чем положительные, независимо от их абсолютной величины. Строки символов сравниваются в соответствии с их представлением в коде, используемом в конкретной СУБД, например, в коде ASCII. Если сравниваются две строки символов, имеющих разные длины, более короткая строка дополняется справа пробелами для того, чтобы они имели одинаковую длину перед осуществлением сравнения.

В заключение отметим, что плохо написанная фраза WHERE может ухудшить производительность полезного во всех прочих отношениях предложения SELECT, поэтому деталями использования фразы WHERE необходимо владеть хорошо.

5.3.1. Использование операторов сравнения

В синтаксисе фразы WHERE показано, что для отбора нужных строк таблицы можно использовать операторы сравнения = (равно), <> (не равно), < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), которые могут предваряться оператором NOT, создавая, например, отношения "не меньше" и "не больше".

Так, для получения перечня продуктов, практически не содержащих углеводов, можно сформировать запрос

```
SELECT Продукт, Белки, Жиры, Углев, K, Ca, Na, B2, PP, C
FROM   Продукты
WHERE  Углев = 0;
```

и получить:

```
ПРОДУКТ   БЕЛКИ   ЖИРЫ  УГЛЕВ     K    CA    NA     B2     PP     C
-------- ------ ------ ------ ----- ----- ----- ------ ------ -----
Говядина  189,0  124,0    0,0  3150    90   600    1,5   28,0     0
Судак     190,0   80,0    0,0  1870   270     0    1,1   10,0    30
```

Возможность использования нескольких условий, соединенных логическими операторами AND, OR, AND NOT и OR NOT, позволяет осуществить более детальный отбор строк. Так, для получения перечня продуктов, практически не содержащих углеводов и натрия, можно сформировать запрос:

```
SELECT Продукт, Белки, Жиры, Углев, K, Ca, Na, B2, PP, C
FROM   Продукты
WHERE  Углев = 0 AND Na = 0;
```

Результат запроса имеет вид:

```
ПРОДУКТ   БЕЛКИ   ЖИРЫ  УГЛЕВ     К    CA    NA     B2     PP     C
-------- ------ ------ ------ ----- ----- ----- ------ ------ -----
Судак     190,0   80,0    0,0  1870   270     0    1,1   10,0    30
```

Добавим к этому запросу еще одно условие:

```
SELECT Продукт, Белки, Жиры, Углев, K, Ca, Na, B2, PP, C
FROM   Продукты
WHERE  Углев = 0 AND Na = 0 AND Продукт <> 'Судак';
```

и получим на экране сообщение "No rows exist or satisfy the specified clause" или "строки не выбраны" или просто заголовок без строк, в зависимости от вида и настройки того приложения, с помощью которого производится реализация запроса.

5.3.2. Использование *BETWEEN*

С помощью `BETWEEN ... AND ...` (находится в интервале от ... до ...) можно отобрать строки, в которых значение какого-либо столбца находится в заданном диапазоне.

Например, выдать перечень продуктов, в которых значение содержания белка находится в диапазоне от 10 до 50:

```
SELECT Продукт, Белки
FROM   Продукты
WHERE  Белки BETWEEN 10 AND 50;

ПРОДУКТ   БЕЛКИ
-------- ------
Майонез    31,0
Сметана    26,0
Молоко     28,0
Морковь    13,0
Лук        17,0
```

Можно задать и `NOT BETWEEN` (не принадлежит диапазону), например:

```
SELECT Продукт, Белки, Жиры
FROM   Продукты
WHERE  Белки NOT BETWEEN 10 AND 50
AND    Жиры > 100;
```

```
ПРОДУКТ    БЕЛКИ   ЖИРЫ
-------- ------ ------
Говядина  189,0  124,0
Масло      60,0  825,0
Яйца      127,0  115,0
```

BETWEEN особенно удобен при работе с данными, задаваемыми интервалами, начало и конец которых расположены в разных столбцах.

Для примера воспользуемся таблицей "минимальных окладов" (табл. 5.1), величина которых непосредственно связана с зарплатами сотрудников университета и другими выплатами. В этой таблице для текущего значения минимального оклада установлена запредельная дата окончания 9 сентября 9999 года.

Таблица 5.1. Минимальные оклады

Миноклад	Начало	Конец
90	01.05.1997	31.03.1999
110	01.04.1999	31.12.2000
200	01.01.2001	30.06.2001
300	01.07.2001	30.11.2001
450	01.12.2001	30.09.2003
600	01.10.2003	31.12.2004
720	01.01.2005	31.08.2005
800	01.09.2005	30.04.2006
1100	01.05.2006	30.09.2006
1221	01.10.2006	09.09.9999

Если, например, потребовалось узнать, какие изменения минимальных окладов производились в 2000/2001 учебном году, то можно выдать запрос:

```
SELECT Начало, Миноклад
FROM   Минимумы
WHERE  Начало BETWEEN '1-9-2000' AND '31-8-2001';
```

и получить результат:

```
НАЧАЛО      МИНОКЛАД
----------- --------
01.01.2001    200,00
01.07.2001    300,00
```

Отметим, что при формировании запросов значения дат следует заключать в апострофы, чтобы СУБД не путала их с выражениями и не пыталась вычитать из 31 значение 8, а затем 2001.

Для выявления всех значений минимальных окладов, которые существовали в 2000/2001 учебном году, можно сформировать запрос:

```
SELECT *
FROM   Миноклады
WHERE  Начало BETWEEN '1-9-2000' AND '31-8-2001'
OR     Конец  BETWEEN '1-9-2000' AND '31-8-2001'
```

получив:

```
МИНОКЛАД НАЧАЛО     КОНЕЦ
-------- ---------- ----------
  110,00 01.04.1999 31.12.2000
  200,00 01.01.2001 30.06.2001
  300,00 01.07.2001 30.11.2001
```

Наконец, для определения минимального оклада на 15-5-2001 можно дать запрос:

```
SELECT Миноклад
FROM   Миноклады
WHERE  '15-5-2001' BETWEEN Начало AND Конец;
```

получив:

```
МИНОКЛАД
--------
  300,00
```

5.3.3. Использование *IN*

Выдать сведения о блюдах на основе яиц, крупы и овощей:

```
SELECT *
FROM   Блюда
WHERE  Основа IN ('Яйца', 'Крупа', 'Овощи');
```

Результат:

```
КОД_БЛЮДА БЛЮДО            КОД_ВИДА ОСНОВА  ВЫХОД ТРУД
--------- ---------------- -------- ------ ------ ----
        1 Салат летний            1 Овощи   200,0    3
        3 Салат витаминный        1 Овощи   200,0    4
       16 Драчена                 3 Яйца    180,0    4
```

```
17 Морковь с рисом          3 Овощи   260,0    3
19 Омлет с луком            3 Яйца    200,0    5
20 Каша рисовая             3 Крупа   210,0    4
21 Пудинг рисовый           3 Крупа   160,0    6
23 Помидоры с луком         3 Овощи   260,0    4
```

Рассмотренная форма IN является в действительности просто краткой записью последовательности отдельных сравнений, соединенных операторами OR. Предыдущее предложение эквивалентно такому:

```
SELECT *
FROM   Блюда
WHERE  Основа = 'Яйца' OR Основа = 'Крупа' OR Основа = 'Овощи';
```

Можно задать и NOT IN (не принадлежит), также есть возможность использования IN (NOT IN) с подзапросом (см. *главу 6*).

5.3.4. Использование *LIKE*

Оператор LIKE позволяет указывать строковый шаблон для проверки на совпадение в предложениях SELECT, INSERT, UPDATE и DELETE. Строковый шаблон может включать обобщающие символы. И в стандарте и в Oracle поддерживаются два обобщающих символа:

- символ _ (подчеркивание) заменяет любой одиночный символ;
- символ % (процент) заменяет любую последовательность из N символов (где N может быть нулем).

Существует также ключевое слово ESCAPE (*отменяющая_посдедовательность*), позволяющее включать в шаблон символы, которые в обычных условиях интерпретировались бы как обобщающие символы. Любой обобщающий символ, перед которым стоит *отменяющая_посдедовательность*, не считается обобщающим, а считается обычным символом.

При организации проверки по строковым шаблонам во фразе LIKE следует помнить:

- значимыми являются все символы, включая пробелы перед строкой и после нее;
- с помощью фразы LIKE можно сравнивать разные типы данных, но строки в них хранятся по-разному. Так, надо представлять различия между типами данных CHAR, VARCHAR и DATE (см. *разд. 4.5*);
- использование фразы LIKE может заставить пользователя СУБД отказаться от употребления индексов или применять альтернативные, менее оптимальные индексы, чем в простой операции сравнения.

Синтаксис фразы LIKE в Oracle имеет вид:

```
WHERE выражение [NOT] {LIKE | LIKEC | LIKE2 | LIKE4 }
      строковый_шаблон [ESCAPE  отменяющая_посдедовательность ]
```

где LIKEC использует полный набор символов UNICODE, LIKE2 использует набор символов UNICODE USC2, а LIKE4 использует набор символов UNICODE USC4.

Поскольку Oracle учитывает регистр, следует включать строковый_шаблон, выражение или и то и другое в функцию UPPER (изменение регистра строки на верхний). В этом случае вы всегда будете сравнивать то, что нужно.

Перейдем к примерам. Выдать перечень салатов:

```
SELECT Блюдо
FROM   Блюда
WHERE  Блюдо LIKE 'Салат%';
```

Результат выборки имеет вид:

```
БЛЮДО
---------------
Салат летний
Салат мясной
Салат витаминный
Салат рыбный
```

В этом примере SELECT осуществляет выборку тех строк таблицы Блюда, для которых значение в столбце Блюдо начинается сочетанием "Салат" и содержит любую последовательность из нуля или более символов, следующих за этим сочетанием. Если бы среди блюд были "Луковый салат", "Фруктовый салат" и т. п., то они не были бы найдены. Для их отыскания надо изменить фразу WHERE:

```
WHERE Блюдо LIKE '%салат%'
```

или исключить возможные различия между малыми и большими буквами:

```
WHERE UPPER (Блюдо) LIKE UPPER ('%Салат%')
```

Это позволит отыскать все салаты.

5.4. Выборка с упорядочением (*ORDER BY*)

Синтаксис фразы упорядочения имеет вид:

```
ORDER BY { выражение для сортировки [ASC | DESC ] } [,...]
```

В выражении для сортировки указывается тот элемент запроса, который будет определять порядок данных в результирующем наборе. Обычно это имена

или псевдонимы столбцов. Указывается также, что выражение для сортировки должно возвращаться в восходящем (ASC) или нисходящем (DESC) порядке. По умолчанию используется восходящий порядок: от ранней даты к поздней, от младшего числового значения к старшему, от текстового значения, начинающегося на символ с наименьшим кодом ASCII, к символу с наибольшим кодом ASCII (для русских букв от "А" до "Я" и от "а" до "я").

Можно задавать несколько выражений для сортировки. При этом используется порядок сортировки от главного к второстепенному: сначала результирующий набор сортируется по первому из указанных столбцов, затем одинаковые значения первого столбца сортируются по второму столбцу, одинаковые значения второго столбца — по третьему столбцу и т. д.

Индивидуальные параметры сортировки столбца (ASC/DESC) не зависят от других столбцов фразы ORDER BY, т. е. можно отсортировать результирующий набор по одному столбцу по возрастанию, по следующему — по убыванию и т. п.

Пустые значения (NULL) при сортировке всегда оказываются рядом (т. е. считаются одинаковыми). В Oracle по умолчанию (ASC) они собираются внизу. При помощи DESC можно переместить их вверх.

Наконец, в Oracle поддерживается выведенная из стандарта сортировка по номеру порядковой позиции столбца (псевдонима, выражения или подзапроса) в списке SELECT.

Перейдем к примерам.

Выдать перечень продуктов, содержание в них основных веществ в порядке убывания содержания белка и возрастания их калорийности:

```
SELECT продукт, белки, жиры, углев,
      (Белки + Углев)*4.1 + Жиры*9.3 Калорий
FROM продукты
ORDER BY Белки DESC, Калорий ASC;
```

Результат выполнения этого запроса имеет вид:

```
ПРОДУКТ   БЕЛКИ   ЖИРЫ  УГЛЕВ    КАЛОРИЙ
-------- ------ ------ ------ ----------
Судак     190,0   80,0    0,0       1523
Говядина  189,0  124,0    0,0     1928,1
Творог    167,0   90,0   13,0       1575
Кофе      127,0   36,0    9,0      892,4
Яйца      127,0  115,0    7,0     1618,9
Мука      106,0   13,0  732,0     3556,7
Рис        70,0    6,0  773,0     3512,1
```

```
Масло       60,0  825,0   90,0     8287,5
Майонез     31,0  670,0   26,0     6464,7
Молоко      28,0   32,0   47,0      605,1
Сметана     26,0  300,0   28,0     3011,4
Лук         17,0    0,0   95,0      459,2
Морковь     13,0    1,0   70,0      349,6
Зелень       9,0    0,0   20,0      118,9
Помидоры     6,0    0,0   42,0      196,8
Яблоки       4,0    0,0  113,0      479,7
Сахар        0,0    0,0  998,0     4091,8
```

Для еще одного примера воспользуемся базой данных "UCHEB". Реализуем запрос

```
SELECT ид, фамилия, имя, отчество, дата_рождения
FROM н_люди
WHERE фамилия = 'Громов'
ORDER BY имя, отчество, дата_рождения DESC;
```

в котором отыскиваются все люди с фамилией Громов, результат сортируется по имени и отчеству, а при одинаковых значениях имени и отчества, по дате рождения (от более поздней к более ранней).

```
    ИД ФАМИЛИЯ  ИМЯ       ОТЧЕСТВО      ДАТА_РОЖДЕНИЯ
------ -------- --------- ------------- -------------
125518 Громов   Александр Анатольевич   18.05.1960
123040 Громов   Александр Вадимович     19.09.1958
102709 Громов   Александр Викторович    23.12.1961
142506 Громов   Александр Дмитриевич    15.04.1986
129531 Громов   Александр Олегович      26.10.1985
117245 Громов   Андрей    Викторович    24.03.1981
133898 Громов   Андрей    Владимирович  25.09.1986
122012 Громов   Андрей    Геннадьевич   18.10.1984
143792 Громов   Валерий   Александрович 13.06.1988
119218 Громов   Валерий   Геннадьевич   02.07.1983
100061 Громов   Геннадий  Юрьевич       29.05.1962
132910 Громов   Дмитрий   Александрович 17.04.1987
145639 Громов   Дмитрий   Евгеньевич    24.12.1989
133899 Громов   Дмитрий   Сергеевич     17.06.1987
126697 Громов   Илья      Сергеевич     23.03.1984
122240 Громов   Илья      Сергеевич     01.01.1984
142202 Громов   Роман     Геннадьевич   02.12.1988
```

5.5. Агрегирование данных

Многие запросы к базе данных не нуждаются в той степени детализации, которую обеспечивают ранее рассмотренные SQL-запросы. Например, во всех запросах, перечисленных далее, требуется узнать всего одно или несколько значений, которые подытоживают информацию, содержащуюся в базе данных.

- Сколько поставщиков поставляет морковь?
- Сколько людей заказало "Салат летний" на завтра?
- Какова средняя трудоемкость приготовления блюд, включенных в меню?
- Какова средняя стоимость заказанных блюд на каждом месте в столовой?
- Какова средняя стоимость всех заказанных блюд?
- Сколько строк в таблице `Состав`?
- Какие фамилии чаще всего встречаются в таблице `н_люди`?
- Сколько человек имеют имя Александр?
- Сколько неуспевающих студентов в каждой из групп?
- Сколько неуспевающих студентов на каждом факультете?
- Сколько неуспевающих студентов в университете?

SQL-запросы такого типа можно создавать с помощью агрегатных функций и фраз `GROUP BY` и `HAVING`.

5.5.1 Агрегатные SQL-функции

В SQL существует ряд специальных стандартных функций (агрегатных SQL-функций). Кроме специального случая `COUNT(*)` каждая из этих функций оперирует совокупностью значений столбца некоторой таблицы и создает единственное значение, определяемое так:

- `COUNT` — число значений в столбце;
- `SUM` — сумма значений в столбце;
- `AVG` — среднее значение в столбце;
- `MIN` — самое малое значение в столбце;
- `MAX` — самое большое значение в столбце.

Для функций `SUM` и `AVG` рассматриваемый столбец должен содержать числовые значения.

Следует отметить, что здесь столбец — это столбец виртуальной таблицы, в которой могут содержаться данные не только из столбца базовой таблицы, но и данные, полученные путем функционального преобразования и (или) связывания символами арифметических операций значений из одного или нескольких столбцов. При этом выражение, определяющее столбец такой таблицы, может быть сколь угодно сложным, но не должно содержать агрегатных SQL-функций (вложенность таких функций не допускается). Однако из агрегатных SQL-функций можно составлять любые выражения.

Аргументу всех функций, кроме `COUNT(*)`, может предшествовать ключевое слово `DISTINCT` (различный), указывающее, что избыточные дублирующие значения должны быть исключены перед тем, как будет применяться функция. Специальная же функция `COUNT(*)` служит для подсчета всех без исключения строк в таблице (включая дубликаты).

5.5.2. Функции без использования фразы *GROUP BY*

Если не используется фраза `GROUP BY`, то в перечень `отбираемых_элементов SELECT` можно включать лишь агрегатные SQL-функции или выражения, содержащие такие функции. Другими словами, нельзя иметь в списке столбцы, не являющиеся аргументами агрегатных SQL-функций.

Например, выдать данные о массе лука (`Код_продукта=10`), проданного поставщиками, и указать количество этих поставщиков:

```
SELECT SUM(К_во), COUNT(К_во)
FROM   Поставки
WHERE  Код_продукта=10;
```

Результат будет выглядеть так:

```
SUM(K_BO) COUNT(K_BO)
--------- -----------
      220           2
```

Если бы для вывода в результат еще и номера продукта был сформирован запрос

```
SELECT Код_продукта, SUM(К_во), COUNT(К_во)
FROM   Поставки
WHERE  Код_продукта=10;
```

то было бы получено сообщение об ошибке (на Oracle — "ORA-00937: групповая функция не является одногруппной"). Это связано с тем, что агрегатная

SQL-функция создает единственное значение из множества значений столбца-аргумента, а для "свободного" столбца должно быть выдано все множество его значений. Без специального указания (оно задается фразой `GROUP BY`) SQL не будет выяснять, одинаковы значения этого множества (как в данном примере, где `Код_продукта=10`) или различны (как было бы при отсутствии фразы `WHERE`). Поэтому подобный запрос отвергается системой.

Правда, никто не запрещает дать запрос:

```
SELECT 'Кол-во лука =', SUM(К_во), COUNT(К_во)
FROM   Поставки
WHERE  Код_продукта=10;
```

и получить:

```
'КОЛ-ВО ЛУКА=' SUM(К_ВО) COUNT(К_ВО)
-------------- --------- -----------
Кол-во лука =         220           2
```

Однако если нам все же хочется увидеть в результате (а не только в тексте запроса) значение кода продукта, то можно дать такой "смешной" запрос:

```
SELECT MAX(Код_продукта), SUM(К_во), COUNT(К_во)
FROM   Поставки
WHERE  Код_продукта=10;
```

основанный на том, что при формировании списка вывода в него попадут строки с кодом продукта, разным 10, и, следовательно, максимальное (или минимальное) значение этого кода будет равно 10:

```
MAX(КОД_ПРОДУКТА)  SUM(К_ВО) COUNT(К_ВО)
----------------- ---------- -----------
               10        220           2
```

Отметим также, что в столбце-аргументе перед применением любой функции, кроме `COUNT(*)`, исключаются все неопределенные значения. Если оказывается, что аргумент — пустое множество, функция `COUNT` принимает значение 0, а остальные — `NULL`.

Например, для получения суммы цен, средней цены, количества поставляемых продуктов и количества разных цен продуктов, проданных коопторгом `УРОЖАЙ` (`Код_поставщика=5`), а также для получения количества продуктов, которые могут поставляться этим коопторгом, можно дать запрос:

```
SELECT SUM(Цена), AVG(Цена), COUNT(Цена),
       COUNT(DISTINCT  Цена), COUNT(*)
FROM   Поставки
WHERE  Код_поставщика=5;
```

и получить:

```
SUM(ЦЕНА)  AVG(ЦЕНА) COUNT(ЦЕНА) COUNT(DISTINCTЦЕНА)   COUNT(*)
---------- ---------- ----------- -------------------- ----------
       6,2       1,24           5                    4          5
```

То в другом примере, где надо узнать "Сколько поставлено судака и сколько поставщиков его поставляют?":

```
SELECT SUM(К_во), COUNT(К_во)
FROM   Поставки
WHERE  Код_продукта=2;
```

будет получен ответ:

```
SUM(K_BO) COUNT(K_BO)
--------- -----------
                    0
```

Так как ни один из поставщиков не поставляет судака, то в этом результате `COUNT(K_BO)` оказалось равным нулю, а `SUM(K_BO)` — неопределенному значению (`NULL`), которое ни в стандарте, ни в Oracle не имеет видимого изображения.

Наконец, попробуем получить сумму массы поставленного лука с его средней ценой:

```
SELECT (SUM(К_во) + AVG(Цена))
FROM   Поставки
WHERE  Код_продукта=10;
```

Результат

```
(SUM(K_BO)+AVG(ЦЕНА))
---------------------
                220,6
```

Система с легкостью складывает два запрошенных числовых значения ("сапоги с яичницей"), не задумываясь о несуразности такого действия. Об этом должен заботиться пользователь, формирующий запрос.

В завершение данного раздела сформируем запрос на получение статистических данных о людях, хранящихся в таблице `Н_ЛЮДИ` базы данных "UCHEBA":

```
SELECT COUNT(*) людей,
       COUNT(DISTINCT фамилия) фамилий,
       COUNT(DISTINCT имя) имен,
       COUNT(DISTINCT отчество) отчеств
FROM н_люди;
```

при выполнении которого будет строка

```
ЛЮДЕЙ ФАМИЛИЙ ИМЕН ОТЧЕСТВ
----- ------- ---- -------
30331   14524  868    1296
```

содержащая общее число людей, хранящихся в таблице, а также число различных фамилий, имен и отчеств у этих людей.

5.5.3. Фраза *GROUP BY*

Мы показали, как можно вычислить массу определенного продукта, поставляемого поставщиками. Предположим, что теперь требуется вычислить общую массу каждого из продуктов, поставляемых в настоящее время поставщиками. Это можно легко сделать с помощью предложения:

```
SELECT Код_продукта, SUM(К_во)
FROM   Поставки
GROUP  BY Код_продукта;
```

Его результат имеет вид:

```
КОД_ПРОДУКТА  SUM(К_ВО)
------------ ----------
           1        370
           3        250
           4        100
           5        170
           6        220
           7        200
           8        150
          10        220
          11        150
          12         30
          13        190
          14         70
          15        370
          16        250
          17         50
```

Фраза `GROUP BY` (группировать по) инициирует перекомпоновку строк таблицы по группам, каждая из которых имеет одинаковые значения в столбце, указанном в `GROUP BY`. В рассматриваемом примере строки таблицы `Поставки` группируются так, что в одной группе содержатся все строки для продукта

с Код_продукта=1, в другой для продукта с Код_продукта=3 (строки с Код_продукта=2 нет, так как судак не поставляется ни одним из поставщиков) и т. д. Затем к каждой группе применяется фраза SELECT. Каждое выражение в этой фразе должно принимать единственное значение для группы, т. е. оно может быть либо:

- значением столбца, указанного в GROUP BY;
- арифметическим выражением, включающим это значение;
- константой;
- одной из SQL-функций, которая оперирует всеми значениями столбца в группе и сводит эти значения к единственному значению (например, к сумме, как в нашем запросе).

Определим теперь среднюю стоимость одного килограмма каждого из продуктов, находящихся в таблице поставки, их количество и среднюю цену этих продуктов:

```
SELECT Код_продукта, ROUND(SUM(цена*к_во)/SUM(К_во),2) Средняя, SUM(К_во)
Всего, AVG(Цена)
FROM   Поставки
GROUP BY Код_продукта;
```

Результат имеет вид:

```
КОД_ПРОДУКТА    СРЕДНЯЯ      ВСЕГО  AVG(ЦЕНА)
------------ ---------- ---------- ----------
           1       3,71        370        3,9
           3          4        250          4
           4       2,52        100       2,52
           5       1,88        170        1,9
           6       2,71        220        2,9
           7        0,4        200        0,4
           8          1        150          1
          10       0,58        220        0,6
          11       1,17        150       1,25
          12       2,67         30       2,75
          13       0,95        190       1,04
          14        0,5         70        0,5
          15       1,73        370       1,75
          16       0,95        250       0,97
          17        4,5         50        4,5
```

Для того чтобы средняя стоимость определялась с точностью до копеек, в запросе использовалась функция ROUND(n,[m]), в которой n — округляемое

числовое значение, а m — число десятичных знаков после запятой (точнее см. в *разд. 4.6*).

Обратите внимание, что ряд значений средней стоимости не совпадает со средней ценой!

Отметим также, что фраза GROUP BY не предполагает ORDER BY (хотя в рассмотренных ранее примерах результаты упорядочены). Чтобы гарантировать упорядочение по Код_продукта результата рассматриваемого примера, следует дать запрос:

```
SELECT код_продукта, SUM(цена*к_во)/SUM(к_во) Средняя, AVG(цена)
FROM   поставки
GROUP BY код_продукта;
ORDER BY Код_продукта;
```

Наконец, отметим, что строки таблицы можно группировать по любой комбинации ее столбцов. Пример такой группировки будет рассмотрен в следующем разделе.

5.5.4. Использование фразы *HAVING*

Фраза HAVING играет такую же роль для групп, что и фраза WHERE для строк: она используется для исключения групп точно так же, как WHERE используется для исключения строк. Эта фраза включается в предложение лишь при наличии фразы GROUP BY, а выражение в HAVING должно принимать единственное значение для группы.

Например, выдать количество людей с совпадающими фамилиями, именами и отчествами. В список включить только тех, которые имеют более трех "полных тезок".

```
SELECT фамилия, имя, отчество, COUNT(*)
FROM н_люди
GROUP BY фамилия, имя, отчество
HAVING COUNT(*) > 3
ORDER BY фамилия, имя, отчество;
```

Оказалось, что в таблице Н_ЛЮДИ базы данных "UCHEBA" имеется девять групп лиц, имеющих более трех "полных тезок", а Смирновых Андреев Сергеевичей — аж восемь штук.

```
ФАМИЛИЯ   ИМЯ       ОТЧЕСТВО      COUNT(*)
--------- --------- ------------- --------
Иванов    Дмитрий   Николаевич           4
Иванов    Дмитрий   Сергеевич            4
```

```
Иванов     Сергей     Александрович        5
Кузнецов   Сергей     Владимирович         4
Кузнецова  Мария      Александровна        4
Михайлов   Андрей     Александрович        5
Смирнов    Александр  Сергеевич            8
Смирнов    Сергей     Владимирович         4
Федоров    Алексей    Владимирович         4
```

В *разд. 6.5* можно познакомиться с более содержательным примером использования этой фразы.

5.6. Иерархические запросы

Если информация в таблице представлена, например, в виде генеалогического дерева (родственные связи в виде дерева, в котором у корня расположен родоначальник, а на ветвях дерева — различные линии его потомков), то хотелось бы иметь возможность получать ее наглядное представление достаточно простым способом.

В синтаксисе предложения `SELECT` (см. *разд. 5.1*) существует фраза:

```
[[ START WITH условие ] CONNECT BY [PRIOR] условие ]
```

где `START WITH` задает строки, которые будут выполнять в результирующем наборе роль родительских (корневых). Условие в этой фразе не должно содержать подзапросов. Если фраза `START WITH` не задана, то все строки будут корневыми.

`CONNECT BY` определяет условие соотношения между родительскими строками и их строками-потомками (дочерними строками). Условие в этой фразе не должно содержать подзапросов.

`PRIOR` используется для указания родительских строк вместо строк-потомков.

В иерархических запросах псевдостолбец `LEVEL` используется для указания корневой точки (1), точек-первичных потомков (2), точек-вторичных потомков (3) и т. д.

В этих запросах нельзя использовать фразы `ORDER BY` и `GROUP BY`. Для сортировки одноранговых потомков одной родительской таблицы можно использовать фразу `ORDER SIBLINGS BY`.

Для примера создадим иерархический запрос для таблицы `H_ОТДЕЛЫ`, описанной в *разд. 2.2* (см. табл. 2.1).

```
SELECT RPAD(' ', (LEVEL-1)*4)||TO_CHAR(о.ид)||' '||
                              о.имя_в_имин_падеже Отделы
```

```
FROM н_отделы о
START WITH о.отд_ид IS NULL
CONNECT BY PRIOR о.ид = о.отд_ид
ORDER SIBLINGS BY о.ид;
```

Результат этого запроса будет иметь вид:

```
ОТДЕЛЫ
--------------------------------------------------------------------------------
777 Санкт-Петербургский государственный университет информационных техно-
логий, механики и оптики
. . .
    701 факультет оптико-информационных систем и технологий
        201 кафедра оптико-электронных приборов и систем
        215 кафедра экологического приборостроения и мониторинга
        304 кафедра прикладной и компьютерной оптики
        305 кафедра информ.-измерительн. систем оптического приборостр-я
        307 кафедра оптических технологий
        308 кафедра компьютеризации и проектирования оптических приборов
        467 межкафедральная лаборатория компьютерной оптики ОФ
        512 кафедра системотехники оптических приб. и комплексов
        515 ВКЦГОИ
        601 деканат факультета оптико-информационных систем и технологий
    702 факультет инженерно-физический
        202 кафедра электроники
        203 кафедра лазерной техники и биомедицинской оптики
        205 кафедра компьютерной теплофизики и энергофизич. мониторинга
        207 кафедра физики и техники оптической связи
        211 кафедра твердотельной оптоэлектроники
        216 кафедра мощных технологических лазеров
        310 кафедра лазерных технологий и экологического приборостроения
        440 межкафедральный вычислительный класс ИФФ
        602 деканат инженерно-физического факультета
    703 факультет компьютерных технологий и управления
        101 кафедра систем управления и информатики
        102 кафедра вычислительной техники
        108 кафедра проектирования компьютерных систем
        109 кафедра информационно-навигационных систем
        111 кафедра информатики и прикладной математики
        116 кафедра безопасных информационных технологий
        208 кафедра электротехники и прецизионных эл/механических систем
```

```
        511 кафедра МПБЭВА
        603 деканат факультета компьютерных технологий и управления
    704 факультет естественнонаучный
        106 кафедра теоретической и прикладной механики
        209 кафедра высшей математики
        210 кафедра физики
        217 кафедра технологии профессионального обучения
        218 кафедра математического моделирования
        468 межкафедральный компьютерный класс ЕНФ
        604 деканат естественнонаучного факультета
    705 факультет точной механики и технологий
        103 кафедра измерительных технологий и компьютерной томографии
. . .
```

Глава 6

Запросы с использованием нескольких таблиц

6.1. О средствах одновременной работы с множеством таблиц

Затрагивая во второй главе вопросы проектирования баз данных, мы выяснили, что базы данных — это множество взаимосвязанных сущностей или отношений (таблиц) в терминологии реляционных СУБД. При проектировании стремятся создавать таблицы, в каждой из которых содержалась бы информация об одном и только об одном типе сущностей. Это облегчает модификацию базы данных и поддержание ее целостности. Но такой подход тяжело усваивается начинающими проектировщиками, которые пытаются привязать проект к будущим приложениям и так организовать таблицы, чтобы в каждой из них хранилось все необходимое для реализации возможных запросов. Типичен вопрос: как же получить сведения о том, где купить продукты для приготовления того или иного блюда и определить его калорийность и стоимость, если нужные данные "рассыпаны" по семи различным таблицам? Не лучше ли иметь одну большую таблицу, содержащую все сведения базы данных "COOK"?

Даже при отсутствии средств одновременного доступа ко многим таблицам нежелателен проект, в котором информация о многих типах сущностей перемешана в одной таблице. SQL же обладает великолепным механизмом для одновременной или последовательной обработки данных из нескольких взаимосвязанных таблиц. В нем реализованы возможности "соединять" или "объединять" несколько таблиц и так называемые "вложенные подзапросы". Например, чтобы получить перечень поставщиков продуктов, необходимых для приготовления сырников, возможен запрос

```
SELECT Продукты.Продукт, Поставки.Цена, Поставщики.Название,
       Поставщики.Статус
```

```
FROM   Продукты, Состав, Блюда, Поставки, Поставщики
WHERE  Продукты.Код_продукта = Состав.Код_продукта
AND    Состав.Код_блюда = Блюда.Код_блюда
AND    Поставки.Код_продукта = Состав.Код_продукта
AND    Поставки.Код_поставщика = Поставщики.Код_поставщика
AND    Блюда.Блюдо = 'Сырники'
AND    Поставки.Цена IS NOT NULL;
```

дающий следующий результат:

```
ПРОДУКТ     ЦЕНА НАЗВАНИЕ СТАТУС
-------- ------- -------- ----------
Яйца        1,80 ПОРТОС   кооператив
Яйца        2,00 КОРЮШКА  кооператив
Сметана     3,60 ПОРТОС   кооператив
Сметана     2,20 ОГУРЕЧИК ферма
Творог      1,00 ОГУРЕЧИК ферма
Мука        0,50 УРОЖАЙ   коопторг
Сахар       0,94 ТУЛЬСКИЙ универсам
Сахар       1,00 УРОЖАЙ   коопторг
```

Он получен следующим образом: СУБД последовательно формирует строки декартова произведения таблиц (см. *разд. 3.3.2*), перечисленных во фразе `FROM`, проверяет, удовлетворяют ли данные сформированной строки условиям фразы `WHERE`, и если удовлетворяют, то включает в ответ на запрос те ее поля, которые перечислены во фразе `SELECT`.

Очевидно, что с помощью соединения несложно сформировать запрос на обработку данных из нескольких таблиц. Кроме того, в такой запрос можно включить любые части предложения `SELECT`, рассмотренные в *главе 5* (выражения с использованием функций, группирование с отбором указанных групп и упорядочением полученного результата). Следовательно, соединения позволяют обрабатывать множество взаимосвязанных таблиц как единую таблицу, в которой перемешана информация о многих типах сущностей. Поэтому начинающий проектировщик базы данных может спокойно создавать маленькие нормализованные таблицы, так как он всегда может получить из них любую "большую" таблицу.

Кроме механизма соединений в SQL есть механизм вложенных подзапросов, позволяющий объединить несколько простых запросов в едином предложении `SELECT`. Иными словами, вложенный подзапрос — это уже знакомый нам подзапрос (с небольшими ограничениями), который вложен в `WHERE`-фразу другого вложенного подзапроса или `WHERE`-фразу основного запроса.

Для иллюстрации вложенного подзапроса вернемся к предыдущему примеру и попробуем получить перечень тех поставщиков продуктов для сырников, которые поставляют нужные продукты за минимальную цену.

```
SELECT Продукты.Продукт, Поставки.Цена, Поставщики.Название,
       Поставщики.Статус
FROM   Продукты, Состав, Блюда, Поставки, Поставщики
WHERE  Продукты.Код_продукта = Состав.Код_продукта
AND    Состав.Код_блюда = Блюда.Код_блюда
AND    Поставки.Код_продукта = Состав.Код_продукта
AND    Поставки.Код_поставщика = Поставщики.Код_поставщика
AND    Блюда.Блюдо = 'Сырники'
AND    Поставки.Цена = (SELECT MIN(Цена)
                        FROM   Поставки X
                        WHERE X.Код_продукта =
                              Поставки.Код_продукта );
```

Результат запроса имеет вид:

```
ПРОДУКТ     ЦЕНА НАЗВАНИЕ СТАТУС
-------- ------- -------- ----------
Яйца        1,80 ПОРТОС   кооператив
Сметана     2,20 ОГУРЕЧИК ферма
Творог      1,00 ОГУРЕЧИК ферма
Мука        0,50 УРОЖАЙ   коопторг
Сахар       0,94 ТУЛЬСКИЙ универсам
```

Здесь с помощью подзапроса, размещенного в трех последних строках запроса, описывается процесс определения минимальной цены каждого продукта для сырников и поиск поставщика, предлагающего этот продукт за такую цену. Механизм реализации подзапросов будет подробно описан в *разд. 6.3*. Там же будет рассмотрено, как и для чего вводится псевдоним `X` для имени таблицы `Поставки`.

В заключение следует подчеркнуть, что во всех фразах, в которых упоминаются имена столбцов, (например, `SELECT`, `WHERE`, …) во избежание двусмысленности ссылки на эти столбцы или символ (`*`) могут (а иногда и должны) уточняться именем соответствующей таблицы (квалификатором), например, `Поставки.Код_поставщика`, `Блюда.*`, `Поставщики.Код_поставщика`, `Меню.*`, `Состав.Код_блюда` и т. п. Это особенно необходимо делать в тех случаях, когда в таблицах, описанных во фразе `FROM`, существуют столбцы с одинаковыми именами (например, `Код_блюда` в `Блюдах`, `Состав` и `Рецепты`, `Код_продукта` в `Состав`, `Продукты` и `Поставки`).

6.1.1. Использование фразы *JOIN*

Рассмотренный ранее способ, создания соединений с помощью фразы WHERE — является "старым" способом соединения. Начиная с SQL:1992 и Oracle 9i лучшим способом является использование фразы JOIN, которая позволяет перенести условия соединения таблиц внутрь фразы FROM, оставив во фразе WHERE лишь условия для отбора строк. Такое разделение условий упрощает понимание текста запроса.

Синтаксис FROM с использованием JOIN имеет вид:

```
FROM таблица [AS псевдоним] {CROSS JOIN |
{ [NATURAL] [тип_соединения] JOIN соединяемая_таблица [AS псевдоним]
    { ON условие_соединения1 [{AND|OR} условие_соединения2 [. . .] ]
    | USING (столбец1 [, . . .]) } }
[. . .]
```

Здесь FROM *таблица* — первая таблица или представление в соединении.

Ключевое слово NATURAL служит для указания того, что соединение таблиц должно проводиться по всем их столбцам с идентичными именами. Это слово позволяет исключить из запроса условия соединения, обычно оговариваемые фразами ON или USING. Запрос не будет выполнен, если этот вид соединения будет проводиться в таблицах, не содержащих столбцов с одинаковыми именами.

С помощью фразы [*тип_соединения*] JOIN *соединяемая_таблица* указывается тип соединения и вторая (и все последующие) таблицы в соединении. Для всех этих таблиц можно определить псевдонимы. Могут использоваться следующие типы соединений.

- CROSS JOIN — осуществляет полное перекрестное соединение двух таблиц. Каждая строка первой таблицы соединяется со всеми строками второй таблицы, что создает результирующий набор огромных размеров — декартово произведение таблиц. Этот же результат будет получен и в случае других типов соединения, если вы забыли прописать условия соединения.
- [INNER] JOIN — осуществляет соединение двух таблиц, где каждая строка первой таблицы соединяется только с теми строками второй таблицы, которые удовлетворяют условию соединения. Ключевое слово INNER (внутренний) можно опускать — этот тип предполагается по умолчанию.
- LEFT [OUTER] JOIN — указывает, что строки будут возвращаться из таблицы, находящейся слева от ключевого слова JOIN. Если строка, возвращаемая из левой таблицы, не имеет соответствующей строки в правой таблице, строка все равно извлекается. В этом случае в столбцах для значений

из правой таблицы будут установлены значения NULL. Рекомендуется везде, где это возможно, использовать левосторонние внешние соединения (LEFT OUTER), чтобы не смешивать левосторонние и правосторонние соединения. Здесь и далее ключевое слово OUTER (внешний) можно опускать — этот тип для LEFT, RIGHT и FULL JOIN предполагается по умолчанию.

- RIGHT [OUTER] JOIN — схожа с LEFT OUTER, но строки будут возвращаться из таблицы, находящейся справа от ключевого слова JOIN.
- FULL [OUTER] JOIN — указывает, что возвращаться будут все строки из обеих таблиц независимо от того, совпадают ли строки в таблицах. Всем столбцам, для которых нет соответствующих значений в соединенной таблице, присваиваются значения NULL.
- UNION JOIN — указывает, что возвращаться будут все столбцы в обеих таблицах и все строки каждого столбца. Всем столбцам, для которых нет соответствующих значений в соединенной таблице, присваиваются значения NULL.

ON *условие_соединения* имеет следующий синтаксис:

```
FROM имя_таблицы1
JOIN имя_таблицы2
   ON имя_таблицы1.столбец1 = имя_таблицы2.столбец2
     [ {AND| OR} имя_таблицы1.столбец3 = имя_таблицы2.столбец4]
     [. . .]
JOIN имя_таблицы3
   ON имя_таблицы1.столбец5 = имя_таблицы3.столбец6
     [ {AND| OR} имя_таблицы1.столбец7 = имя_таблицы3.столбец8]
     [. . .]
[JOIN. . .]
```

Если столбцы, по которым производится соединение таблиц, имеют совпадающие имена, то вместо условия с ON можно написать USING (*столбец1* [, . . .]).

При использовании JOIN, первый запрос, рассмотренный в *разд. 6.1*, может быть записан так:

```
SELECT Продукты.Продукт, Поставки.Цена, Поставщики.Название,
       Поставщики.Статус
FROM   Продукты
  JOIN Состав ON Продукты.Код_продукта = Состав.Код_продукта
  JOIN Поставки ON Поставки.Код_продукта = Состав.Код_продукта
  JOIN Блюда ON Состав.Код_блюда = Блюда.Код_блюда
```

```
  JOIN Поставщики ON Поставки.Код_поставщика =
                     Поставщики.Код_поставщика
WHERE  Блюдо = 'Сырники'
   AND Цена IS NOT NULL;
```

или так:

```
SELECT Продукты.Продукт, Поставки.Цена, Поставщики.Название,
       Поставщики.Статус
FROM   Продукты
  JOIN Состав USING (Код_продукта)
  JOIN Поставки USING (Код_продукта)
  JOIN Блюда USING (Код_блюда)
  JOIN Поставщики USING (Код_поставщика)
WHERE  Блюдо = 'Сырники'
   AND Цена IS NOT NULL;
```

или вот так:

```
SELECT Продукты.Продукт, Поставки.Цена, Поставщики.Название,
       Поставщики.Статус
FROM   Продукты
NATURAL JOIN Состав
NATURAL JOIN Поставки
NATURAL JOIN Блюда
NATURAL JOIN Поставщики
WHERE  Блюдо = 'Сырники'
AND    Цена IS NOT NULL;
```

6.2. Запросы, использующие соединения

В данном разделе будут рассмотрены примеры реализации реляционных операций с помощью описанных ранее средств языка SQL.

6.2.1. Декартово произведение таблиц

В *разд. 3.3.2* показано, что соединения — это подмножества декартова произведения. Так как декартово произведение n таблиц — это таблица, содержащая все возможные строки r, такие, что r является сцеплением какой-либо строки из первой таблицы, строки из второй таблицы, ... и строки из n-й таблицы. Так как мы уже научились выделять с помощью `SELECT` любое под-

множество реляционной таблицы, то осталось лишь выяснить, можно ли с помощью SELECT получить декартово произведение.

Для получения декартова произведения нескольких таблиц можно:

1. Указать во фразе FROM перечень перемножаемых таблиц, а во фразе SELECT все их столбцы.
2. Воспользоваться фразой CROSS JOIN.

Так как количество строк декартова произведения равно произведению количества строк соединяемых таблиц, то "перемножим" маленькие таблицы Виды_блюд (табл. 3.2) и Трапезы (табл. 3.7):

```
SELECT Виды_блюд.*, Трапезы.*
FROM   Виды_блюд, Трапезы;
```

и получим таблицу, содержащую 5 × 3 = 15 строк:

```
КОД_ВИДА ВИД     КОД_ТРАПЕЗЫ ТРАПЕЗА
-------- ------- ----------- -------
       1 Закуска           1 Завтрак
       2 Суп               1 Завтрак
       3 Горячее           1 Завтрак
       4 Десерт            1 Завтрак
       5 Напиток           1 Завтрак
       1 Закуска           2 Обед
       2 Суп               2 Обед
       3 Горячее           2 Обед
       4 Десерт            2 Обед
       5 Напиток           2 Обед
       1 Закуска           3 Ужин
       2 Суп               3 Ужин
       3 Горячее           3 Ужин
       4 Десерт            3 Ужин
       5 Напиток           3 Ужин
```

В рекомендуемой нотации с использованием JOIN аналогичный запрос имеет вид:

```
SELECT Виды_блюд.*, Трапезы.*
FROM   Виды_блюд
  CROSS JOIN Трапезы;
```

В другом примере, где перемножаются таблицы Меню, Трапезы, Виды_блюд, Блюда:

```
SELECT Меню.*, Трапезы.*, Виды_блюд.*, Блюда.*
FROM   Меню, Трапезы, Виды_блюд, Блюда;
```

образуется таблица, содержащая 21 × 3 × 5 × 33 = 10 395 строк. Далее приведена выборка из 39 первых строк этой таблицы. В ней для уменьшения ширины изъят столбец ДАТА таблицы МЕНЮ, содержащий одинаковые значения 15.05.1989.

МЕНЮ			ТРАПЕЗЫ		ВИДЫ_БЛЮД		БЛЮДА					
СТРОКА	КОД_ТРАПЕЗЫ	КОД_БЛЮДА	КОД_ТРАПЕЗЫ	ТРАПЕЗА	КОД_ВИДА	ВИД	КОД_БЛЮДА	БЛЮДО	КОД_ВИДА	ОСНОВА	ВЫХОД	ТРУД	
1	1	3	1	Завтрак	1	Закуска	1	Салат летний	1	Овощи	200,0	3	
1	1	3	1	Завтрак	1	Закуска	2	Салат мясной	1	Мясо	200,0	4	
1	1	3	1	Завтрак	1	Закуска	3	Салат витаминный	1	Овощи	200,0	4	*
								. . .					
1	1	3	1	Завтрак	1	Закуска	12	Суп молочный	2	Молоко	500,0	3	
1	1	3	1	Завтрак	1	Закуска	13	Бастурма	3	Мясо	300,0	5	
								. . .					
1	1	3	1	Завтрак	1	Закуска	32	Кофе черный	5	Кофе	100,0	1	
1	1	3	1	Завтрак	1	Закуска	33	Кофе на молоке	5	Кофе	200,0	2	
2	1	6	1	Завтрак	1	Закуска	1	Салат летний	1	Овощи	200,0	3	
2	1	6	1	Завтрак	1	Закуска	2	Салат мясной	1	Мясо	200,0	4	
2	1	6	1	Завтрак	1	Закуска	3	Салат витаминный	1	Овощи	200,0	4	
2	1	6	1	Завтрак	1	Закуска	4	Салат рыбный	1	Рыба	200,0	4	
2	1	6	1	Завтрак	1	Закуска	5	Паштет из рыбы	1	Рыба	120,0	5	
2	1	6	1	Завтрак	1	Закуска	6	Мясо с гарниром	1	Мясо	250,0	3	*
								. . .					

При анализе этих строк мы нашли только две актуальных (отмечены символом "*"), в которых совпадают номера блюд таблиц Меню и Блюда. В осталь-

ных полная чепуха: к закускам отнесены супы и напитки, на завтрак предлагается незапланированный суп и т. д.

6.2.2. Эквисоединение таблиц

Если из декартова произведения убрать ненужные строки и столбцы, то можно получить актуальные таблицы, соответствующие любому из соединений.

Очевидно, что отбор актуальных строк обеспечивается вводом в запрос `WHERE` фразы, в которой устанавливается соответствие между:

- кодами трапез в таблицах `Меню` и `Трапезы`

  (`Меню.Код_трапезы = Трапезы.Код_трапезы`);
- кодами видов блюд в таблицах `Меню` и `Виды_блюд`

  (`Меню.Вид_блюда = Виды_блюд.Вид_блюда`),
- кодами блюд в таблицах `Меню` и `Блюда`

  (`Меню.Код_блюда = Блюда.Код_блюда`).

Такой скорректированный запрос, содержащий все столбцы,

```
SELECT Меню.*, Трапезы.*, Виды_блюд.*, Блюда.*
FROM   Меню, Трапезы, Виды_блюд, Блюда
WHERE  Меню.Код_блюда = Блюда.Код_блюда
  AND  Блюда.Код_вида = Виды_блюд.Код_вида
  AND  Меню.Код_трапезы = Трапезы.Код_трапезы;
```

позволит получить эквисоединение таблиц `Меню`, `Трапезы`, `Виды_блюд` и `Блюда`, содержащее всего 21 строку (столько же, сколько в таблице `Меню`):

СТРОКА	КОД_ТРАПЕЗЫ	КОД_БЛЮДА	КОД_ТРАПЕЗЫ	ТРАПЕЗА	КОД_ВИДА	ВИД	КОД_БЛЮДА	БЛЮДО	КОД_ВИДА	ОСНОВА	ВЫХОД	ТРУД
1	1	3	1	Завтрак	1	Закуска	3	Салат витаминный	1	Овощи	200,0	4
2	1	6	1	Завтрак	1	Закуска	6	Мясо с гарниром	1	Мясо	250,0	3
3	1	19	1	Завтрак	3	Горячее	19	Омлет с луком	3	Яйца	200,0	5
								. . .				

19	3	16	3	Ужин	3	Закуска	16	Драчена	3	Яйца	180,0	4
20	3	30	3	Ужин	5	Закуска	30	Компот	5	Фрукты	200,0	2
21	3	31	3	Ужин	5	Закуска	31	Молочный напиток	5	Молоко	200,0	2

Аналогичный результат может быть получен с помощью запросов, использующих рекомендуемую нотацию с фразой JOIN:

```
SELECT Меню.*, Трапезы.*, Виды_блюд.*, Блюда.*
FROM   Меню
 JOIN Блюда ON Меню.Код_блюда = Блюда.Код_блюда
 JOIN Виды_блюд ON Блюда.Код_вида = Виды_блюд.Код_вида
 JOIN Трапезы ON Меню.Код_трапезы = Трапезы.Код_трапезы;
```

6.2.3. Естественное соединение таблиц

Легко заметить, что в эквисоединение таблиц вошли дубликаты столбцов, по которым проводилось соединение (Код_трапезы, Код_вида и Код_блюда). Для исключения этих дубликатов можно создать естественное соединение тех же таблиц:

```
SELECT Строка, Код_трапезы, Код_блюда, Трапеза, Код_вида, Вид,
       Блюдо, Основа, Выход, Труд
FROM   Меню
 JOIN  Блюда ON Меню.Код_блюда = Блюда.Код_блюда
 JOIN  Виды_блюд ON Блюда.Код_вида = Виды_блюд.Код_вида
 JOIN  Трапезы ON Меню.Код_трапезы = Трапезы.Код_трапезы;
```

Реализация естественного соединения таблиц имеет вид

СТРОКА	КОД_ТРАПЕЗЫ	КОД_БЛЮДА	ТРАПЕЗА	КОД_ВИДА	ВИД	БЛЮДО	ОСНОВА	ВЫХОД	ТРУД
1	1	3	Завтрак	1	Закуска	Салат витаминный	Овощи	200,0	4
2	1	6	Завтрак	1	Закуска	Мясо с гарниром	Мясо	250,0	3
3	1	19	Завтрак	3	Горячее	Омлет с луком	Яйца	200,0	5
						. . .			

```
19   3   16  Ужин    3   Закуска Драчена           Яйца    180,0 4
20   3   30  Ужин    5   Закуска Компот            Фрукты  200,0 2
21   3   31  Ужин    5   Закуска Молочный напиток  Молоко  200,0 2
```

6.2.4. Композиция таблиц

Для исключения всех столбцов, по которым проводится соединение таблиц, надо создать композицию

```
SELECT Строка, Трапеза, Вид, Блюдо, Основа, Выход, Труд
FROM   Меню
 JOIN  Блюда ON Меню.Код_блюда = Блюда.Код_блюда
 JOIN  Виды_блюд ON Блюда.Код_вида = Виды_блюд.Код_вида
 JOIN  Трапезы ON Меню.Код_трапезы = Трапезы.Код_трапезы;
```

имеющую вид:

СТРОКА	ТРАПЕЗА	ВИД	БЛЮДО	ОСНОВА	ВЫХОД	ТРУД
1	Завтрак	Закуска	Салат витаминный	Овощи	200,0	4
2	Завтрак	Закуска	Мясо с гарниром	Мясо	250,0	3
3	Завтрак	Горячее	Омлет с луком	Яйца	200,0	5
			. . .			
19	Ужин	Закуска	Драчена	Яйца	180,0	4
20	Ужин	Закуска	Компот	Фрукты	200,0	2
21	Ужин	Закуска	Молочный напиток	Молоко	200,0	2

6.2.5. Тета-соединение таблиц

В базе данных "COOK" трудно подобрать несложный пример, иллюстрирующий тета-соединение таблиц. Поэтому сконструируем такой надуманный запрос:

```
SELECT Виды_блюд.*, Трапезы.*
FROM   Виды_блюд, Трапезы
WHERE  Вид > Трапеза;
```

Он позволяет выбрать из полученного в *разд. 6.2.1* декартова произведения таблиц `Виды_блюд` и `Трапезы` лишь те строки, в которых значение трапезы "меньше" (по алфавиту) значения вида блюда:

```
КОД_ВИДА ВИД     КОД_ТРАПЕЗЫ ТРАПЕЗА
-------- ------- ----------- -------
       1 Закуска           1 Завтрак
       2 Суп               1 Завтрак
       5 Напиток           1 Завтрак
       2 Суп               2 Обед
```

6.2.6. Соединение таблицы со своей копией

В ряде приложений возникает необходимость одновременной обработки данных какой-либо таблицы и одной или нескольких ее виртуальных копий, создаваемых на время выполнения запроса.

Например, при вводе новых людей в таблицу н_люди базы данных "UCHEBA" возможен повторный ввод данных о каком-либо человеке с присвоением ему второго идентификатора. Для выявления таких ошибок можно соединить таблицу н_люди с ее временной копией, приравняв значения всех одноименных столбцов этих таблиц, кроме столбцов с идентификатором. Для последних надо использовать условие неравенства значений.

Временную копию таблицы можно сформировать, указав имя псевдонима за именем таблицы во фразе FROM. Так, с помощью предложения

```
SELECT копия.ид, н_люди.ид, н_люди.фамилия, н_люди.имя,
       н_люди.отчество, н_люди.дата_рождения
FROM   н_люди, н_люди копия
WHERE  н_люди.фамилия = копия.фамилия
  AND  н_люди.имя = копия.имя
  AND  н_люди.отчество = копия.отчество
  AND  н_люди.дата_рождения = копия.дата_рождения
  AND  н_люди.ид <> копия.ид
ORDER BY н_люди.фамилия, н_люди.имя, н_люди.отчество;
```

или его новой нотации

```
SELECT копия.ид, н_люди.ид, н_люди.фамилия, н_люди.имя,
       н_люди.отчество, н_люди.дата_рождения
FROM   н_люди
 JOIN  н_люди копия ON н_люди.фамилия= копия.фамилия
                   AND н_люди.имя= копия.имя
                   AND н_люди.отчество= копия.отчество
                   AND н_люди.дата_рождения= копия.дата_рождения
WHERE  н_люди.ид <> копия.ид
ORDER BY н_люди.фамилия, н_люди.имя, н_люди.отчество;
```

получим 224 строки с искомыми людьми.

Рассмотрим выборку из 16 строк полученной таблицы:

```
КОПИЯ.ИД     ИД ФАМИЛИЯ     ИМЯ        ОТЧЕСТВО      ДАТА_РОЖДЕНИЯ
-------- ------ ----------- --------- ------------- -------------
  143878 145982 Марцуков    Алексей   Александрович 26.01.1987
  145982 143878 Марцуков    Алексей   Александрович 26.01.1987
  143880 146001 Мирошникова Катерина  Федоровна     20.09.1987
  143545 146001 Мирошникова Катерина  Федоровна     20.09.1987
  143545 143880 Мирошникова Катерина  Федоровна     20.09.1987
  146001 143880 Мирошникова Катерина  Федоровна     20.09.1987
  146001 143545 Мирошникова Катерина  Федоровна     20.09.1987
  143880 143545 Мирошникова Катерина  Федоровна     20.09.1987
  113191 142799 Михайлов    Андрей    Александрович 17.11.1962
  142588 142800 Михайлов    Андрей    Александрович 10.04.1989
  142800 142588 Михайлов    Андрей    Александрович 10.04.1989
  142799 113191 Михайлов    Андрей    Александрович 17.11.1962
  143881 146007 Михайлов    Дмитрий   Андреевич     06.02.1987
  146007 143881 Михайлов    Дмитрий   Андреевич     06.02.1987
  140476 143744 Михайлова   Анастасия Владимировна  06.12.1987
  143744 140476 Михайлова   Анастасия Владимировна  06.12.1987
```

Отметим, что в ней для каждого совпадающего сочетания фамилии, имени, отчества и даты рождения выведено не менее двух строк, которые содержат одинаковые идентификаторы из `н_люди` и `копия`, но находящиеся в разных столбцах (формально это разные строки). Исключить фактические дубликаты строк можно, например, убрав из запроса столбец `копия.ид` и добавив в него ключевое слово `DISTINCT`:

```
SELECT DISTINCT н_люди.ид, н_люди.фамилия, н_люди.имя,
       н_люди.отчество, н_люди.дата_рождения
FROM   н_люди
 JOIN  н_люди копия ON н_люди.фамилия= копия.фамилия
                   AND н_люди.имя= копия.имя
                   AND н_люди.отчество= копия.отчество
                   AND н_люди.дата_рождения= копия.дата_рождения
WHERE  н_люди.ид <> копия.ид
ORDER BY н_люди.фамилия, н_люди.имя, н_люди.отчество;
```

исключив из результата три "лишних" строки для Мирошниковой:

```
143878 Марцуков     Алексей    Александрович 26.01.1987
145982 Марцуков     Алексей    Александрович 26.01.1987
143545 Мирошникова  Катерина   Федоровна     20.09.1987
```

```
143880 Мирошникова  Катерина   Федоровна     20.09.1987
146001 Мирошникова  Катерина   Федоровна     20.09.1987
113191 Михайлов     Андрей     Александрович 17.11.1962
142799 Михайлов     Андрей     Александрович 17.11.1962
142800 Михайлов     Андрей     Александрович 10.04.1989
142588 Михайлов     Андрей     Александрович 10.04.1989
143881 Михайлов     Дмитрий    Андреевич     06.02.1987
146007 Михайлов     Дмитрий    Андреевич     06.02.1987
140476 Михайлова    Анастасия  Владимировна  06.12.1987
143744 Михайлова    Анастасия  Владимировна  06.12.1987
```

Тот же результат можно получить и без DISTINCT, использовав коррелированный вложенный подзапрос (см. *разд. 6.3.6*):

```
SELECT ид, фамилия, имя, отчество, дата_рождения
FROM н_люди
WHERE EXISTS (SELECT *
              FROM н_люди копия
              WHERE н_люди.фамилия = копия.фамилия
                AND н_люди.имя = копия.имя
                AND н_люди.отчество = копия.отчество
                AND н_люди.дата_рождения = копия.дата_рождения
                AND н_люди.ид <> копия.ид)
ORDER BY фамилия, имя, отчество;
```

Так как во внешнем запросе используется только одна таблица Н_ЛЮДИ, то в нем не нужно использовать квалификаторы перед именами столбцов.

6.2.7. Внешние соединения

До сих пор мы создавали соединения, в которых каждому связываемому значению в одной из таблиц всегда находилось соответствующее значение в другой. Такие соединения принято называть внутренними соединениями. А как быть, если строка одной из связываемых таблиц не имеет пары в другой. Например, если мы ввели в таблицу БЛЮДА несколько новых блюд (40 — Шашлык по-карски, 41 — Борщ, 42 — Щи русские, 43 — Рыба "Орли" и 44 — Икра из свеклы), но не успели (или забыли) ввести рецепты этих блюд, а сформировали запрос на вывод значений блюд с их рецептами:

```
SELECT код_блюда, блюдо, основа, код_вида,
       SUBSTR(рецепт,1,20),вариант
FROM   блюда
JOIN   рецепты USING (код_блюда)
```

```
ORDER BY код_блюда, вариант;
```

Результат выполнения этого запроса не содержит введенных нами блюд, так как в таблице РЕЦЕПТЫ нет строк с кодами блюд с 40 по 44.

```
 КОД_
БЛЮДА БЛЮДО            ОСНОВА КОД_ВИДА SUBSTR(РЕЦЕПТ,1,20)  ВАРИАНТ
----- ---------------- ------ -------- -------------------- -------
                                . . .
   21 Пудинг рисовый   Крупа         3 Готовую рисовую расс
   22 Вареники ленивые Молоко        3 В протертый творог п
   23 Помидоры с луком Овощи         3 Спассеровать на масл
   24 Суфле из творога Молоко        3 В протертый творог п
   25 Рулет с яблоками Фрукты        4 Очистить яблоки, раз
   26 Яблоки печеные   Фрукты        4 Не прорезая насквозь
   27 Суфле яблочное   Фрукты        4 Запеченные яблоки с
   28 Крем творожный   Молоко        4 Яйца размешать с сах
   29 «Утро»           Фрукты        5 Очищенную и промытую
   30 Компот           Фрукты        5 Яблоки очистить от к
   31 Молочный напиток Молоко        5 Яблоки натереть на т
   32 Кофе черный      Кофе          5 Вскипятить воду в ко       1
   32 Кофе черный      Кофе          5 Кофеварку или кастрю       2
   33 Кофе на молоке   Кофе          5 Сварить черный кофе,
```

Для того чтобы они появились следует воспользоваться не операцией JOIN, соответствующей по умолчанию [INNER] JOIN, а, например, операцией LEFT [OUTER] JOIN:

```
SELECT код_блюда, блюдо, основа, код_вида,
       SUBSTR(рецепт,1,20), вариант
FROM блюда
 LEFT JOIN рецепты USING (код_блюда)
ORDER BY код_блюда,вариант;
```

```
 КОД_
БЛЮДА БЛЮДО            ОСНОВА КОД_ВИДА SUBSTR(РЕЦЕПТ,1,20)  ВАРИАНТ
----- ---------------- ------ -------- -------------------- -------
                                . . .
   32 Кофе черный      Кофе          5 Вскипятить воду в ко       1
   32 Кофе черный      Кофе          5 Кофеварку или кастрю       2
   33 Кофе на молоке   Кофе          5 Сварить черный кофе,
   40 Шашлык по-карски Мясо          3
```

```
   41 Рыба "Орли"       Рыба          3
   42 Щи русские        Мясо          2
   43 Борщ              Мясо          4
   44 Икра из свеклы    Овощи         1
```

Нужно отметить, что в этих запросах для уменьшения длины выводимых строк использовалась функция SUBSTR (рецепт,1,20), которая, в данном случае, вырезает из текста рецепта 20 символов, начиная с первого (1). (С описанием этой функции можно познакомиться в *разд. 4.6*.)

Вот еще один пример внешнего соединения:

```
SELECT блюда.блюдо, продукты.продукт, состав.вес
FROM блюда
 FULL  JOIN  состав USING (КОД_БЛЮДА)
 FULL  JOIN  продукты USING (КОД_ПРОДУКТА);
```

который при соединении таблиц БЛЮДА, СОСТАВ и ПРОДУКТЫ позволяет вывести те значения блюд и продуктов, которые введены в соответствующие таблицы, но отсутствуют в таблице СОСТАВ:

```
БЛЮДО             ПРОДУКТ   ВЕС
----------------- -------- ----
. . .
Уха из судака     Судак     100
Уха из судака     Морковь    20
Уха из судака     Лук        20
Уха из судака     Масло       5
Уха из судака     Зелень      2
Яблоки печеные    Яблоки    150
Яблоки печеные    Сахар      20
Яблоки печеные    Масло       2
Шашлык по-карски
Борщ
Щи русские
Рыба "Орли"
Икра из свеклы
                  Кура
                  Тыква
                  Свекла
```

6.3. Вложенные подзапросы

6.3.1. Виды вложенных подзапросов

Вложенный подзапрос (SUBQUERY) — это предложение SELECT, заключенное в круглые скобки и вложенное в WHERE (HAVING)-фразу другого предложения SELECT или иных предложений, использующих WHERE-фразу. Вложенный подзапрос может содержать в своей WHERE (HAVING)-фразе другой вложенный подзапрос и т. д. Нетрудно догадаться, что вложенный подзапрос создан для того, чтобы при отборе строк таблицы, сформированной основным запросом, можно было использовать данные из других таблиц (например, при отборе блюд для меню использовать данные о наличии продуктов в кладовой пансионата).

Существуют простые и коррелированные вложенные подзапросы. Они включаются в WHERE (HAVING)-фразу с помощью условий IN, EXISTS или одного из условий сравнения (= | <> | < | <= | > | >=). Простые вложенные подзапросы обрабатываются системой "снизу вверх". Первым обрабатывается вложенный подзапрос самого нижнего уровня. Множество значений, полученное в результате его выполнения, используется при реализации подзапроса более высокого уровня и т. д.

Запросы с коррелированными вложенными подзапросами обрабатываются системой в обратном порядке. Сначала выбирается первая строка рабочей таблицы, сформированной основным запросом, и из нее выбираются значения тех столбцов, которые используются во вложенном подзапросе (вложенных подзапросах). Если эти значения удовлетворяют условиям вложенного подзапроса, то выбранная строка включается в результат. Затем выбирается вторая строка и т. д., пока в результат не будут включены все строки, удовлетворяющие вложенному подзапросу (последовательности вложенных подзапросов).

Следует отметить, что SQL обладает большой избыточностью в том смысле, что он часто предоставляет несколько различных способов формулировки одного и того же запроса. Поэтому во многих примерах данной главы будут использованы уже знакомые нам по предыдущей главе концептуальные формулировки запросов. И несмотря на то, что часть из них успешнее реализуется с помощью соединений, здесь все же будут приведены их варианты с использованием вложенных подзапросов. Это связано с необходимостью детального знакомства с созданием и принципом выполнения вложенных подзапросов, так как существует немало задач (особенно на удаление и изменение

данных), которые не могут быть реализованы другим способом. Кроме того, разные формулировки одного и того же запроса требуют для своего выполнения различные ресурсы памяти и могут значительно отличаться по времени реализации в разных СУБД.

6.3.2. Простые вложенные подзапросы

Простые вложенные подзапросы используются для представления множества значений, исследование которых должно осуществляться в каком-либо предикате IN, что иллюстрируется в следующем примере: выдать название и статус поставщиков продукта с номером 11, т. е. помидоров.

```
SELECT Название, Статус
FROM   Поставщики
WHERE  Код_поставщика IN
       (SELECT Код_поставщика
        FROM   Поставки
        WHERE  Код_продукта = 11);
```

Результат такого запроса имеет вид:

```
НАЗВАНИЕ СТАТУС
-------- ----------
СЫТНЫЙ   рынок
КОРЮШКА  кооператив
```

Как уже отмечалось в *разд. 6.3.1*, при обработке полного запроса система выполняет прежде всего вложенный подзапрос. Этот подзапрос выдает множество номеров поставщиков, которые поставляют продукт с кодом, равным 11, а именно множество (1, 8). Поэтому первоначальный запрос эквивалентен такому простому запросу:

```
SELECT Название,  Статус
FROM   Поставщики
WHERE  Код_поставщика IN (1, 8);
```

Подзапрос с несколькими уровнями вложенности можно проиллюстрировать на том же примере. Пусть требуется узнать не поставщиков продукта 11, как это делалось в предыдущих запросах, а поставщиков помидоров, являющихся продуктом с номером 11. Для этого можно дать запрос

```
SELECT Название, Статус
FROM   Поставщики
WHERE  Код_поставщика IN
       (SELECT Код_поставщика
```

```
        FROM Поставки
        WHERE Код_продукта IN
              (SELECT Код_продукта
               FROM Продукты
               WHERE Продукт = 'Помидоры'));
```

В данном случае результатом самого внутреннего подзапроса является только одно значение (11). Как уже было показано ранее, подзапрос следующего уровня в свою очередь дает в результате множество (1, 8). Последний, самый внешний `SELECT`, вычисляет приведенный ранее окончательный результат. Вообще допускается любая глубина вложенности подзапросов.

Тот же результат можно получить с помощью соединения

```
SELECT Название, Статус
FROM   Поставщики
 JOIN  Поставки USING (Код_Поставщика)
 JOIN  Продукты USING (Код_Продукта)
WHERE  Продукт = 'Помидоры';
```

При выполнении этого компактного запроса система должна одновременно обрабатывать данные из трех таблиц, тогда как в предыдущем примере эти таблицы обрабатываются поочередно. Естественно, что для их реализации требуются различные ресурсы памяти и времени, однако этого невозможно ощутить при работе с ограниченным объемом данных в иллюстративной базе "COOK".

6.3.3. Использование одной и той же таблицы во внешнем и вложенном подзапросе

Выдать номера поставщиков, которые поставляют хотя бы один продукт, поставляемый поставщиком 6.

```
SELECT DISTINCT Код_поставщика
FROM   Поставки
WHERE  Код_продукта IN
        (SELECT Код_продукта
         FROM   Поставки
         WHERE  Код_поставщика = 6);
```

Отметим, что ссылка на `Поставки` во вложенном подзапросе означает не то же самое, что ссылка на `Поставки` во внешнем запросе. В действительности,

два имени Поставки описывают различные значения. Чтобы этот факт стал явным, полезно использовать псевдонимы, например, А и Б:

```
SELECT DISTINCT А.Код_поставщика
FROM   Поставки А
WHERE  А.Код_продукта IN
          (SELECT Б.Код_продукта
           FROM Поставки Б
           WHERE Б.Код_поставщика = 6);
```

Здесь А и Б — произвольные псевдонимы таблицы Поставки, определяемые во фразе FROM и используемые как явные уточнители во фразах SELECT и WHERE. Напомним, что псевдонимы определены лишь в пределах одного запроса.

6.3.4. Вложенный подзапрос с оператором сравнения, отличным от *IN*

Выдать номера поставщиков, находящихся в том же городе, что и поставщик с номером 6.

```
SELECT А.Код_поставщика
FROM   Поставщики А
WHERE  А.Город =
        (SELECT Б.Город
         FROM Поставщики Б
         WHERE Б.Код_поставщика = 6 );
```

Результат:

```
КОД_ПОСТАВЩИКА
--------------
             1
             4
             6
```

В подобных запросах можно использовать и другие операторы сравнения (<>, <=, <, >= или >), однако, если вложенный подзапрос возвращает более одного значения и не используется оператор IN, будет возникать ошибка.

6.3.5. Коррелированные вложенные подзапросы

Выдать название и статус поставщиков продукта с номером 11.

```
SELECT Название, Статус
```

```
FROM   Поставщики
WHERE  11 IN
         (SELECT Код_продукта
          FROM    Поставки
          WHERE   Код_поставщика = Поставщики.Код_поставщика);
```

Такой подзапрос отличается от рассмотренного в *разд. 6.3.2* тем, что вложенный подзапрос не может быть обработан прежде, чем будет обрабатываться внешний запрос. Это связано с тем, что вложенный подзапрос зависит от значения `Поставщики.Код_поставщика`, а оно изменяется по мере того, как система проверяет различные строки таблицы `Поставщики`. Следовательно, с концептуальной точки зрения обработка осуществляется таким образом:

1. Система проверяет первую строку таблицы `Поставщики`. Предположим, что это строка поставщика с номером 1. Тогда значение `Поставщики.Код_поставщика` в данный момент имеет значение, равное 1, и система обрабатывает внутренний запрос

   ```
   (SELECT Код_продукта
   FROM    Поставки
   WHERE   Код_поставщика = 1);
   ```

   получая в результате множество (11, 12, 15). Теперь система может завершить обработку для поставщика с номером 1. Выборка значений `Название` и `Статус` для `Код_поставщика=1` (СЫТНЫЙ и рынок) будет проведена тогда и только тогда, когда `Код_продукта=11` будет принадлежать этому множеству, что, очевидно, справедливо.

2. Далее система будет повторять обработку такого рода для следующего поставщика и т. д. до тех пор, пока не будут рассмотрены все строки таблицы `Поставщики`.

Подобные подзапросы называются *коррелированными*, так как их результат зависит от значений, определенных во внешнем подзапросе. Обработка коррелированного подзапроса, следовательно, должна повторяться для каждого значения, извлекаемого из внешнего подзапроса, а не выполняться раз и навсегда.

Рассмотрим пример использования одной и той же таблицы во внешнем подзапросе и коррелированном вложенном подзапросе. Выдать номера всех продуктов, поставляемых только одним поставщиком.

```
SELECT DISTINCT A.Код_продукта
FROM   Поставки A
WHERE  A.Код_продукта NOT IN
       (SELECT Б.Код_продукта
```

```
        FROM   Поставки Б
        WHERE Б.Код_поставщика <> А.Код_поставщика);
```

Результат:

```
КОД_ПРОДУКТА
------------
           3
           7
           8
          14
          17
```

Действие этого запроса можно пояснить следующим образом: "Поочередно для каждой строки таблицы Поставки, скажем А, выделить значение номера продукта (Код_продукта), если и только если это значение не входит в некоторую строку, скажем, Б, той же таблицы, а значение столбца номер поставщика (Код_поставщика) в строке Б не равно его значению в строке А".

Отметим, что в этой формулировке должен быть использован по крайней мере один псевдоним — либо А, либо Б.

6.3.6. Запросы, использующие *EXISTS*

Квантор EXISTS (существует) — понятие, заимствованное из формальной логики. В языке SQL предикат с квантором существования представляется выражением EXISTS (SELECT * FROM ...).

Такое выражение считается истинным только тогда, когда результат вычисления "SELECT * FROM ..." является непустым множеством, т. е. когда существует какая-либо запись в таблице, указанной во фразе FROM подзапроса, которая удовлетворяет условию WHERE подзапроса. (Практически этот подзапрос всегда будет коррелированным множеством.)

Рассмотрим примеры. Выдать названия поставщиков, поставляющих продукт с номером 11.

```
SELECT Название
FROM Поставщики
WHERE EXISTS
      (SELECT *
       FROM   Поставки
       WHERE  Код_поставщика = Поставщики.Код_поставщика
         AND  Код_продукта = 11);
```

```
НАЗВАНИЕ
--------
СЫТНЫЙ
КОРЮШКА
```

Система последовательно выбирает строки таблицы `Поставщики`, выделяет из них значения столбцов `Название` и `Код_поставщика`. Затем она проверяет, является ли истинным условие существования, т. е. существует ли в таблице `Поставки` хотя бы одна строка со значением `Код_продукта=11` и значением `Код_поставщика`, равным значению `Код_поставщика`, выбранному из таблицы `Поставщики`. Если условие выполняется, то полученное значение столбца `Название` включается в результат.

Предположим, что первые значения полей `Название` и `Код_поставщика` равны, соответственно, `'СЫТНЫЙ'` и `1`. Так как в таблице `Поставки` есть строка с `Код_продукта=11` и `Код_поставщика=1`, то значение `'СЫТНЫЙ'` должно быть включено в результат.

Хотя этот первый пример только показывает иной способ формулировки запроса для задачи, решаемой и другими путями (с помощью оператора `IN` или соединения), `EXISTS` представляет собой одну из наиболее важных возможностей SQL. Фактически любой запрос, который выражается через `IN`, может быть альтернативным образом сформулирован также с помощью `EXISTS`. Однако обратное высказывание несправедливо.

Выдать название и статус поставщиков, не поставляющих продукт с номером 11:

```
SELECT Название, Статус
FROM   Поставщики
WHERE  NOT EXISTS
       (SELECT *
        FROM   Поставки
        WHERE  Код_поставщика = Поставщики.Код_поставщика
          AND  Код_продукта = 11);
```

Результат:

```
НАЗВАНИЕ СТАТУС
-------- ----------
ПОРТОС   кооператив
ШУШАРЫ   совхоз
ТУЛЬСКИЙ универсам
УРОЖАЙ   коопторг
ЛЕТО     агрофирма
```

```
ОГУРЕЧИК ферма
```

Еще один пример запроса, использующего `EXISTS`, был рассмотрен в *разд. 6.2.6*.

6.3.7. Функции в подзапросе

Теперь, после знакомства с различными формулировками вложенных подзапросов и псевдонимами легче понять текст и алгоритм реализации второго запроса *разд. 6.1* на получение тех поставщиков продуктов для сырников, которые поставляют эти продукты за минимальную цену:

```
SELECT Продукты.Продукт, Поставки.Цена, Поставщики.Название,
       Поставщики.Статус
FROM   Продукты, Состав, Блюда, Поставки, Поставщики
WHERE  Продукты.Код_продукта = Состав.Код_продукта
AND    Состав.Код_блюда = Блюда.Код_блюда
AND    Поставки.Код_продукта = Состав.Код_продукта
AND    Поставки.Код_поставщика = Поставщики.Код_поставщика
AND    Блюда.Блюдо = 'Сырники'
AND    Поставки.Цена = (SELECT MIN(Цена)
                        FROM   Поставки X
                        WHERE  X.Код_продукта =
                               Поставки.Код_продукта );
```

Естественно, что это коррелированный подзапрос: здесь сначала определяется минимальная цена продукта, входящего в состав сырников, и только затем выясняется его поставщик.

На этом примере мы закончим знакомство с вложенными подзапросами, предложив попробовать свои силы в составлении ряда запросов, с помощью механизма таких подзапросов:

1. Выдать названия всех мясных блюд.
2. Выдать количество всех блюд, в состав которых входят помидоры.
3. Выдать блюда, продукты для которых поставляются агрофирмой ЛЕТО.

6.4. Фразы для работы с наборами: *EXCEPT* (*MINUS*), *INTERSECT*, *UNION*

Существует класс операторов для работы с наборами данных: EXCEPT (исключать), INTERSECT (пересекать) и UNION (объединять) — используемые для одновременного манипулирования результирующими наборами двух или более запросов (см. операции "разность", "пересечение" и "объединение" в *разд. 3.3.2*).

Обобщенный синтаксис этих операторов можно представить так:

```
<предложение SELECT_1>
{ EXCEPT | INTERSECT | UNION } [ALL | DISTINCT]
<предложение SELECT_2>
{ EXCEPT | INTERSECT | UNION } [ALL | DISTINCT]
```

Существует только одно важное правило, о котором следует помнить при использовании любого из этих операторов: порядок, количество и тип данных столбцов должны быть идентичны во всех запросах.

Типы данных не обязательно должны быть одинаковы, но они должны быть совместимы. Например, типы CHAR и VARCHAR являются совместимыми. По умолчанию результирующий набор использует размер наибольшего из совместимых типов, и в запросе, в котором объединяются три столбца типа CHAR — CHAR(3), CHAR(10) и CHAR(12), результаты будут в формате CHAR(12), а в столбцы меньшего размера будут добавляться дополнительные пробелы.

Ключевое слово ALL служит для указания, что результирующий набор должен учитывать все дублирующие строки. Для исключения таких строк надо использовать ключевое слово DISTINCT.

Сортировать можно только окончательный результат, т. е. единственная фраза ORDER BY может быть прописана лишь в последнем запросе.

Оператор EXCEPT возвращает результирующий набор для запросов, включающий в себя все записи, полученные первым запросом, которые не обнаруживаются в результатах последующих запросов. (В Oracle существует эквивалент EXCEPT — оператор MINUS.)

Оператор INTERSECT извлекает идентичные строки из результирующих наборов одного или нескольких запросов. В некотором отношении он очень напоминает INNER JOIN.

Оператор UNION объединяет результирующие наборы нескольких запросов и создает из них один результирующий набор.

В Oracle не используется ключевое слово DISTINCT, так как, если отсутствует ключевое слово ALL, то по умолчанию из результирующего набора исключаются дублирующие строки.

В качестве примера использования таких операторов, мы здесь приведем только один, с помощью которого можно было получать внешние соединения до появления в SQL фразы JOIN.

Выполним "старым" способом запросы из *разд. 6.2.7*. Сначала запрос, использующий фразу LEFT JOIN, который приходилось составлять из двух предложений SELECT, объединяемых оператором UNION:

```
SELECT Блюда.Код_блюда, Блюдо, Основа, Код_вида,
       SUBSTR(рецепт,1,20), Вариант
FROM Блюда, Рецепты
WHERE Блюда.Код_блюда = Рецепты.Код_блюда
UNION
SELECT Блюда.Код_блюда, Блюдо, Основа, Код_вида,
       NULL, NULL
FROM блюда
WHERE NOT EXISTS (SELECT *
                  FROM Рецепты
                  WHERE Код_блюда = Блюда.Код_блюда)
ORDER BY код_блюда, вариант;
```

Второе предложение SELECT позволяет добавить в список блюд с рецептами те блюда, для которых рецепты не введены.

Аналогичным образом формулировали запрос, который при соединении таблиц Блюда, Состав и Продукты позволяет вывести те значения блюд и продуктов, которые введены в соответствующие таблицы, но отсутствуют в таблице Состав:

```
SELECT Блюдо, Продукт, Вес
FROM Блюда, Состав, Продукты
WHERE Блюда.Код_блюда = Состав.Код_блюда
AND Состав.Код_продукта = Продукты.Код_продукта
UNION
SELECT Блюдо, NULL, NULL
FROM Блюда
WHERE NOT EXISTS (SELECT *
                  FROM Состав
                  WHERE Код_блюда = Блюда.Код_блюда)
UNION
SELECT NULL, Продукт, NULL
```

```
FROM Продукты
WHERE NOT EXISTS (SELECT *
                  FROM Состав
                  WHERE Код_продукта = Продукты.Код_продукта)
ORDER BY 3;
```

Здесь в список блюд, с входящими в них продуктами, добавляются блюда без списка продуктов и продукты, не входящие в состав блюд.

Кроме того, здесь для сортировки использована допустимая в Oracle конструкция упорядочения по номеру столбца в перечне столбцов фразы `SELECT`.

6.5. Заключение

Краткое знакомство с возможностями предложения `SELECT` показало, что с его помощью можно реализовать все реляционные операции. Кроме того, в предложении `SELECT` выполняются разнообразные вычисления, агрегирование данных, их упорядочение и ряд других операций, позволяющих описать в одном предложении ту работу, для выполнения которой потребовалось бы написать несколько страниц программы на алгоритмических языках Си, Паскаль или на внутренних языках ряда распространенных СУБД.

Пример 6.1. Получить перечень и вес (в килограммах) всех продуктов, которые нужны для приготовления блюд, заказанных отдыхающими на 15.05.1989. Это можно сделать, соединив таблицы `Выбор`, `Меню`, `Блюда`, `Состав` и `Продукты`, а также агрегировав полученное соединение по продуктам, получая сумму столбца `Вес` для каждого из них:

```
SELECT Продукт, ROUND((SUM(Вес)/1000),3)
FROM   Продукты
  JOIN Состав USING (Код_продукта)
  JOIN Блюда USING (Код_блюда)
  JOIN Меню USING (Код_блюда)
  JOIN Выбор USING (Строка)
GROUP BY Продукт;
```

Результат такого запроса имеет вид:

```
ПРОДУКТ  ROUND((SUM(ВЕС)/1000),3)
-------- -------------------------
Говядина                    6,260
Зелень                      1,170
Кофе                        0,096
```

```
Лук                          1,120
Майонез                      1,665
Масло                        1,234
Молоко                      16,400
Мука                         0,363
Помидоры                    10,015
Рис                          3,120
Сахар                        2,635
Сметана                      3,180
Творог                       3,590
Яблоки                      13,595
Яйца                         4,570
```

Так как в таблице состав всех продуктов, входящих в блюдо, дан в граммах, то для получения суммарного веса продуктов, необходимых для приготовления блюд на все трапезы 32 отдыхающих, в килограммах мы разделили эту сумму на 1000. Кроме того, мы воспользовались функцией ROUND (см. *разд. 4.6* и *5.5.3*) для округления результатов и сохранения в них трех цифр после запятой.

Следовательно, для приготовления всех блюд на 15.05.1989 требуется чуть больше 6 кг говядины, 13 кг яблок, 16 литров молока, т. е. вполне приемлемые количества.

Пример 6.2. А сколько же продуктов в кладовой пансионата и какова средняя стоимость каждого из них? Если считать, что все поставленные до 15.05.1989 продукты еще не расходовались, то можно воспользоваться таким запросом

```
SELECT (SELECT Продукт
        FROM Продукты
        WHERE Код_продукта = Поставки.Код_продукта) Продукт,
        SUM(К_во) Всего,
        ROUND(SUM(Цена*К_во)/SUM(К_во),2) Средняя
FROM   Поставки
GROUP BY Код_продукта
ORDER BY Продукт;
```

в результате выполнения которого будет получена таблица:

```
ПРОДУКТ       ВСЕГО    СРЕДНЯЯ
-------- ---------- ----------
Говядина        370       3,71
Зелень           30       2,67
```

```
Кофе              50        4,5
Лук              220       0,58
Майонез          100       2,52
Масло            250          4
Молоко           200        0,4
Мука              70        0,5
Помидоры         150       1,17
Рис              190       0,95
Сахар            250       0,95
Сметана          220       2,71
Творог           150          1
Яблоки           370       1,73
Яйца             170       1,88
```

Здесь мы впервые включили подзапрос в список фразы `SELECT` для получения названия продукта по его коду, взятому из таблицы `Поставки`.

Пример 6.3. Так как при составлении меню шеф-повар должен знать текущее количество и стоимость продуктов, то целесообразно иметь запрос, который бы учитывал не только поступление продуктов в кладовую, но и их ежедневный расход. Этот запрос можно оформить в виде представления (например, `Наличие`) с тем, чтобы у шеф-повара всегда существовала виртуальная таблица с необходимой ему информацией (о представлениях рассказано в *разд. 4.3* и *7.3.4*).

```
CREATE OR REPLACE VIEW НАЛИЧИЕ (Код_продукта, К_во, Стоимость)
AS
SELECT Расход.Код_продукта, Склад.Всего - Расход.Итого К_во,
       Стоимость
FROM (SELECT Код_продукта, ROUND(SUM(Вес)/1000,3) Итого
      FROM   Продукты
        JOIN Состав USING (Код_продукта)
        JOIN Блюда USING (Код_блюда)
        JOIN Меню USING (Код_блюда)
        JOIN Выбор USING (Строка)
      GROUP BY Код_продукта) Расход,
      (SELECT Код_продукта, SUM(к_во) Всего,
        ROUND(SUM(цена*к_во)/SUM(к_во),2) Стоимость
      FROM   поставки
      GROUP BY Код_продукта) Склад
WHERE Расход.Код_продукта = Склад.Код_продукта;
```

Во FROM-фразе этого представления используются не базовые таблицы, а две рабочие таблицы, создаваемые подзапросами. Первый из них, которому мы дали псевдоним ВСЕГО, это несколько модифицированный запрос из примера 6.1. Второй подзапрос с псевдонимом СКЛАД — это модифицированный запрос из примера 6.2.

Если теперь выполнить запрос

```
SELECT * FROM Наличие;
```

то СУБД произведет загрузку результирующих данных в виртуальную таблицу НАЛИЧИЕ и выдаст ее результат в виде:

```
КОД_ПРОДУКТА       К_ВО  СТОИМОСТЬ
------------ ---------- ----------
           1     363,74       3,71
           3     248,77          4
           4      98,33       2,52
           5     165,43       1,88
           6     216,82       2,71
           7      183,6        0,4
           8     146,41          1
          10     218,88       0,58
          11     139,98       1,17
          12      28,83       2,67
          13     186,88       0,95
          14      69,64        0,5
          15      356,4       1,73
          16     247,36       0,95
          17       49,9        4,5
```

Пример 6.4. Если требуется получить калорийность и стоимость тех блюд, для которых:

- ❒ есть все составляющие их продукты;
- ❒ калорийность не превышает 400 ккал;
- ❒ стоимость не превышает 0.5 рубля;
- ❒ результат надо упорядочить по возрастанию калорийности блюд в рамках их видов,

то можно дать запрос:

```
SELECT   Вид, Блюдо,
         ROUND(SUM(((Белки+Углев)*4.1+Жиры*9.3)*Вес/1000),1) Калорий,
    ROUND(SUM(Стоимость*Вес/1000)+MIN(Труд/100),2) Цена
  FROM Блюда
```

```
    JOIN Состав USING (Код_блюда)
    JOIN Продукты USING (Код_продукта)
    JOIN Наличие USING (Код_продукта)
    JOIN Виды_блюд USING (Код_вида)
    WHERE Код_блюда NOT IN
                (SELECT Код_блюда
                 FROM Состав
                 WHERE Код_продукта NOT IN
                      (SELECT Код_продукта
                       FROM Наличие))
  GROUP BY Вид, Блюдо
  HAVING SUM(Стоимость*Вес/1000)+MIN(Труд/100) < 0.5
     AND SUM(((Белки+Углев)*4.1+Жиры*9.3)*Вес/1000) < 400
  ORDER BY Вид, Калорий;
```

Результат выполнения этого запроса имеет вид:

```
ВИД     БЛЮДО                КАЛОРИЙ       ЦЕНА
------- ---------------- ---------- ----------
Горячее Помидоры с луком      244,8       0,45
Горячее Драчена               334,1       0,34
Горячее Каша рисовая          339,3       0,28
Горячее Омлет с луком         355,0       0,37
Десерт  Яблоки печеные        170,4       0,32
Десерт  Крем творожный        394,4       0,29
Закуска Салат летний          155,6       0,34
Закуска Салат витаминный      217,5       0,40
Закуска Творог                330,1       0,24
Напиток Кофе черный             7,1       0,05
Напиток Компот                 74,5       0,15
Напиток Кофе на молоке        154,8       0,11
Напиток Молочный напиток      265,0       0,36
Суп     Суп молочный          396,6       0,23
```

Обратите внимание на прием, использованный при суммировании стоимостей продуктов, входящих в состав блюда, и стоимости его приготовления (`Труд`).

В состав блюда может входить несколько продуктов, и для получения общей стоимости всех этих продуктов надо использовать агрегирующую функцию `SUM(Стоимость*Вес/1000)` с группировкой по столбцу `БЛЮДО`. Однако если к этой сумме добавить слагаемое `Труд/100`, то будет выдано сообщение "ORA-00979: выражение не является выражением GROUP BY". Сделав же `Труд/100` аргументом агрегирующей функции `MIN`, мы как бы "обманываем"

СУБД, получая искомый результат. Равным образом можно было бы использовать вместо MIN функции MAX или AVG. Можно поступить и по-другому: не добавлять агрегирующую функцию MIN, а добавить во фразу GROUP BY столбец Труд.

```
SELECT   Вид, Блюдо,
         ROUND(SUM(((Белки+Углев)*4.1+Жиры*9.3)*Вес/1000),1) Калорий,
    ROUND(SUM(Стоимость*Вес/1000)+Труд/100,2) Цена
  FROM Блюда
    JOIN Состав USING (Код_блюда)
    JOIN Продукты USING (Код_продукта)
    JOIN Наличие USING (Код_продукта)
    JOIN Виды_блюд USING (Код_вида)
    WHERE Код_блюда NOT IN
                (SELECT Код_блюда
                 FROM Состав
                 WHERE Код_продукта NOT IN
                      (SELECT Код_продукта
                       FROM Наличие))
  GROUP BY Вид, Блюдо, Труд
  HAVING SUM(Стоимость*Вес/1000)+Труд/100 < 0.5
     AND SUM(((Белки+Углев)*4.1+Жиры*9.3)*Вес/1000) < 400
  ORDER BY Вид, Калорий;
```

ЧАСТЬ III

ЯЗЫК SQL. ИЗМЕНЕНИЕ ДАННЫХ

Глава 7

Организация доступа к базе данных

7.1. О системе баз данных

В *главе 1* рассматривалась концепция баз данных и архитектура СУБД. Эти сведения позволяют определить систему баз данных как компьютеризированную систему, основное назначение которой — хранение информации и предоставление пользователям средств для ее извлечения и модификации. Система баз данных состоит из четырех главных компонентов: данные, аппаратное обеспечение, программное обеспечение и пользователи.

7.1.1. Данные

Системы баз данных устанавливают как на персональные компьютеры (как правило, однопользовательские системы), так и на большие вычислительные машины (многопользовательские системы). В однопользовательских системах к базе данных может получить доступ одновременно только один пользователь, а в многопользовательских — сразу несколько пользователей. С точки зрения пользователей между этими системами фактически не существует большого различия, поскольку основная цель многопользовательских систем состоит в том, чтобы позволить каждому отдельному пользователю работать с ней так, как если бы он работал с однопользовательской системой. Различия между этими двумя видами систем проявляются в их внутренней структуре и поэтому не видны конечному пользователю.

В общем случае данные в базе данных являются интегрированными и разделяемыми. Под интеграцией данных подразумевается возможность представить базу данных как объединение нескольких отдельных файлов данных, полностью или частично исключающее избыточность хранения информации. Разделяемость данных — это возможность использования несколькими

различными пользователями отдельных элементов, хранимых в базе данных: каждый пользователь может получить доступ к одним и тем же данным, возможно, даже одновременно (параллельный доступ). Такое разделение данных, с параллельным или последовательным доступом, частично является следствием того, что база данных имеет интегрированную структуру.

Одним из следствий интеграции и разделяемости является то, что каждый конкретный пользователь обычно имеет дело лишь с небольшой частью всей базы данных, причем обрабатываемые различными пользователями части могут произвольным образом перекрываться. Иначе говоря, каждая база данных воспринимается различными пользователями по-разному. Фактически, даже те два пользователя базы данных, которые работают с одними и теми же частями базы данных, могут иметь значительно отличающиеся представления о них.

7.1.2. Аппаратное обеспечение

Аппаратной части системы в данной книге уделяется мало внимания, так как проблемы, присущие этой области, не являются специфическими для систем баз данных и достаточно подробно освещены в литературе.

7.1.3. Программное обеспечение

Между данными, которые реально хранятся на компьютере, и пользователями системы располагается СУБД. Все запросы пользователей на получение доступа к базе данных обрабатываются СУБД. Все имеющиеся средства добавления файлов (или таблиц), выборки и обновления данных в этих файлах или таблицах также предоставляет СУБД. Основная задача СУБД — дать пользователю базы данных возможность работать с ней, не вникая во все подробности работы на уровне аппаратного обеспечения. Следовательно, СУБД позволяет конечному пользователю рассматривать базу данных как объект более высокого уровня по сравнению с аппаратным обеспечением, а также предоставляет в его распоряжение набор операций на языке высокого уровня (SQL).

7.1.4. Пользователи

Пользователей можно разделить на три большие и отчасти перекрывающиеся группы (см. рис. 1.2).

- *Прикладные программисты*, создающие на каком-либо языке программирования (C++, Java, PHP или специализированном языке конкретной СУБД, например, PL/SQL) пакетные или интерактивные приложения,

предназначенные для работы конечных пользователей. Они предоставляют пользователям непосредственный оперативный доступ к базе данных через рабочую станцию, терминал или персональный компьютер. Большинство современных приложений относится именно к этой категории.

- *Конечные пользователи*, получающие доступ к базе данных с помощью одного из интерактивных приложений или же интерфейса, интегрированного в программное обеспечение самой СУБД. Например, в составе СУБД Oracle существует встроенное приложение SQL*Plus, обеспечивающее интерактивное выполнение операторов языка SQL для определения данных, манипулирования данными и определения правил доступа к СУБД Oracle. Оно может использоваться и как инструмент для создания отчетов. Существуют также различные бесплатные (например, Oracle SQL Developer) и покупные (например, PL/SQL Developer или TOAD) пакеты, предназначенные для повышения скорости разработки баз данных и приложений, а также упрощения ежедневных задач администрирования.
- *Администраторы баз данных*, работа которых заключается в создании самих баз данных и организации технического контроля, необходимого для обеспечения решений, принятых при проектировании базы данных. Они (он) несут также ответственность за обеспечение необходимого быстродействия системы, ее техническое обслуживание и защиту данных.

7.2. Защита данных

После создания базы данных, загрузки данными и ввода в эксплуатацию, необходимо подключить к ней пользователей и принять меры к защите хранимых данных от несанкционированного доступа, изменения или разрушения.

В самом общем виде требования к безопасности формулируются так:

- данные в любой таблице должны быть доступны не всем пользователям, а лишь некоторым из них;
- некоторым пользователям разрешено обновлять данные в таблицах, в то время как другие допускаются лишь к выборке данных из этих же таблиц;
- для некоторых таблиц необходимо обеспечить выборочный доступ к ряду ее столбцов;
- некоторым пользователям должен быть запрещен непосредственный (через запросы) доступ к таблицам, но разрешен доступ к этим же таблицам в диалоге с прикладной программой.

Для обеспечения этих требований необходимо, чтобы каждый из пользователей был известен СУБД и ему были предоставлены определенные права на доступ к тем или иным объектам базы данных.

Обычно лиц, которым необходимо дать доступ к базе данных или которые желают получить такой доступ (и имеют на это право), включают в состав пользователей, давая им имя (идентификатор) и пароль. При этом администратор базы данных сначала предоставляет каждому пользователю минимальный набор привилегий, расширяя затем этот набор для тех или иных пользователей по мере необходимости.

Так для студентов, с которыми работают авторы, создана база данных "UCHEB" (см. *часть VI*). В этой базе данных пользователями являются все студенты и преподаватели университета. Их имена практически совпадают с их идентификаторами (ИД) в таблице Н_ЛЮДИ (см. *разд. 5.4*). Эти имена и пароли выдаются студентам при поступлении и используются ими в процессе обучения как для работы с Корпоративным порталом университета (**http://cis.ifmo.ru**), так и для работы с базой данных во время занятий и различных видов внеурочной деятельности. Студентам предоставляется возможность соединяться с базой данных "UCHEB", делать разнообразные запросы на получение данных из ее таблиц, создавать собственные базы данных (в пределах выделенных квот) и т. п.

Привилегии в СУБД могут быть разделены на две категории.

- *Системные привилегии* позволяют пользователям выполнять определенное действие на уровне системы или над конкретным типом объектов. К системным привилегиям можно отнести, например, привилегию на создание таблиц (`CREATE TABLE`) или привилегию на удаление строки из любой таблицы в базе данных (`DELETE ANY TABLE`). Большинство системных привилегий должны быть доступны только администраторам и разработчикам приложений. Бесконтрольная их выдача может создать реальную угрозу как функционированию системы, так и потери данных.
- *Объектные привилегии* позволяют пользователям выполнять определенные действия с конкретным объектом. К объектным привилегиям, например, относятся привилегии на удаление строки в определенной таблице или запуск определенной процедуры. Объектные привилегии назначаются конечным пользователям для того, чтобы они могли работать с приложениями к базе данных для решения конкретных задач.

Наряду с привилегиями, ограничивающими доступ к таблицам, ключевую роль в защите данных играют также представления (см. *разд. 4.2*). Создавая представление и давая пользователю разрешение на доступ к нему, а не к исходной таблице, можно тем самым ограничить доступ пользователя к опре-

деленным столбцам и строкам. Таким образом, представления позволяют осуществить четкий контроль над тем, какие данные доступны тому или иному пользователю.

Наконец, возможна и более изощренная (но редко используемая) форма защиты — шифрование данных, т. е. хранение и передача особо важных данных в зашифрованном виде.

7.3. Средства языка SQL

В действующем стандарте языка SQL поддерживаются как механизм авторизации доступа (задания привилегий), так и механизм представлений. Шифрование данных в стандарте отсутствует и поддерживается (при необходимости) специальными приложениями. В СУБД Oracle есть механизм шифрования данных (см. *разд. 13.3.1*).

7.3.1. Предложение *GRANT*

Это предложение назначает пользователям и ролям (см. далее) привилегии, которые позволяют им обращаться к объектам базы данных и использовать их (т. е. объектные привилегии).

Синтаксис предложения `GRANT` в стандарте SQL:2003 имеет вид:

```
GRANT { {объектная_привилегия [,...]| роль [,...]} }
[ON имя_объекта_базы]
[TO получающий_привилегию [,...] ];
```

Рассмотрим ключевые слова.

`Объектная_ привилегия`

В этом предложении можно присвоить одну или несколько привилегий, разделяя их запятыми (нельзя лишь объединять `ALL PRIVILEGES` с другими привилегиями). Существуют следующие стандартные привилегии:

- `ALL PRIVILEGES` — краткое обозначение всех привилегий, которое не рекомендуется использовать, поскольку это способствует неаккуратному назначению прав доступа;
- `EXECUTE` — предоставляется право запускать хранимую процедуру, пользовательскую функцию или пакет (см. *главу 18*);
- `{ SELECT | INSERT | UPDATE | DELETE }` — предоставляется право выполнять соответствующие операции применительно к указанному объекту базы данных (таблице, представлению и пр.);

- REFERENCES — предоставляется право определять ограничения, обеспечивающие ссылочную целостность.

Ограничимся пока этими объектными привилегиями.

Роль

Роль — это именованный набор привилегий, которые можно присваивать пользователям и другим ролям базы данных. Если роль назначается пользователю, этот пользователь получает все привилегии и допуски, содержащиеся в данной роли. Роли повсеместно используются как один из лучших способов обеспечения безопасности и управления привилегиями в базе данных. Роль должна быть заранее создана с помощью предложения CREATE ROLE (см. далее).

ON *имя_объекта_базы*

Привилегии присваиваются для доступа к конкретному существующему объекту базы данных (*имя_объекта_базы*). К объектам базы данных относятся: таблицы, представления, последовательности, хранимые процедуры и т. д.

TO *получатель_привилегии*

Привилегия присваивается указанному в *получатель_привилегии* пользователю или роли. Можно присваивать привилегии нескольким пользователям и (или) ролям, для чего их разделяют запятыми. В качестве альтернативы можно присвоить привилегии с ключевым словом PUBLIC, это означает, что все пользователи (в том числе и те, которые появятся в будущем) будут иметь указанные привилегии.

В Oracle существует значительно больше, чем в стандарте, объектных привилегий и ключевых слов. С ними можно познакомиться в *разд. 4.4.5*.

Например, пусть нами был создан новый пользователь Повар, которому мы хотим назначить привилегию на выборку данных из таблицы Меню пользователя COOK

```
GRANT SELECT ON COOK.МЕНЮ TO Повар;
```

Создадим роль Пансион и предоставим ей несколько привилегий:

```
CREATE ROLE Пансион;
GRANT SELECT ON COOK.РЕЦЕПТЫ TO Пансион;
GRANT SELECT ON COOK.ТРАПЕЗЫ TO Пансион;
GRANT SELECT ON COOK.ВИДЫ_БЛЮД TO Пансион;
GRANT SELECT ON COOK.БЛЮДА TO Пансион;
```

Назначим роль Пансион пользователю Повар:

```
GRANT Пансион TO Повар;
```

7.3.2. Предложение *REVOKE*

Предложение REVOKE служит для отмены назначенных привилегий и ролей и имеет синтаксис:

```
REVOKE { {привилегия [,...]| роль [,...]} }
[ON имя_объекта_базы]
[FROM имя_получателя [,...] ];
```

Рассмотрим ключевые слова.

привилегия

В этом предложении можно отменить одну или несколько привилегий доступа, разделяя их запятыми.

- ALL PRIVILEGES — отменяются все назначенные привилегии доступа к указанному объекту;
- EXECUTE — отменяется право запускать хранимую процедуру, пользовательскую функцию или пакет;
- { SELECT | INSERT | UPDATE | DELETE } — отменяется право выполнять соответствующие операции применительно к указанному объекту базы данных (таблице, представлению и пр.);
- REFERENCES — отменяется право определять ограничения, обеспечивающие ссылочную целостность.

Роль

Отменяется назначение роли пользователю или другой роли.

ON *имя_объекта_базы*

Пользователь или роль лишаются привилегии доступа к конкретному существующему объекту базы данных (*имя_объекта_базы*). К объектам базы данных относятся: таблицы, представления, последовательности, хранимые процедуры и т. д.

FROM *имя_получателя*

Указывается пользователь или роль, лишающаяся данной привилегии.

Например, пусть нам необходимо отобрать у пользователя Повар привилегию на выборку данных из таблицы Меню

```
REVOKE SELECT ON COOK.МЕНЮ FROM Повар;
```

Отменим роль Пансион у пользователя Повар:

```
REVOKE Пансион FROM Повар;
```

7.3.3. Синонимы

Синоним — псевдоним (дополнительное имя) для объекта (таблицы, представления, последовательности и пр.). Они используются для:

- маскировки действительного имени и владельца объекта;
- обеспечения общего доступа к объекту;
- достижения прозрачности местоположения для таблиц, представлений или программных единиц удаленной базы данных;
- упрощения кодирования предложений SQL для пользователей базы данных.

Синтаксис предложения для создания синонима имеет вид:

```
CREATE [ OR REPLACE ] [ PUBLIC ] SYNONYM
    [схема.]синоним
  FOR [схема.]объект;
```

где объектом может быть:

- таблица;
- представление;
- материализованное представление;
- последовательность;
- хранимая процедура (функция или пакет);
- синоним.

Синоним может быть общим (`PUBLIC`) или личным. Обычный пользователь может создать личный синоним, который будет доступен только этому пользователю. Администраторы баз данных чаще всего создают общие синонимы, благодаря которым объекты базовых схем становятся доступными для общего пользования всех пользователей базы данных.

Преимуществом общедоступных синонимов является то, что они могут создаваться и поддерживаться в одном месте. Если во время создания синонима определена схема, то пользователям не нужно указывать имя схемы при выполнении запросов по отношению к таблице. Одним из распространенных применений синонимов является создание общедоступного синонима таблицы с таким же именем, что и у исходной таблицы.

Например, для того чтобы все пользователи (в том числе и пользователь `Повар`) могли обращаться к некоторым таблицам пользователя `COOK` без ука-

зания имени схемы, необходимо после назначения привилегий (*разд. 7.3.1*) создать синонимы:

```
CREATE PUBLIC SYNONYM МЕНЮ FOR COOK.МЕНЮ;
CREATE PUBLIC SYNONYM РЕЦЕПТЫ FOR COOK.РЕЦЕПТЫ;
CREATE PUBLIC SYNONYM ТРАПЕЗЫ FOR COOK.ТРАПЕЗЫ;
CREATE PUBLIC SYNONYM ВИДЫ_БЛЮД FOR COOK.ВИДЫ_БЛЮД;
CREATE PUBLIC SYNONYM БЛЮДА FOR COOK.БЛЮДА;
```

7.3.4. Представления

Как уже упоминалось в *разд. 4.3*, представление — это пустая именованная таблица, определяемая перечнем тех столбцов таблиц и признаками тех их строк, которые хотелось бы в ней увидеть. Представление является как бы "окном" в одну или несколько базовых таблиц. Оно создается с помощью предложения `CREATE VIEW` (создать представление), упрощенный синтаксис которого имеет вид:

```
CREATE [OR REPLACE] VIEW имя_представления {[(столбец [,...])]}
AS предложение_select
[WITH CHECK OPTION];
```

При выполнении этого предложения создается представление (виртуальная таблица) с именем, указанным в *имя_представления*.

Фраза `OR REPLACE` позволяет заменить существующее представление с именем *имя_представления* на новое.

Список имен столбцов (*столбец* `[,...]`), количество которых должно совпадать с количеством столбцов, генерируемых *предложением_select*, должен быть определен лишь в тех случаях, когда:

- ❒ хотя бы один из столбцов *предложения_select* не имеет имени, так как создается с помощью выражения, SQL-функции или константы;
- ❒ два или более столбцов подзапроса имеют одно и то же имя.

Если же список отсутствует, то представление наследует имена столбцов из подзапроса.

Необязательная фраза `WITH CHECK OPTION` (с проверкой) указывает, что для операций `INSERT` и `UPDATE` над этим представлением должна осуществляться проверка, обеспечивающая удовлетворение `WHERE`-фразы подзапроса.

Например, создадим представление `Мясные_блюда`

```
CREATE VIEW Мясные_блюда
AS SELECT БЛ, Блюдо, В, Выход
```

```
FROM   Блюда
WHERE  Основа = 'Мясо';
```

которое может рассматриваться пользователем как новая таблица в базе данных.

Уничтожение ненужных представлений выполняется с помощью предложения `DROP VIEW` (уничтожить представление), имеющего следующий формат:

```
DROP VIEW представление;
```

Операции выборки из представлений

Создав представление `Мясные_блюда`, пользователь может считать, что в базе данных реально существует такая таблица, и дать, например, запрос на получение из нее всех данных:

```
SELECT *
FROM   Мясные_блюда;
```

результат которого имеет вид

БЛ	Блюдо	В	Выход
2	Салат мясной	З	200
6	Мясо с гарниром	З	250
9	Суп харчо	С	500
13	Бастурма	Г	300
14	Бефстроганов	Г	210

Поскольку при определении представления может быть использован любой допустимый подзапрос, то выборка данных может осуществляться как из базовых таблиц, так и из представлений:

```
CREATE VIEW Горячие_мясные_блюда
       AS SELECT     Блюдо, Продукт, Вес
              FROM   Мясные_блюда, Состав, Продукты
              WHERE  Мясные_блюда.БЛ = Состав.БЛ
              AND    Продукты.ПР = Состав.ПР
              AND    В = 'Г';
```

Если теперь возникла необходимость получить сведения о горячих мясных блюдах, в состав которых входят помидоры, то можно сформировать запрос

```
SELECT Блюдо, Продукт, Вес
FROM   Горячие_мясные_блюда
```

```
WHERE  Блюдо IN
       ( SELECT Блюдо
         FROM Горячие_мясные_блюда
         WHERE  Продукт = 'Помидоры')
```

и получить:

Блюдо	Продукт	Вес
Бастурма	Говядина	180
Бастурма	Помидоры	100
Бастурма	Лук	40
Бастурма	Зелень	20
Бастурма	Масло	5

Легко заметить, что данный запрос, осуществляющий выбор данных через два представления, выглядит для пользователя точно так же, как обычный SELECT, оперирующий обычной базовой таблицей. Однако СУБД преобразует его при выполнении в эквивалентную операцию над лежащими в основе базовыми таблицами (перед выполнением проводит слияние выданного пользователем SELECT с предложениями SELECT из описаний представлений, хранящихся в каталоге).

Обновление представлений

Рассматриваемые в *главе 8* операции DELETE, INSERT и UPDATE могут оперировать не только базовыми таблицами, но и представлениями. Однако если из базовых таблиц можно удалять любые строки, обновлять значения любых их столбцов и вводить в такие таблицы новые строки, то этого нельзя сказать о представлениях, не все из которых являются обновляемыми.

Безусловно обновляемыми являются представления, полученные из единственной базовой таблицы простым исключением некоторых ее строк и (или) столбцов, обычно называемые "представление-подмножество строк и столбцов". Таким является представление Мясные_блюда, полученное из базовой таблицы Блюда исключением из нее столбца Труд и строк, не содержащих значение 'Мясо' в столбце Основа. Работая с ним, можно:

- вставить (операция INSERT) новую строку, например строку (34, 'Шашлык', 'Г', 150), фактически вставляя соответствующую строку (34, 'Шашлык', 'Г', 150, NULL) в лежащую в основе базовую таблицу Блюда;

- удалить (операция DELETE) существующую строку из представления, например строку (13, 'Бастурма', 'Г', 300), фактически удаляя соответствующую строку (13, 'Бастурма', 'Г', 300, 5) из таблицы Блюда;
- обновить (операция UPDATE) какое-либо поле в существующей строке, например увеличить массу порции Бефстроганова с 210 до 250 граммов, фактически осуществляя то же самое изменение в соответствующем поле таблицы Блюда.

Однако если бы представление Мясные_блюда имело вместо столбца Выход столбец Вых_труд, полученный путем суммирования значений столбцов Выход и Труд таблицы Блюда, то указанные ранее операции INSERT и UPDATE были бы отвергнуты системой.

Действительно, как распределить вводимое значение столбца Вых_труд (153 или 254) между значениями столбцов Выход и Труд базовой таблицы Блюда? Была бы отвергнута и операция удаления масла из состава Бастурмы, если бы ее попытались выполнить путем удаления строки ('Бастурма', 'Масло', 5) в представлении Горячие_мясные_блюда.

Встает множество вопросов:

- надо ли удалять из базовой таблицы Блюда строку, содержащую значение 'Бастурма' в столбце Блюдо;
- надо ли удалять из базовой таблицы Продукты строку, содержащую значение 'Масло' в столбце Продукт;
- надо ли удалять из базовой таблицы Состав все строки, содержащие значение 5 в столбце Вес?

Последний вопрос возник потому, что при конструировании представления Горячие_мясные_блюда в него не была включена информация о связях между лежащими в его основе базовыми таблицами Блюда, Состав и Продукты. И следовательно, у системы нет прямых путей для поиска той единственной строки таблицы Состав, которая должна быть удалена.

Таким образом, некоторые представления по своей природе обновляемы, в то время как другие таковыми не являются. Здесь следует обратить внимание на слова "по своей природе". Дело заключается не просто в том, что некоторая СУБД не способна поддерживать определенные обновления, в то время как другие СУБД могут это делать. Никакая СУБД не может непротиворечивым образом поддерживать без дополнительной помощи обновление такого представления, как Горячие_мясные_блюда. "Без дополнительной помощи" означает здесь "без помощи какого-либо человека — пользователя".

Как было указано ранее, к теоретически обновляемым представлениям относятся представления-подмножества строк и столбцов. Однако существуют

некоторые представления, которые не являются представлениями-подмножествами строк и столбцов, но также теоретически обновляемы. Хотя известно, что такие есть и можно привести их примеры, но невозможно дать их формального определения. Неверным является такое формальное определение некоторых авторов — "нельзя обновлять соединение". Во-первых, в некоторых соединениях с успехом выполняется операция `UPDATE`, а, во-вторых, как было показано ранее, не обновляемы и некоторые представления, не являющиеся соединениями. Кроме того, не все СУБД поддерживают обновление любых теоретически обновляемых представлений. Поэтому пользователь должен сам оценивать возможность использования операций `DELETE`, `INSERT` или `UPDATE` в созданном им представлении.

Для чего нужны представления

Одна из основных задач, которую позволяют решать представления, — обеспечение независимости пользовательских программ от изменения логической структуры базы данных при ее расширении и (или) изменении размещения столбцов, возникающего, например, при расщеплении таблиц. В последнем случае можно создать представление-соединение с именем и структурой расщепленной таблицы, позволяющее сохранить программы, существовавшие до изменения структуры базы данных.

Кроме того, представления дают возможность различным пользователям по-разному видеть одни и те же данные, возможно, даже в одно и то же время. Это особенно ценно при работе различных категорий пользователей с единой интегрированной базой данных. Пользователям предоставляют только интересующие их данные в наиболее удобной для них форме (окно в таблицу или в любое соединение любых таблиц).

Наконец, от определенных пользователей могут быть скрыты некоторые данные, невидимые через предложенное им представление. Таким образом, принуждение пользователя осуществлять доступ к базе данных через представления является простым, но эффективным механизмом для управления санкционированием доступа.

Например, для того чтобы скрыть от какого-то пользователя сведения о витаминах, входящих в продукты (табл. 3.3), и данные о судаке (`Код_продукта=2`) из той же таблицы, можно закрыть ему к ней доступ, а открыть доступ к представлению `Составляющие`:

```
CREATE VIEW Составляющие
AS SELECT Код_продукта, Продукт, Белки, Жиры, Углев
FROM   Продукты
WHERE  Код_продукта <> 2;
```

7.3.5. Разграничение доступа к записям таблицы

В зависимости от выдвигаемых требований в общем случае существует два основных способа решения данной задачи:

- с использованием представлений;
- с использованием процедур, функций, пакетов.

Оба метода имеют характерные особенности. Например, при использовании представления, сделанного по очень большой таблице или таблицам, в соединениях будет наблюдаться резкое падение производительности, из-за того, что по представлению нельзя построить индекс. При работе с процедурами, функциями или пакетами теряется "прозрачность" представления. И к тому же возникает множество проблем, в случае изменения структуры таблиц, явно прописанных в тексте процедур, функций, пакетов. Так что необходимо очень тщательно подходить к выбору стратегии разграничения доступа.

При использовании программных объектов подразумевается, что пользователь работает в первую очередь с приложением, а не с командной строкой SQL*Plus.

Всю работу с таблицами в свете решаемой задачи можно разделить на два вида:

- пользователь работает только со своими строками, например которые он сам и создал;
- доступ пользователя к записям таблицы определяется каким-то более сложным образом, например, рядовые сотрудники имеют один уровень доступа, более продвинутые — имеют значительно расширенный доступ, а единицы — обладают правом работы со всей таблицей (мандатный контроль доступа).

Соответственно, решение задачи в первом случае сводится к введению дополнительного поля в таблицу, с которой производится работа. Во втором случае, все более серьезно. Для организации такого доступа нам потребуется создание дополнительной таблицы полномочий, в которой каждому пользователю мы сопоставим его уровень доступа.

Итак, разберем все по порядку. В качестве примера возьмем базу данных "COOK" и рассмотрим следующих ее пользователей:

- заведующий производством (пользователь `zav`);
- первый агент по закупкам, работающий с питерскими организациями (`agent1`);
- второй агент по закупкам, работающий с эстонскими организациями (`agent2`).

Соответственно нет необходимости в том, чтобы агенты по закупкам выясняли информацию о совершенных друг с другом сделках. А заведующему производством неважно, кто совершил ту или иную сделку.

Работа только со своими строками

С использованием представлений

Задача решается просто — создаются соответствующие представления для таблиц `Поставщики` и `Поставки`. И затем выдается привилегия на доступ к этим представлениям. А для заведующего производством можно дать напрямую привилегию на `SELECT` этих таблиц.

Однако предварительно нужно изменить эти таблицы следующим образом.

```
ALTER TABLE Поставщики ADD agent_column VARCHAR2(20);
ALTER TABLE Поставки ADD agent_column VARCHAR2(20);
```

Соответственно в эти колонки будут заноситься имена пользователей-агентов, в таблицу `Поставщики` вручную, а в таблицу `Поставки` автоматически с помощью триггера (см. *разд. 18.4*).

Скрипты по созданию триггеров и представлений:

```
CREATE OR REPLACE TRIGGER postavshiki_ins
BEFORE INSERT ON Поставщики
FOR EACH ROW
BEGIN
  :NEW.agent_column := USER;
END postavshiki_ins;

CREATE OR REPLACE TRIGGER postavki_ins
BEFORE INSERT ON Поставки
FOR EACH ROW
BEGIN
  :NEW.agent_column := USER;
END postavki_ins;

CREATE VIEW postavshiki AS SELECT ПС, НАЗВАНИЕ, СТАТУС,
ГОРОД, АДРЕС, ТЕЛЕФОН
FROM ПОСТАВЩИКИ WHERE agent_column = USER;

CREATE VIEW postavki AS SELECT ПС, ПР, ЦЕНА, К_ВО, ДАТА
FROM ПОСТАВКИ WHERE agent_column = USER;
```

Скрипты для выдачи привилегий на работу с представлениями и таблицами (для удобства, все сделано с использованием ролей):

```
GREATE ROLE zav_role;
GRANT SELECT ON Поставщики TO zav_role;
GRANT SELECT ON Поставки TO zav_role;

GREATE ROLE agent_role;
GRANT SELECT, INSERT ON postavshiki TO agent_role;
GRANT SELECT, INSERT ON postavki TO agent_role;
```

Выдаем роли пользователям `zav`, `agent1`, `agent2`.

```
GRANT agent_role TO agent1;
GRANT agent_role TO agent2;
GRANT zav_role TO zav;
```

Далее они могут безболезненно работать с таблицами `Поставщики` и `Поставки` (хотя невредно было бы создать `PUBLIC`-синонимы).

```
CREATE PUBLIC SYNONYM Поставщики FOR cook.Поставщики;
CREATE PUBLIC SYNONYM Поставки FOR cook.Поставки;
CREATE PUBLIC SYNONYM postavshiki FOR cook.postavshiki;
CREATE PUBLIC SYNONYM postavki FOR cook.postavki;
```

С использованием пакетов

Пример использования пакета рассмотрен в *разд. 18.5*.

Глава 8

Внесение изменений в базу данных

8.1. Особенности и синтаксис предложений модификации

Для изменения содержимого базы данных SQL предусматривает три операции: `INSERT` (вставка строк в таблицу), `DELETE` (удаление строк из таблицы) и `UPDATE` (обновление значений в существующих строках таблицы).

Операция вставки `INSERT` может быть единичной либо групповой. Для единичной вставки необходимо явным образом определить значения колонок новой строки. Если указан неполный перечень колонок таблицы, то оставшиеся колонки получают неопределенное значение (`NULL`).

Операция групповой вставки предполагает добавление строк в таблицу из некоторой другой таблицы, указанной явно или с помощью `SELECT`-предложения.

Операция удаления `DELETE` является групповой, т. е. применяется ко всем строкам таблицы, удовлетворяющим требуемому условию. В отличие от запись-ориентированных языков манипулирования данными, SQL не использует понятия текущей строки в таблице, предполагая, что любая строка может быть однозначно идентифицирована в таблице с помощью значения первичного ключа. Если условие отсутствует, то удаляются все строки таблицы. Условие может использоваться для проверки вхождения значения в некоторую другую таблицу, заданную с помощью `SELECT`-конструкции.

Операция обновления значений в строках таблицы `UPDATE` также является групповой. Условие определяет, к каким строкам требуется применить операцию обновления. В условии, аналогично `DELETE`, можно использовать `SELECT`-конструкции.

В SQL большое внимание уделяется обеспечению целостности данных при выполнении операций обновления. Предусмотрена возможность учета специальных

ограничений целостности. Любые операции, нарушающие такие ограничения, отклоняются.

Наиболее часто используемым примером ограничения целостности является ограничение на диапазон допустимых значений в таблицах. Очень часто значения в таблице являются корректными только в том случае, когда они присутствуют в одной или нескольких других таблицах, логически связанных между собой.

При выполнении удаления или обновления строк для обеспечения целостности данных иногда необходимо выполнять определенные сопутствующие операции в других логически связанных таблицах. Например, удаление строк в одной таблице может сопровождаться удалением связанных строк в одной или нескольких других таблицах. Может также возникнуть необходимость заменить определенные значения связанных строк другой таблицы на неопределенные значения. При этом такие действия могут выполняться рекурсивно для достаточно сложных многотабличных структур.

Такого рода ограничения целостности определяются при создании отдельных таблиц и определении структуры базы данных.

Группу операций модификации данных, имеющих логически законченный смысл, после полного выполнения которых база данных останется корректной, называют *транзакцией* (см. *главу 9*). В SQL предусмотрены средства управления транзакциями, позволяющие отслеживать выполнение транзакций, обрабатывать возникающие ошибки и координировать обработку базы данных несколькими приложениями или пользователями в параллельном режиме.

Утверждение `COMMIT` означает удачное окончание текущей транзакции и начало новой. Утверждение `ROLLBACK` указывает на необходимость выполнения обратного отката, т. е. автоматического восстановления состояния базы данных на момент начала транзакции.

В большинстве случаев координация работы в многопользовательском режиме выполняется с помощью механизма блокировок монопольного захвата некоторой части базы данных. Выполнять блокировки можно автоматически, блокируя данные некоторой транзакцией, как только к ним происходит обращение, и освобождать их при обработке `COMMIT` и `ROLLBACK`.

В SQL можно блокировать таблицы в монопольном режиме (чтение и запись со стороны других транзакций откладываются до момента окончания транзакции) или в режиме разделения (откладываются только обновления со стороны других транзакций).

8.2. Предложение *DELETE*

Предложение DELETE имеет формат

```
DELETE FROM { имя_таблицы | ONLY (имя_таблицы) }
[ { WHERE условие_поиска | WHERE CURENT OF имя_курсора } ];
```

и позволяет удалить содержимое всех строк указанной таблицы (при отсутствии WHERE-фразы) или тех ее строк, которые определяются WHERE-фразой.

`FROM имя_таблицы`

Указывается таблица (*имя_таблицы*), из которой будут удаляться строки.

`ONLY (имя_таблицы)`

Запрещается каскадное распространение удаления записей на подтаблицы целевой таблицы или представления.

`WHERE условие_поиска`

Устанавливается поисковый критерий с использованием одного или нескольких *условий_поиска*, которые обеспечивают удаление только указанных строк.

`WHERE CURENT OF имя_курсора`

Удаляет текущую запись в объявленном и открытом курсоре (см. *разд. 17.7*) с именем *имя_курсора*.

8.2.1. Удаление единственной записи

Удалить поставщика с Код_поставщика=7.

```
DELETE
FROM    Поставщики
WHERE   Код_поставщика=7;
```

Если таблица Поставки содержит в момент выполнения этого предложения какие-либо поставки для поставщика с Код_поставщика=7, то такое удаление нарушит непротиворечивость базы данных. К сожалению, нет операции удаления, одновременно воздействующей на несколько таблиц. Однако в некоторых СУБД реализованы механизмы поддержания целостности (см. *разд. 2.4* и *главу 14*), позволяющие отменить некорректное удаление или каскадировать удаление в нескольких таблицах.

8.2.2. Удаление множества записей

Удалить все поставки:

```
DELETE
FROM    Поставки;
```

Удалить все мясные блюда:

```
DELETE FROM Блюда
WHERE  Основа = 'Мясо';
```

8.2.3. Удаление с вложенным подзапросом

Удалить все поставки для поставщика из Паневежиса:

```
DELETE
FROM   Поставки
WHERE  Код_поставщика IN
     (SELECT  Код_поставщика
      FROM    Поставщики
      WHERE   Город = 'Паневежис');
```

8.3. Предложение *INSERT*

Предложение INSERT имеет формат

```
INSERT INTO [ ONLY ] {имя_таблицы | имя_представления}
[ ( столбец [,...] ) ]
{ (DEFAULT VALUES | VALUES (скалярное_выражение [,...] ) |
                              предложение_SELECT );
```

и позволяет записать строки в указанную таблицу (представление). Здесь:

ONLY

Запрещает вставлять в подтаблицы значения, которые вставляются в таблицу *имя_таблицы*.

```
{ имя_таблицы | имя_представления } [ ( столбец [,...] ) ]
```

Объявляет обновляемую целевую таблицу или представление, в которое будут вставляться данные. Список столбцов можно не указывать, если данные будут вставляться во все столбцы и их порядок соответствует порядку столбцов в описании таблицы.

DEFAULT VALUES

В таблицу вставляются все значения, определяемые параметром DEFAULT столбца (значения по умолчанию), если таковые существуют, в противном случае вставляются пустые значения (NULL).

```
VALUES (скалярное_выражение [,...] )
```

Указываются *скалярные_выражения*, значения которых будут вставляться в целевую таблицу. Количество вставляемых *скалярных_выражений* должно

в точности совпадать с количеством столбцов в списке столбцов. Более того, их значения должны быть совместимы по типу и размеру со столбцами целевой таблицы. Каждое *скалярное_выражение* в списке соответствует столбцу в списке столбцов с тем же порядковым номером. Таким образом, данные из первого *скалярного_выражения* попадают в первый столбец, данные из второго — во второй и т. д. до тех пор, пока все столбцы не будут заполнены. При желании можно использовать ключевое слово DEFAULT, чтобы вставить в столбец значение, заданное по умолчанию, или NULL, чтобы вставить пустое значение.

скалярное_выражение

Это любое одиночное значение, например строковая константа или числовое значение, скалярная функция или скалярный подзапрос.

предложение_SELECT

Строки, извлекаемые данным предложением SELECT, вставляются в целевую таблицу или представление. Значения, извлекаемые SELECT, должны прямо соответствовать столбцам, указанным в списке столбцов. На целевую таблицу или представление нельзя ссылаться во фразе FROM или JOIN предложения SELECT.

8.3.1. Вставка единственной записи в таблицу

Добавить в таблицу Блюда блюдо:

```
Шашлык (Код_блюда - 34, Блюдо - Шашлык, В - 3, Основа - Мясо,
Выход - 150)
```

при неизвестной пока трудоемкости приготовления этого блюда.

```
INSERT
INTO    Блюда (БЛ, Блюдо, В, Основа, Выход)
VALUES (34, 'Шашлык', 3, 'Мясо', 150);
```

Создается новая запись для блюда с номером 34, с неопределенным значением в столбце Труд.

Порядок полей в INSERT не обязательно должен совпадать с порядком полей, в котором они определялись при создании таблицы. Вполне допустима и такая версия предыдущего предложения:

```
INSERT
INTO   Блюда (Основа, В, Блюдо, БЛ, Выход)
VALUES ('Мясо', 3, 'Шашлык', 34, 150);
```

При известной трудоемкости приготовления шашлыка (например, 5 коп.) сведения о нем можно ввести с помощью укороченного предложения:

```
INSERT
INTO   Блюда
VALUES (34, 'Шашлык', 3, 'Мясо', 150, 5);
```

в котором должен соблюдаться строгий порядок перечисления вводимых значений. Так как, не имея перечня загружаемых столбцов, СУБД может использовать лишь перечень, который определен при создании модифицируемой таблицы.

В предыдущих примерах проводилась модификация стержневой сущности, т. е. таблицы с первичным ключом `Код_блюда` (см. *разд. 2.4*). Почти все СУБД имеют механизмы для предотвращения ввода неуникального первичного ключа, например, ввода "Шашлыка" под номером, меньшим 34. А как быть с ассоциациями или другими таблицами, содержащими внешние ключи?

Пусть, например, потребовалось добавить в рецепт блюда Салат летний (`Код_блюда=1`) немного (15 г) лука (`Код_продукта=10`), и мы воспользовались предложением

```
INSERT
INTO   Состав (Код_блюда, Код_продукта, Вес)
VALUES (1, 10, 15);
```

Подобно операции `DELETE` операция `INSERT` может нарушить непротиворечивость базы данных. Если не принять специальных мер, то СУБД не проверяет, имеется ли в таблице `Блюда` блюдо с первичным ключом `Код_блюда=1` и в таблице `Продукты` — продукт с первичным ключом `Код_продукта=10`. Отсутствие любого из этих значений породит противоречие: в базе появится ссылка на несуществующую запись. Проблемы, возникающие при использовании внешних ключей, подробно рассмотрены в *главе 14*, а здесь отметим, что все "приличные" СУБД имеют механизмы для предотвращения ввода записей со значениями внешних ключей, отсутствующих среди значений соответствующих первичных ключей.

8.3.2. Вставка множества записей

Создать временную таблицу `К_меню`, содержащую калорийность и стоимость всех блюд, которые можно приготовить из имеющихся продуктов. (Эта таблица будет использоваться шеф-поваром для составления меню на следующий день.)

Для создания описания временной таблицы можно, например, воспользоваться предложением

```
CREATE TABLE К_меню
 ( Код_вида NUMBER(2),
   Блюдо VARCHAR2(16),
   Калор_блюда NUMBER(4),
   Стоим_блюда NUMBER(4,2)
 )';
```

а для ее загрузки данными, предложением `INSERT` с вложенным подзапросом:

```
INSERT
INTO   К_меню
SELECT Блюда.Код_вида, Блюдо,
  ROUND(SUM(((Белки+Углев)*4.1+Жиры*9.3) * Вес/1000)) Колор_блюда,
  ROUND((SUM(Стоимость/К_во*Вес/1000) + MIN(Труд/100))*10,2) Стоим_блюда
FROM   Блюда, Виды_блюд, Состав, Продукты, Наличие
WHERE  Блюда.Код_блюда     = Состав. Код_блюда
AND    Состав.Код_продукта = Продукты.Код_продукта
AND    Состав.Код_продукта = Наличие.Код_продукта
AND    Блюда.Код_вида      = Виды_блюд.Код_вида
AND    Блюда.Код_блюда NOT IN
         (SELECT   Код_блюда
          FROM  Состав
          WHERE Код_продукта IN
            (SELECT Код_продукта
             FROM   Наличие
             WHERE  К_во = 0))
GROUP BY Блюда.Код_вида, Блюдо
ORDER BY Блюда.Код_вида, Колор_блюда;
```

В этом запросе предложение `SELECT` выполняется так же, как обычно, но результат не выводится на экран, а копируется в таблицу `К_меню`. Теперь с этой копией можно работать как с обычной базовой таблицей (`Блюда`, `Продукты` и пр.), т. е. выбирать из нее данные на экран или принтер, обновлять в ней данные и т. п. Никакая из этих операций не будет оказывать влияния на исходные данные (например, изменение в ней названия блюда Салат летний на Салат весенний не приведет к подобному изменению в таблице `Блюда`, где сохранится старое название). Так как это может привести к противоречиям, то подобные временные таблицы уничтожают после их использования. Поэтому программа, обслуживающая шеф-повара, должна исполнять предложение `DROP TABLE К_меню` после того, как будет закончено составление меню.

8.4. Предложение *UPDATE*

Предложение UPDATE имеет формат

```
UPDATE [ ONLY ] { имя_таблицы | имя_представления }
SET {{имя_столбца = { DEFAULT | NULL | скалярное_выражение },
 имя_столбца = { DEFAULT | NULL | скалярное_выражение } [,...] }
[ WHERE условие_поиска | WHERE CURENT OF имя_курсора ];
```

и позволяет изменить данные в существующей таблице. (Соблюдайте осторожность при использовании UPDATE без фразы WHERE, поскольку при этом будут затронуты все строки таблицы.)

ONLY

Запрещает распространять обновление на подтаблицы целевой таблицы или представления.

имя_таблицы | *имя_представления*

Обновляемая целевая таблица или представление. Обновление представлений подчиняется особым правилам (см. *разд. 7.3.4*).

SET

Столбцу или строке присваивается определенное значение.

имя_столбца

Используется вместе с фразой SET (например, SET Блюдо = 'Картофель, запеченный с грибами в сметанном соусе'). Позволяет присваивать столбцу определенное значение. В одном UPDATE можно обновлять значения в нескольких столбцах, но нельзя обновлять значение одного столбца несколько раз.

DEFAULT

Для столбца устанавливается значение, заданное по умолчанию при определении таблицы.

скалярное_выражение

Столбцу присваивается любое одиночное значение, например строковая константа или числовое значение, скалярная функция или скалярный подзапрос.

WHERE *условие_поиска*

Устанавливается поисковый критерий с использованием одного или нескольких *условий_поиска*, которые обеспечивают обновление только указанных строк.

WHERE CURENT OF *имя_курсора*

Обновляет текущую запись в объявленном и открытом курсоре (см. *разд. 17.7*) с именем *имя_курсора*.

8.4.1. Обновление единственной записи

Изменить название блюда с кодом `Код_блюда=5` на Форшмак, увеличить его выход на 30 г и установить `NULL`-значение в столбец `Труд`.

```
UPDATE Блюда
SET    Блюдо = 'Форшмак', Выход = (Выход+30), Труд = NULL
WHERE  Код_блюда = 5;
```

8.4.2. Обновление множества записей

Утроить цену всех продуктов таблицы поставки кроме кофе (`Код_продукта=17`).

```
UPDATE Поставки
SET    Цена = Цена*3
WHERE  Код_продукта <> 17;
```

8.4.3. Обновление с подзапросом

Установить равной нулю цену и количество продуктов для поставщиков из Паневежиса и Резекне.

```
UPDATE Поставки
SET    Цена = 0, К_во = 0
WHERE  Код_поставщика IN
   (SELECT Код_поставщика
    FROM   Поставщики
    WHERE  Город IN ('Паневежис', 'Резекне'));
```

8.4.4. Обновление нескольких таблиц

Пусть требуется изменить в базе данных `Код_продукта=13` на `Код_продукта=20`.

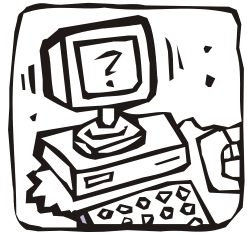
Так как столбец `Код_продукта` встречается в трех таблицах (`Продукты`, `Состав` и `Поставки`), а синтаксис `UPDATE` не позволяет одновременно обновлять более одной таблицы, то приходится выдавать три сходных запроса:

```
UPDATE Продукты
SET    Код_продукта = 20
WHERE  Код_продукта = 13;
```

```
UPDATE Состав
SET    Код_продукта = 20
WHERE  Код_продукта = 13;

UPDATE Поставки
SET    Код_продукта = 20
WHERE  Код_продукта = 13;
```

Это может привести к противоречию в базе данных (нарушению целостности по ссылкам), поскольку после выполнения первого предложения таблицы `Состав` и `Поставки` будут ссылаться на уже несуществующий продукт. База станет непротиворечивой только после выполнения третьего запроса.



Глава 9

Транзакции и параллелизм

9.1. Что такое транзакция

В *разд. 8.4.4* рассматривался пример, в котором требовалось изменить `Код_продукта=13` на новое значение `Код_продукта=20` и для этого пришлось проводить последовательное изменение в трех таблицах.

```
UPDATE Продукты          UPDATE Состав            UPDATE Поставки

SET   Код_продукта=20    SET   Код_продукта=20    SET   Код_продукта=20

WHERE                    WHERE                    WHERE
Код_продукта=13;         Код_продукта=13;         Код_продукта=13;
```

Этот пример приведен здесь для иллюстрации того, что единственная, с точки зрения пользователя, операция может потребовать нескольких операций над базой данных. Более того, в ходе выполнения этих операций может нарушаться непротиворечивость базы данных. Например, в ней могут временно содержаться записи поставок, для которых не имеется соответствующих записей поставляемых продуктов. Положение не спасает и перестановка последовательности обновляемых таблиц. Противоречивость исчезнет только после выполнения всех обновлений, т. е. выполнения логической единицы работы — полной замены кода продукта в базе данных (независимо от количества таблиц, в которых встречается код продукта).

Теперь можно дать определение транзакции.

Транзакция, или *логическая единица работы*, — это в общем случае последовательность ряда таких операций, которые преобразуют некоторое непротиворечивое состояние базы данных в другое непротиворечивое состояние, не гарантируя сохранения непротиворечивости во все промежуточные моменты времени.

В литературе для объяснения транзакции обычно приводится следующий классический пример.

Необходимо перевести с банковского счета номер 5 на счет номер 7 сумму в 10 денежных единиц.

Этого можно достичь, например, такой последовательностью действий:

```
Начать транзакцию
   прочесть баланс на счету номер 5
   уменьшить баланс на 10 денежных единиц
   сохранить новый баланс счета номер 5
   прочесть баланс на счету номер 7
   увеличить баланс на 10 денежных единиц
   сохранить новый баланс счета номер 7
Окончить транзакцию
```

Эти действия представляют собой логическую единицу работы "перевод суммы между счетами", и таким образом, являются транзакцией. Если прервать данную транзакцию, к примеру, в середине, и не аннулировать все изменения, легко оставить владельца счета номер 5 без 10 единиц, тогда как владелец счета номер 7 их не получит.

Никто кроме пользователя, генерирующего ту или иную последовательность SQL-предложений, не может знать о том, когда может возникнуть противоречивое состояние базы данных и после выполнения каких SQL-предложений оно исчезнет, т. е. база данных вновь станет актуальной. Поэтому в большинстве СУБД создается механизм обработки транзакций, при инициировании которого все изменения данных рассматриваются как предварительные до тех пор, пока пользователь (реже система) не выдаст предложения:

- `COMMIT` (фиксировать), превращающее все предварительные обновления в окончательные ("зафиксированные");
- `ROLLBACK` (откат), аннулирующее все предварительные обновления.

Таким образом, транзакцией можно назвать последовательность SQL-предложений, расположенных между "точками синхронизации", учреждаемых в начале выполнения программы и издании `COMMIT` или `ROLLBACK`, и только в этих случаях. При этом следует иметь в виду, что возможен неявный `COMMIT` (существует режим `AUTOCOMMIT`, в котором система издает `COMMIT` после выполнения каждого SQL-предложения) и `ROLLBACK` (выполняемый при аварийном завершении программы).

Ясно теперь, что пользователь должен сам решать, включать ли механизм обработки транзакций и если включать, то где издавать `COMMIT` (`ROLLEBACK`), т. е. какие последовательности SQL-предложений являются транзакциями.

9.2. Предложения *COMMIT*, *ROLLBACK* и *SAVEPOINT*

Предложение COMMIT явным образом закрывает открытую транзакцию и делает изменения в базе данных постоянными. Открываться транзакции могут неявно, например, при выполнении предложений INSERT, DELETE или UPDATE, или явно, с помощью предложения START (в Oracle все транзакции запускаются неявно). В любом случае открытая транзакция закрывается явно предложением COMMIT.

Синтаксис этого предложения очень прост:

```
COMMIT [WORK]
```

Здесь необязательное слово WORK является пустым и не оказывает никакого влияния.

Предложение ROLLBACK возвращает транзакцию в ее исходное состояние или к определенной, заранее заданной точке сохранения (SAVEPOINT). Кроме того, это предложение закрывает все открытые курсоры (см. *разд. 17.7*).

Синтаксис предложения имеет вид:

```
ROLLBACK [WORK]
[TO SAVEPOINT имя_точки_сохранения]
```

Здесь, как и в предложении COMMIT, необязательное слово WORK является пустым и не оказывает никакого влияния.

```
[TO SAVEPOINT имя_точки_сохранения]
```

Позволяет не отменять всю транзакцию, а откатить ее к указанной точке сохранения (т. е. выполнить частичный откат). Параметр *имя_точки_сохранения* может представлять собой постоянное выражение или переменную. Если TO SAVEPOINT опущено, то закрываются все курсоры, в противном случае закрываются только те курсоры, которые были открыты соответствующим предложением SAVEPOINT.

Предложение SAVEPOINT разделяет транзакцию на логические точки сохранения. В одной транзакции может быть несколько точек сохранения.

Синтаксис этого предложения имеет вид:

```
SAVEPOINT имя_точки_сохранения
```

В текущей транзакции устанавливается точка сохранения с именем *имя_точки_сохранения*.

9.3. Многопользовательский режим работы

9.3.1. Параллелизм транзакций

Поддержание механизма транзакций — показатель уровня развитости СУБД и основа обеспечения целостности базы данных. Транзакции также составляют основу изолированности в многопользовательских системах, где с одной базой данных параллельно могут работать несколько пользователей и (или) прикладных программ. Одна из основных задач СУБД — обеспечение изолированности, т. е. создание такого режима функционирования, при котором каждому пользователю казалось бы, что база данных доступна только ему. Такую задачу СУБД принято называть параллелизмом транзакций.

Большинство выполняемых действий производится в теле транзакций. По умолчанию каждая команда выполняется как самостоятельная транзакция. Как было показано ранее, при необходимости пользователь может явно указать ее начало и конец, чтобы иметь возможность включить в нее несколько команд.

При выполнении транзакции система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в транзакцию. В частности, разработано четыре правила, известные как требования ACID (Atomicity, Consistency, Isolation, Durability — неделимость, согласованность, изолированность, устойчивость), гарантирующих правильность и надежность работы системы.

- Транзакция неделима в том смысле, что представляет собой единое целое. Все ее компоненты либо имеют место, либо нет. Не бывает частичной транзакции. Если может быть выполнена лишь часть транзакции, она отклоняется.
- Транзакция является согласованной, потому что не нарушает бизнес-логику и отношения между элементами данных. Это свойство очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество транзакций от разных систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.
- Транзакция всегда изолирована, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих транзакций — это свойство называется *сериализуемостью* и означает, что транзакции в последовательности независимы.
- Транзакция устойчива. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до начала транзакции)

состояние, т. е. происходит фиксация транзакции, означающая, что ее действие постоянно даже при сбое системы. При этом подразумевается некая форма хранения информации в постоянной памяти как часть транзакции.

Указанные ранее правила выполняет сервер. Программист лишь выбирает нужный уровень изоляции, заботится о соблюдении логической целостности данных и бизнес-правил. На него возлагаются обязанности по созданию эффективных и логически верных алгоритмов обработки данных. Он решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций. Следует по возможности использовать небольшие транзакции, т. е. включающие как можно меньше команд и изменяющие минимум данных. Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей.

9.3.2. Блокировки

Повышение эффективности работы при использовании небольших транзакций связано с тем, что при выполнении транзакции сервер накладывает на данные блокировки.

Блокировкой называется временное ограничение на выполнение некоторых операций обработки данных. Блокировка может быть наложена как на отдельную строку таблицы, так и на всю базу данных. Управлением блокировками на сервере занимается менеджер блокировок, контролирующий их применение и разрешение конфликтов. Транзакции и блокировки тесно связаны друг с другом. Транзакции накладывают блокировки на данные, чтобы обеспечить выполнение требований ACID. Без использования блокировок несколько транзакций могли бы изменять одни и те же данные.

Блокировка представляет собой метод управления параллельными процессами, при котором объект базы данных не может быть модифицирован без ведома транзакции. Производится блокирование доступа к объекту со стороны других транзакций, чем исключается непредсказуемое изменение объекта.

Различают два вида блокировки:

- блокировка записи — транзакция блокирует строки в таблицах таким образом, что запрос другой транзакции к этим строкам будет отменен;
- блокировка чтения — транзакция блокирует строки так, что запрос со стороны другой транзакции на блокировку записи этих строк будет отвергнут, а на блокировку чтения — принят.

В СУБД используют протокол доступа к данным, позволяющий избежать проблемы параллелизма. Его суть заключается в следующем:

- транзакция, результатом действия которой на строку данных в таблице является ее извлечение, обязана наложить блокировку чтения на эту строку;
- транзакция, предназначенная для модификации строки данных, накладывает на нее блокировку записи;
- если запрашиваемая блокировка на строку отвергается из-за уже имеющейся блокировки, то транзакция переводится в режим ожидания до тех пор, пока блокировка не будет снята;
- блокировка записи сохраняется вплоть до конца выполнения транзакции.

Решение проблемы параллельной обработки базы данных заключается в том, что строки таблиц блокируются, а последующие транзакции, модифицирующие эти строки, отвергаются и переводятся в режим ожидания. В связи со свойством сохранения целостности базы данных транзакции являются подходящими единицами изолированности пользователей. Действительно, если каждый сеанс взаимодействия с базой данных реализуется транзакцией, то пользователь начинает с того, что обращается к согласованному состоянию базы данных — состоянию, в котором она могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Если бы в СУБД не были реализованы механизмы блокирования, то при одновременном чтении и изменении одних и тех же данных несколькими пользователями могли бы возникнуть следующие проблемы одновременного доступа:

- проблема последнего изменения возникает, когда несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна, т. к. каждая последующая транзакция перезапишет изменения, сделанные предыдущей. Выход из этой ситуации заключается в последовательном внесении изменений;
- проблема "грязного" чтения возможна в том случае, если пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию. Для исключения подобных проблем необходимо производить считывание данных после окончания всех изменений;
- проблема неповторяемого чтения является следствием неоднократного считывания транзакцией одних и тех же данных. Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при

повторном чтении первая транзакция получит уже иной набор данных, что приведет к нарушению их целостности или логической несогласованности;

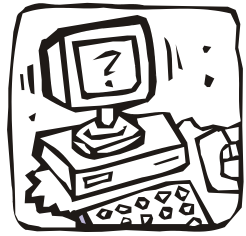
- проблема чтения фантомов появляется после того, как одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

Для решения перечисленных проблем в специально разработанном стандарте определены четыре уровня блокирования. Уровень изоляции транзакции определяет, могут ли другие (конкурирующие) транзакции вносить изменения в данные, измененные текущей транзакцией, а также может ли текущая транзакция видеть изменения, произведенные конкурирующими транзакциями, и наоборот. Каждый последующий уровень поддерживает требования предыдущего и налагает дополнительные ограничения:

- уровень 0 — запрещение "загрязнения" данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции;
- уровень 1 — запрещение "грязного" чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой;
- уровень 2 — запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии;
- уровень 3 — запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющиеся строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

ЧАСТЬ IV

ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ



Глава 10

Введение в проектирование

10.1. Цели проектирования

Только небольшие организации могут обобществить данные в одной полностью интегрированной базе данных. Чаще всего администратор баз данных (даже если это группа лиц) практически не в состоянии охватить и осмыслить все информационные требования сотрудников организации (т. е. будущих пользователей системы). Поэтому информационные системы больших организаций содержат множество баз данных, нередко распределенных между несколькими взаимосвязанными компьютерами различных подразделений. (Так в больших городах создается не одна, а несколько овощных баз, расположенных в разных районах.)

Отдельные базы данных могут объединять все данные, необходимые для решения одной или нескольких прикладных задач, или данные, относящиеся к какой-либо предметной области (например, финансам, учебному процессу, кулинарии и т. п.). Первые обычно называют *прикладными базами данных*, а вторые — *предметными базами данных* (соотносящимся с предметами организации, а не с ее информационными приложениями). Первые можно сравнить с базами материально-технического снабжения или отдыха, а вторые — с овощными и обувными базами.

Предметные базы данных позволяют обеспечить поддержку любых текущих и будущих приложений, поскольку набор их элементов данных включает наборы элементов данных прикладных баз данных. Вследствие этого предметные базы данных создают основу для обработки неформализованных, изменяющихся и неизвестных запросов и приложений (приложений, для которых невозможно заранее определить требования к данным). Такая гибкость и приспосабливаемость позволяет создавать на основе предметных баз данных достаточно стабильные информационные системы, т. е. системы, в которых

большинство изменений можно осуществить без вынужденного переписывания старых приложений.

Основывая же проектирование баз данных на текущих и предвидимых приложениях, можно существенно ускорить создание высокоэффективной информационной системы, т. е. системы, структура которой учитывает наиболее часто встречающиеся пути доступа к данным. Поэтому прикладное проектирование до сих пор привлекает некоторых разработчиков. Однако по мере роста числа приложений таких информационных систем быстро увеличивается число прикладных баз данных, резко возрастает уровень дублирования данных и повышается стоимость их ведения.

Таким образом, каждый из рассмотренных подходов к проектированию воздействует на результаты проектирования в разных направлениях. Желание достичь и гибкости, и эффективности привело к формированию методологии проектирования, использующей как предметный, так и прикладной подходы. В общем случае предметный подход используется для построения первоначальной информационной структуры, а прикладной — для ее совершенствования с целью повышения эффективности обработки данных.

При проектировании информационной системы необходимо провести анализ целей этой системы и выявить требования к ней отдельных пользователей (сотрудников организации). Сбор данных начинается с изучения сущностей организации и процессов, использующих эти сущности. Сущности группируются по "сходству" (частоте их использования для выполнения тех или иных действий) и по количеству ассоциативных связей между ними (самолет — пассажир, преподаватель — дисциплина, студент — сессия и т. д.). Сущности или группы сущностей, обладающие наибольшим сходством и (или) с наибольшей частотой ассоциативных связей, объединяются в предметные базы данных. (Нередко сущности объединяются в предметные базы данных без использования формальных методик — по "здравому смыслу".) Для проектирования и ведения каждой предметной базы данных (нескольких баз данных) назначается администратор базы данных, который далее занимается детальным проектированием базы.

Далее будут рассматриваться вопросы, связанные с проектированием отдельных реляционных предметных баз данных.

Основная цель проектирования базы данных — это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Так называемый

"чистый" проект базы данных ("каждый факт в одном месте") можно создать, используя методологию нормализации отношений. И хотя нормализация должна использоваться на завершающей проверочной стадии проектирования базы данных, мы начнем обсуждение вопросов проектирования с рассмотрения причин, которые заставили Кодда создать основы теории нормализации.

Прежде чем приступить к подробному изложению, сделаем несколько предварительных замечаний.

Следует заметить, что речь здесь пойдет о *логическом* (или *концептуальном)* проектировании, а не о разработке физического проекта. Это вовсе не значит, что физическое проектирование не имеет большого значения. Наоборот, создание физического проекта играет очень важную роль. Тем не менее, необходимо сделать следующие оговорки.

- Физическое проектирование может рассматриваться как отдельный последующий этап. Иначе говоря, для "правильного" проектирования базы данных, прежде всего, необходимо создать ее приемлемый логический (т. е. реляционный) проект и лишь затем в качестве отдельного этапа разработки выполнить отображение этого логического проекта на определенные физические структуры, поддерживаемые конкретной СУБД. Другими словами, как отмечалось ранее, физический проект создается на базе логического проекта и никак иначе.
- Физическое проектирование по определению является зависимым от специфики конкретной целевой СУБД. Логический проект, наоборот, совершенно независим от СУБД, и для его реализации могут использоваться некоторые строгие теоретические принципы.

К сожалению, на практике часто случается так, что решения, принятые в процессе физической реализации проекта, могут оказывать существенное обратное влияние на его логический уровень. Иначе говоря, может потребоваться выполнить нескольких итераций цикла "логическое проектирование — физическое проектирование" и пойти на определенные компромиссы. Тем не менее, изложение материала в этой части ведется исходя из того, что предварительно необходимо создать логический проект без учета особенностей его будущей физической реализации (например, без учета требований к определенному уровню производительности).

После всего сказанного следует подчеркнуть, что проектирование базы данных еще во многом продолжает оставаться скорее искусством, чем наукой. Конечно, существуют некоторые научные принципы такого проектирования, которые будут изложены в следующих главах. Однако при проектировании

базы данных возникает множество других проблем, которые не всегда можно решить, руководствуясь этими правилами.

Следует отметить некоторые допущения, используемые в дальнейшем изложении.

- Проектирование базы данных заключается не только в создании правильной структуры данных. Еще одной и, вероятно, более важной задачей является обеспечение целостности данных.
- Далее в большинстве случаев проектирование рассматривается *независимо от приложения*. Иначе говоря, интерес представляют сами данные, а не то, как они будут *использоваться*. Независимость от приложения в этом смысле желательна по той простой причине, что в момент проектирования базы данных обычно еще неизвестны все возможные способы использования ее данных. Таким образом, необходимо, чтобы созданный проект был *стабильным*, т. е. оставался работоспособным даже при возникновении в приложениях новых (т. е. неизвестных на момент создания исходного макета) требований к данным. Следуя этим допущениям, можно сказать, что здесь обсуждается создание *концептуальной схемы*, т. е. абстрактного логического проекта, не зависящего от аппаратного обеспечения, операционной системы, целевой СУБД, языка программирования, требований пользователей и т. д.

10.2. Универсальное отношение

Предположим, что проектирование базы данных "COOK" (табл. 3.1—3.10) начинается с выявления атрибутов и подбора данных, образец которых (часть блюд изготовленных и реализованных 15/05/1989 г.) показан в табл. 10.1. (Отметим, что в табл. 10.1 добавлен столбец `СТРАНА`, отсутствующий в табл. 3.4.)

Этот вариант таблицы "COOK" не является отношением, так как большинство ее строк содержат множественные значения. Для придания таким данным формы отношения необходимо реконструировать таблицу. Наиболее просто это сделать с помощью простого процесса вставки, результат которого показан в табл. 10.2. Однако такое преобразование приводит к возникновению большого объема избыточных данных.

Таблица 10.1. Основные данные, необходимые для создания базы данных "COOK"

БЛЮДО	ВИД	ОСНОВА	ВЫХОД	ПРОДУКТ	ПОСТАВЩИК	СТАТУС	СТРАНА	ГОРОД	АДРЕС
Бастурма	Горячее	Мясо	300,0	Говядина	ПОРТОС	кооператив	Латвия	Резекне	Садовая,27
				Зелень	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
				Лук	УРОЖАЙ	кооптторг	Россия	Луга	Песчаная,19
				Масло	ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
				Помидоры	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
Салат мясной	Закуска	Мясо	200,0	Говядина	ПОРТОС	кооператив	Латвия	Резекне	Садовая,27
				Зелень	ШУШАРЫ	совхоз	Россия	Пушкин	Новая, 17
				Майонез	ТУЛЬСКИЙ	универсам	Россия	Ленинград	Тульский,5
				Помидоры	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
				Яйца	КОРЮШКА	кооператив	Эстония	Йыхви	Нарвское ш.,64
Суп харчо	Суп	Мясо	500,0	Говядина	ОГУРЕЧИК	ферма	Латвия	Паневежис	Укмерге,15
				Зелень	ШУШАРЫ	совхоз	Россия	Пушкин	Новая, 17
				Лук	ЛЕТО	агрофирма	Россия	Ленинград	Пулковская,8
				Масло	ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
				Помидоры	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3

Таблица 10.2. Универсальное отношение СООК

БЛЮДО	ВИД	ОСНОВА	ВЫХОД	ПРОДУКТ	ПОСТАВЩИК	СТАТУС	СТРАНА	ГОРОД	АДРЕС
Бастурма	Горячее	Мясо	300,0	Говядина	ПОРТОС	кооператив	Латвия	Резекне	Садовая,27
Бастурма	Горячее	Мясо	300,0	Зелень	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
Бастурма	Горячее	Мясо	300,0	Лук	УРОЖАЙ	кооптorг	Россия	Луга	Песчаная,19
Бастурма	Горячее	Мясо	300,0	Масло	ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
Бастурма	Горячее	Мясо	300,0	Помидоры	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
Салат мясной	Закуска	Мясо	200,0	Говядина	ПОРТОС	кооператив	Латвия	Резекне	Садовая,27
Салат мясной	Закуска	Мясо	200,0	Зелень	ШУШАРЫ	совхоз	Россия	Пушкин	Новая, 17
Салат мясной	Закуска	Мясо	200,0	Майонез	ТУЛЬСКИЙ	универсам	Россия	Ленинград	Тульский,5
Салат мясной	Закуска	Мясо	200,0	Помидоры	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
Салат мясной	Закуска	Мясо	200,0	Яйца	КОРЮШКА	кооператив	Эстония	Йыхви	Нарвское ш.,64
Суп харчо	Суп	Мясо	500,0	Говядина	ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
Суп харчо	Суп	Мясо	500,0	Зелень	ШУШАРЫ	совхоз	Россия	Пушкин	Новая, 17
Суп харчо	Суп	Мясо	500,0	Лук	ЛЕТО	агрофирма	Россия	Ленинград	Пулковская,8
Суп харчо	Суп	Мясо	500,0	Масло	ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
Суп харчо	Суп	Мясо	500,0	Помидоры	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3

Таблица 10.2 представляет собой экземпляр корректного отношения. Его называют *универсальным отношением* проектируемой базы данных. В одно универсальное отношение включаются все представляющие интерес атрибуты, и оно может содержать все данные, которые предполагается размещать в базе данных в будущем. Для малых баз данных (включающих не более 15 атрибутов) универсальное отношение может использоваться в качестве отправной точки при проектировании базы данных.

10.3. Почему проект базы данных может быть плохим?

Начинающий проектировщик будет использовать универсальное отношение `COOK` (табл. 10.2) в качестве завершенной базы данных (естественно после дополнения этого отношения рядом столбцов, которые были из него изъяты для уменьшения ширины таблицы). Действительно, зачем разбивать отношение `COOK` на несколько более мелких отношений (см. например, табл. 3.1—3.10), если оно заключает в себе все данные?

А разбивать надо потому, что при использовании универсального отношения возникает несколько проблем.

- *Избыточность.* Данные практически всех столбцов многократно повторяются. Повторяются и некоторые наборы данных: `Блюдо` — `Вид` — `Основа` — `Выход`, `Поставщик` — `Статус` — `Страна` — `Город` — `Адрес` — `Цена`. Нежелательно и повторение рецептов (не показанных в табл. 10.2), некоторые из них намного больше рецепта "Лобио" (см. рис. 2.7). И уж совсем плохо, что все данные о блюде (включая рецепт) повторяются каждый раз, когда это блюдо включается в меню.
- *Потенциальная противоречивость (аномалии обновления).* Вследствие избыточности можно обновить адрес поставщика в одной строке, оставляя его неизменным в других. Если поставщик кофе сообщил о своем переезде по другому адресу, и была обновлена строка с продуктом кофе, то у поставщика Тульский появляется два адреса, один из которых не актуален. Следовательно, при обновлениях необходимо просматривать всю таблицу для нахождения и изменения всех подходящих строк.
- *Аномалии включения.* В базе данных не может быть записан новый поставщик (Няринга, Литва, Вильнюс), если поставляемый им продукт (Огурцы) не используется ни в одном блюде. Можно, конечно, поместить неопределенные значения в столбцы `Блюдо`, `Вид`, `Основа` и `Выход` для этого

поставщика. Но если появится блюдо, в котором используется этот продукт, не забудем ли мы удалить строку с неопределенными значениями?

По аналогичным причинам нельзя ввести и новый продукт (например, Баклажаны), который предлагает существующий поставщик (например, Сытный).

А как ввести новое блюдо, если в нем используется новый продукт (Крабы)?

БЛЮДА

БЛЮДО	ВИД	ОСНОВА	ВЫХОД
Суп харчо	Суп	Мясо	500,0
Бастурма	Горячее	Мясо	300,0
Кофе черный	Напиток	Кофе	100,0
...			

РЕЦЕПТЫ

БЛЮДО	РЕЦЕПТ
Суп харчо	Грудинку говядины наре
Бастурма	Мясо нарезать кубиками
Кофе черный	Кофеварку или кастрюлю
...	

ПРОДУКТЫ

ПРОДУКТ	БЕЛКИ	ЖИРЫ	УГЛЕВ
Говядина	189,0	124,0	0,0
Лук	17,0	0,0	95,0
Масло	60,0	825,0	90,0
Помидоры	6,0	0,0	42,0
Зелень	9,0	0,0	20,0
Рис	70,0	6,0	773,0
Кофе	127,0	36,0	9,0
...			

СОСТАВ

БЛЮДО	ПРОДУКТ	ВЕС
Суп харчо	Говядина	80
Суп харчо	Лук	30
Суп харчо	Масло	15
Суп харчо	Зелень	15
Суп харчо	Помидоры	25
Суп харчо	Рис	35
Бастурма	Говядина	180
...		

ПОСТАВЩИКИ

ПОСТАВЩИК	СТАТУС	СТРАНА	ГОРОД	АДРЕС
ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
УРОЖАЙ	кооптторг	Россия	Луга	Песчаная,19
ШУШАРЫ	совхоз	Россия	Пушкин	Новая, 17
СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
ТУЛЬСКИЙ	универсам	Россия	Ленинград	Тульский,5
...				

ПОСТАВКИ

ПРОДУКТ	ПОСТАВЩИК	ЦЕНА	К_ВО	ДАТА
Говядина	ОГУРЕЧИК	4,20	70	10.05.1989
Лук	УРОЖАЙ	0,50	130	14.05.1989
Масло	ОГУРЕЧИК	4,00	250	10.05.1989
Рис	ТУЛЬСКИЙ	0,88	150	09.05.1989
Помидоры	СЫТНЫЙ	1,50	50	14.05.1989
Зелень	СЫТНЫЙ	3,00	10	10.05.1989
Зелень	ШУШАРЫ	2,50	20	14.05.1989
Кофе	ТУЛЬСКИЙ	4,50	50	09.05.1989
...				

Рис. 10.1. Преобразование универсального отношения COOK (первый вариант)

- *Аномалии удаления.* Обратная проблема возникает при необходимости удаления всех продуктов, поставляемых данным поставщиком, или всех блюд, использующих эти продукты. При таких удалениях будут утрачены сведения о таком поставщике.

Многие проблемы этого примера исчезнут, если выделить в отдельные таблицы сведения о блюдах, рецептах, продуктах и их поставщиках, а также создать связующие таблицы `Состав` и `Поставки` (рис. 10.1).

Включение. Простым добавлением строк (`Поставщики`; `Няринга, Вильнюс`) и (`Поставки`; `Огурцы, Няринга, Вильнюс, 0,50, 40`) можно ввести информацию о новом поставщике. Аналогично можно ввести данные о новом продукте (`Продукты`; `Баклажаны, 240`) и (`Поставки`; `Баклажаны, Полесье, Украина, Киев, 50`).

Удаление. Удаление сведений о некоторых поставках или блюдах не приводит к потере сведений о поставщиках.

Обновление. В таблицах с рис. 10.1 все еще много повторяющихся данных, находящихся в связующих таблицах (`Состав` и `Поставки`). Следовательно, в данном варианте базы данных сохранилась потенциальная противоречивость: для изменения названия поставщика с Урожай на Полесье придется изменять не только строку таблицы `Поставщики`, но и множество строк таблицы `Поставки`. При этом не исключено, что в базе данных будут одновременно храниться: Полесье, Палесье, Няринга, Няренга и другие варианты названий.

Кроме того, повторяющиеся текстовые данные (такие как название блюда "Рулет из телячьей грудинки с сосисками и гарниром из разноцветного пюре" или продукта "Колбаса московская сырокопченая") существенно увеличивают объем хранимых данных.

Для исключения ссылок на длинные текстовые значения последние обычно нумеруют: нумеруют блюда в больших кулинарных книгах, товары (продукты) в каталогах и т. д. Воспользуемся этим приемом (использованием суррогатных ключей — см. *разд. 2.4*) для исключения избыточного дублирования данных и появления ошибок при копировании длинных текстовых значений (рис. 10.2). Теперь при изменении названия поставщика Урожай на Полесье исправляется единственное значение в таблице `Поставщики`. И даже если оно вводится с ошибкой (Палесье), то это не может повлиять на связь между поставщиками и продуктами (в связующей таблице `Поставки` используются номера поставщиков и продуктов, а не их названия).

БЛЮДА

БЛ	БЛЮДО	ВИД	ОСНОВА	ВЫХОД
1	Суп харчо	Суп	Мясо	500,0
2	Бастурма	Горячее	Мясо	300,0
3	Кофе черный	Напиток	Кофе	100,0

...

РЕЦЕПТЫ

БЛ	РЕЦЕПТ
1	Грудинку говядины нарез
2	Мясо нарезать кубиками
3	Кофеварку или кастрюлю

...

ПРОДУКТЫ

ПР	ПРОДУКТ	БЕЛКИ	ЖИРЫ	УГЛЕВ
1	Говядина	189,0	124,0	0,0
2	Лук	17,0	0,0	95,0
3	Масло	60,0	825,0	90,0
4	Помидоры	6,0	0,0	42,0
5	Зелень	9,0	0,0	20,0
6	Рис	70,0	6,0	773,0
7	Кофе	127,0	36,0	9,0

...

СОСТАВ

БЛ	ПР	ВЕС
1	1	80
1	2	30
1	3	15
1	5	15
1	4	25
1	6	35
2	1	180

...

ПОСТАВЩИКИ

ПС	ПОСТАВЩИК	СТАТУС	СТРАНА	ГОРОД	АДРЕС
1	ОГУРЕЧИК	ферма	Литва	Паневежис	Укмерге,15
2	УРОЖАЙ	кооптorг	Россия	Луга	Песчаная,19
3	ШУШАРЫ	совхоз	Россия	Пушкин	Новая, 17
4	СЫТНЫЙ	рынок	Россия	Ленинград	Сытнинская,3
5	ТУЛЬСКИЙ	универсам	Россия	Ленинград	Тульский,5

...

ПОСТАВКИ

ПР	ПС	ЦЕНА	К_ВО	ДАТА
1	1	4,20	70	10.05.1989
2	2	0,50	130	14.05.1989
3	1	4,00	250	10.05.1989
6	5	0,88	150	09.05.1989
4	4	1,50	50	14.05.1989
5	4	3,00	10	10.05.1989
5	3	2,50	20	14.05.1989
7	5	4,50	50	09.05.1989

...

Рис. 10.2. Преобразование универсального отношения COOK (второй вариант)

10.4. Процедура проектирования

Как уже отмечалось в *разд. 1.3*, проект базы данных должен начинаться с выбора предметной области (той части реального мира, данные о которой надо отразить в базе данных) и выявления требований к ней отдельных пользователей (сотрудников организации, для которых создается база данных).

Проектирование обычно поручается человеку (группе лиц) — *администратору данных* (АД). Объединяя частные представления о содержимом базы данных, полученные в результате опроса пользователей и (или) анализа их технических заданий, и свои представления о данных, которые могут потребоваться в будущих приложениях, АД сначала создает обобщенное неформальное описание создаваемой базы данных. Это выполненное с использованием текста (на естественном языке), образцов входных и выходных документов, математических формул, таблиц, графики и других средств, понятных потенциальным пользователям и всем людям, работающим над проектированием базы данных, и есть концептуальная модель данных, которую часто называют логической или *информационно-логической (инфологической)* моделью (см. *главу 2*).

Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных и от той СУБД, которая будет использоваться для построения и ведения базы данных. В конце концов, этой средой может быть память человека, а не ЭВМ. Поэтому концептуальная модель не должна изменяться до тех пор, пока какие-то изменения в реальном мире не потребуют изменения в ней некоторых определений, чтобы эта модель продолжала отражать предметную область.

10.4.1. Этапы проектирования базы данных

Основная цель проектирования базы данных — это сокращение избыточности хранимых данных, а следовательно, экономия объема используемой памяти, уменьшение затрат на многократные операции обновления избыточных копий и устранение возможности возникновения противоречий из-за хранения в разных местах сведений об одном и том же объекте. Так называемый, "чистый" проект базы данных ("Каждый факт в одном месте") можно создать, используя при тестировании проекта методологию нормализации отношений (см. *главу 11*). Начинать же создание "чистого" проекта необходимо, "забыв" об обеспечении просьб и пожеланий будущих пользователей, включая разработчиков и администраторов приложений, которые, как правило, хотят, чтобы база данных содержала таблицы с теми или иными результирующими данными, т. е. многочисленными дубликатами. Их просьбы будут в дальнейшем учтены на этапах оптимизации базы данных (см. *главу 16*) и проектирования приложений (см. *часть VI*).

Процесс проектирования состоит из трех основных этапов:

1. На основе информации, полученной при анализе, необходимо создать подробное описание предметной области, обращая особое внимание на требования к данным.

2. Построить *инфологическую модель базы данных* — обобщенное, не привязанное к каким-либо компьютерам и СУБД, описание предметной области (наборы данных, их типов, длин, связей и т. п.).
3. Выбрать СУБД, под управлением которой должна функционировать база данных, и создать *даталогическую (табличную) модель базы данных* — инфологическую модель, переведенную на язык выбранной СУБД.

В литературе по проектированию информационных систем (основой которых являются базы данных) обычно подробно рассматриваются различные подходы к организации проектирования и отмечается, что проектирование, как правило, не имеет четко выраженного начала и окончания и часто продолжается на этапах тестирования и реализации. Это иллюстрируется различными моделями жизненного цикла (рис. 10.3) и подробным обсуждением их достоинств и недостатков.

Также описывается целый ряд стандартов, регламентирующих жизненный цикл, а в некоторых случаях и процессы разработки.

Значительный вклад в теорию проектирования и разработки информационных систем внесла компания IBM, предложив еще в середине 1970-х годов методологию организационного планирования (Business System Planning, BSP). Метод структурирования информации с использованием матриц пересечения бизнес-процессов, функциональных подразделений, функций систем обработки данных (информационных систем), информационных объектов, документов и баз данных, предложенный в BSP, используется сегодня не только в IT-проектах, но и проектах по реинжинирингу бизнес-процессов, изменению организационной структуры. Важнейшие шаги процесса BSP, их последовательность (получить поддержку высшего руководства, определить процессы предприятия, определить классы данных, провести интервью, обработать и организовать данные интервью) можно встретить практически во всех формальных методиках, а также в проектах, реализуемых на практике.

Среди наиболее известных стандартов можно выделить:

- ГОСТ 34.601-90 — распространяется на автоматизированные системы и устанавливает стадии и этапы их создания. Кроме того, в стандарте содержится описание содержания работ на каждом этапе. Стадии и этапы работы, закрепленные в стандарте, в большей степени соответствуют каскадной модели жизненного цикла.
- ISO/IEC 12207:1995 — стандарт на процессы и организацию жизненного цикла. Распространяется на все виды заказного программного обеспечения. Стандарт не содержит описания фаз, стадий и этапов.

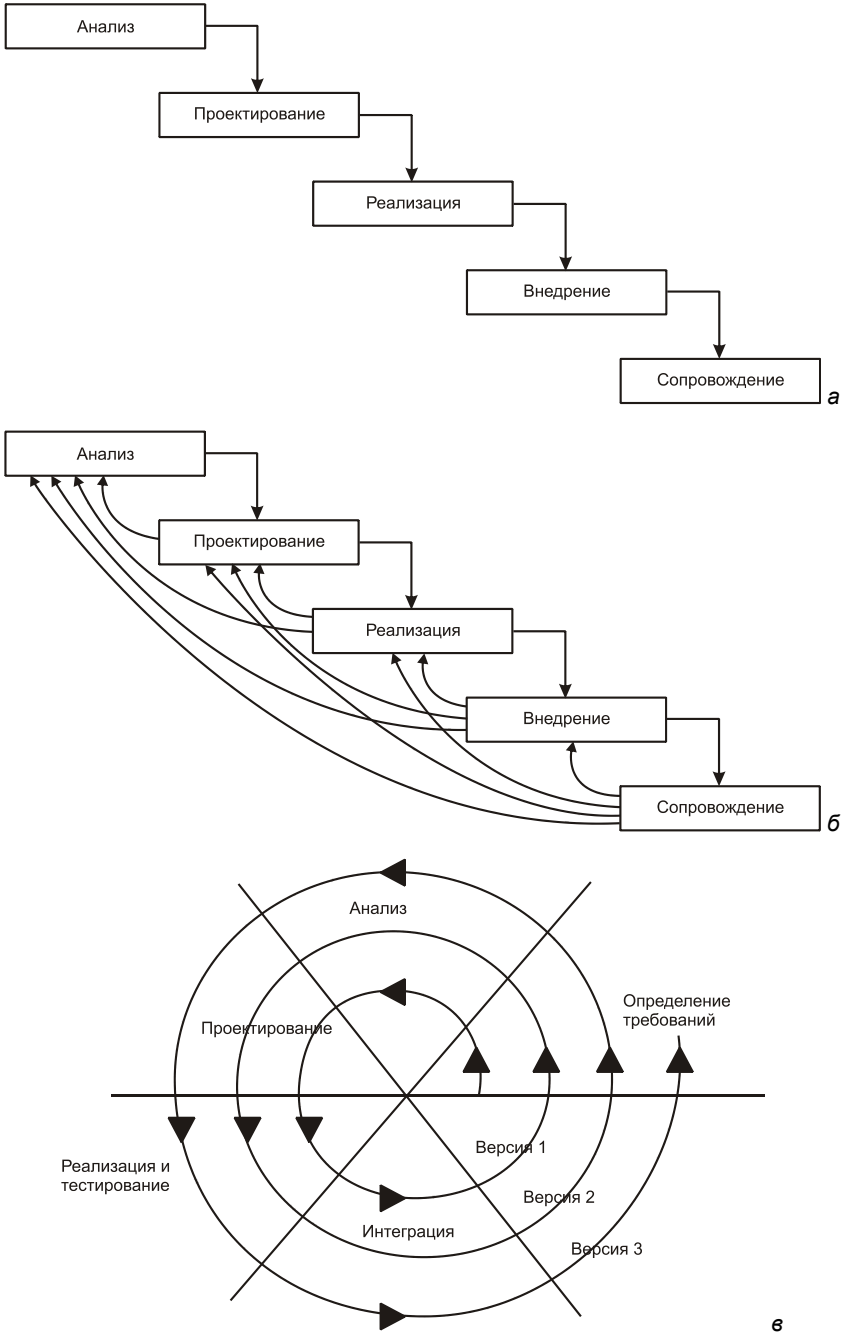


Рис. 10.3. Модели жизненного цикла: *а* — каскадная; *б* — поэтапная (водопад); *в* — спиральная

- Custom Development Method (методика Oracle) по разработке прикладных информационных систем — технологический материал, детализированный до уровня заготовок проектных документов, рассчитанных на использование в проектах с применением Oracle. Применяется CDM для классической модели жизненного цикла (предусмотрены все работы/задачи и этапы), а также для технологий "быстрой разработки" (Fast Track) или "облегченного подхода", рекомендуемых в случае малых проектов.
- Rational Unified Process (RUP) предлагает итеративную модель разработки, включающую четыре фазы: начало, исследование, построение и внедрение. Каждая фаза может быть разбита на этапы (итерации), в результате которых выпускается версия для внутреннего или внешнего использования. Прохождение через четыре основные фазы называется циклом разработки, каждый цикл завершается генерацией версии системы. Если после этого работа над проектом не прекращается, то полученный продукт продолжает развиваться и снова минует те же фазы. Суть работы в рамках RUP — это создание и сопровождение моделей на базе UML (Unified Modeling Language — унифицированный язык моделирования).
- Microsoft Solution Framework (MSF) сходна с RUP, также включает четыре фазы (анализ, проектирование, разработка, стабилизация), является итерационной, предполагает использование объектно-ориентированного моделирования. MSF в сравнении с RUP в большей степени ориентирована на разработку бизнес-приложений.
- Extreme Programming (XP). Экстремальное программирование (самая новая среди рассматриваемых методологий) сформировалось в 1996 году. В основе методологии командная работа, эффективная коммуникация между заказчиком и исполнителем в течение всего проекта по разработке информационной системы, а разработка ведется с использованием последовательно дорабатываемых прототипов.

В соответствии с базовым международным стандартом ISO/IEC 12207 все процессы жизненного цикла делятся на три группы:

- *основные процессы* — приобретение, поставка, разработка, эксплуатация, сопровождение;
- *вспомогательные процессы* — документирование, управление конфигурацией, обеспечение качества, разрешение проблем, аудит, аттестация, совместная оценка, верификация;
- *организационные процессы* — создание инфраструктуры, управление, обучение, усовершенствование.

Ограничимся в данной главе этим кратким обзором методологий проектирования, перенесем описание процесса проектирования даталогической модели в *разд. 11.4* и приведем здесь лишь некоторые принципы проверки качества и полноты инфологической модели.

Качество сущностей. Основной гарантией качества сущности является ответ на вопрос, действительно ли объект является сущностью, то есть важным объектом или явлением, информация о котором должна храниться в базе данных.

Список проверочных вопросов для сущности:

1. Отражает ли имя сущности суть данного объекта?
2. Нет ли пересечения с другими сущностями?
3. Имеются ли хотя бы два атрибута?
4. Всего атрибутов не более восьми?
5. Есть ли синонимы/омонимы данной сущности?
6. Сущность определена полностью?
7. Есть ли уникальный идентификатор?
8. Имеется ли хотя бы одна связь?
9. Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, архивированию и использованию значения сущности?
10. Ведется ли история изменений?
11. Имеет ли место соответствие принципам нормализации данных?
12. Нет ли такой же сущности в другой прикладной системе, возможно, под другим именем?
13. Не имеет ли сущность слишком общий смысл?
14. Достаточен ли уровень обобщения, воплощенный в ней?

Качество атрибутов. Следует выяснить, а действительно ли это атрибуты, то есть описывают ли они тем или иным образом данную сущность.

Список проверочных вопросов для атрибута:

1. Является ли наименование атрибута существительным единственного числа, отражающим суть обозначаемого атрибутом свойства?
2. Не включает ли в себя наименование атрибута имя сущности (этого быть не должно)?
3. Имеет ли атрибут только одно значение в каждый момент времени?
4. Отсутствуют ли повторяющиеся значения (или группы)?

5. Описаны ли формат, длина, допустимые значения, алгоритм получения и т. п.?
6. Не может ли этот атрибут быть пропущенной сущностью, которая пригодилась бы для другой прикладной системы (уже существующей или предполагаемой)?
7. Не может ли он быть пропущенной связью?
8. Нет ли где-нибудь ссылки на атрибут как на "особенность проекта", которая при переходе на прикладной уровень должна исчезнуть?
9. Есть ли необходимость в истории изменений?
10. Зависит ли его значение только от данной сущности?
11. Если значение атрибута является обязательным, всегда ли оно известно?
12. Зависит ли его значение только от какой-то части уникального идентификатора?
13. Зависит ли его значение от значений некоторых атрибутов, не включенных в уникальный идентификатор?

Качество связи. Нужно выяснить, отражают ли связи действительно важные отношения, наблюдаемые между сущностями.

Список проверочных вопросов для связи:

1. Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи и вписывается ли в принятый синтаксис?
2. Участвуют ли в ней только две стороны?
3. Не является ли связь переносимой?
4. Заданы ли степень связи и обязательность для каждой стороны?
5. Допустима ли конструкция связи?
6. Не относится ли конструкция связи к редко используемым?
7. Не является ли она избыточной?
8. Не изменяется ли она с течением времени?
9. Если связь обязательная, всегда ли она отражает отношение к сущности, представляющей противоположную сторону?

Глава 11

Нормализация

11.1. О нормализации, функциональных и многозначных зависимостях

Нормализация — это разбиение таблицы на две или более, обладающих лучшими свойствами при включении, изменении и удалении данных. Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором *каждый факт появляется лишь в одном месте*, т. е. исключена избыточность информации. Это делается не столько с целью экономии памяти, сколько для исключения возможной противоречивости хранимых данных.

Как указывалось в *разд. 3.1*, каждая таблица в реляционной базе данных удовлетворяет условию, в соответствии с которым в позиции на пересечении каждой строки и столбца таблицы всегда находится единственное атомарное значение, и никогда не может быть множества таких значений. Любая таблица, удовлетворяющая этому условию, называется *нормализованной* (см. таблицы с рис. 10.1 и 10.2). Фактически, ненормализованные таблицы, т. е. таблицы, содержащие повторяющиеся группы (см. табл. 10.1), даже не допускаются в реляционной БД.

Всякая нормализованная таблица автоматически считается таблицей в *первой нормальной форме*, сокращенно *1НФ*. Таким образом, строго говоря, "нормализованная" и "находящаяся в 1НФ" означают одно и то же. Однако на практике термин "нормализованная" часто используется в более узком смысле — "полностью нормализованная", который означает, что в проекте не нарушаются никакие принципы нормализации.

Теперь в дополнение к 1НФ можно определить дальнейшие уровни нормализации — *вторую нормальную форму (2НФ), третью нормальную форму (3НФ)* и т. д. По существу, таблица находится в 2НФ, если она находится

в 1НФ и удовлетворяет, кроме того, некоторому дополнительному условию, суть которого будет рассмотрена далее. Таблица находится в 3НФ, если она находится во 2НФ и, помимо этого, удовлетворяет еще другому дополнительному условию и т. д.

Таким образом, каждая нормальная форма является в некотором смысле более ограниченной, но и более *желательной*, чем предшествующая. Это связано с тем, что "(N+1)-я нормальная форма" не обладает некоторыми непривлекательными особенностями, свойственными "N-й нормальной форме". Общий смысл дополнительного условия, налагаемого на (N+1)-ю нормальную форму по отношению к N-й нормальной форме, состоит в исключении этих непривлекательных особенностей. В *разд. 10.3* мы выявляли непривлекательные особенности табл. 10.2 и для их исключения выполняли "интуитивную нормализацию".

Теория нормализации основывается на наличии той или иной зависимости между полями таблицы. Определены два вида таких зависимостей: функциональные и многозначные.

Функциональная зависимость. Поле Б таблицы функционально зависит от поля А той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений поля А обязательно существует только одно из различных значений поля Б. Отметим, что здесь допускается, что поля А и Б могут быть составными.

Например, в таблице `Блюда` (см. рис. 10.2) поля `Блюдо` и `Вид` функционально зависят от ключа `БЛ`, а в таблице `Поставщики` поле `Страна` функционально зависит от составного ключа (`Поставщик, Город`). Однако последняя зависимость не является функционально полной, так как `Страна` функционально зависит и от части ключа — поля `Город`.

Полная функциональная зависимость. Поле Б находится в полной функциональной зависимости от составного поля А, если оно функционально зависит от А и не зависит функционально от любого подмножества поля А.

Многозначная зависимость. Поле А многозначно определяет поле Б той же таблицы, если для каждого значения поля А существует хорошо определенное множество соответствующих значений Б.

Для примера рассмотрим табл. 11.1. В ней есть многозначная зависимость "`Дисциплина-Преподаватель`": дисциплина (в примере Информатика) может читаться несколькими преподавателями (в примере Шипиловым и Голованевским). Есть и другая многозначная зависимость "`Дисциплина-Учебник`": при изучении Информатики используются учебники "Паскаль для всех" и "Язык Си". При этом `Преподаватель` и `Учебник` не связны функциональной

зависимостью, что приводит к появлению избыточности (для добавления еще одного учебника придется ввести в таблицу две новых строки). Дело улучшается при замене этой таблицы на две: ("`Дисциплина-Преподаватель`" и "`Дисциплина-Учебник`").

Таблица 11.1. *Обучение*

Дисциплина	**Преподаватель**	**Учебник**
Информатика	Шипилов П. А.	Форсайт Р. Паскаль для всех
Информатика	Шипилов П. А.	Уэйт М. и др. Язык Си
Информатика	Голованевский Г. Л.	Форсайт Р. Паскаль для всех
Информатика	Голованевский Г. Л.	Уэйт М. и др. Язык Си
...	...	...

11.2. Нормальные формы

В *разд. 11.1* было дано определение первой нормальной формы (1НФ). Приведем здесь более строгое ее определение, а также определения других нормальных форм.

Таблица находится в *первой нормальной форме (1НФ)* тогда и только тогда, когда ни одна из ее строк не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.

Из таблиц, рассмотренных в *главе 10*, не удовлетворяет этим требованиям (т. е. не находится в 1НФ) только табл. 10.1.

Таблица находится во *второй нормальной форме (2НФ)*, если она удовлетворяет определению 1НФ и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом.

Кроме табл. 10.1 не удовлетворяет этим требованиям только табл. 10.2.

Как обосновано далее (пример 11.1), она имеет составной первичный ключ

`(Блюдо, Продукт, Поставщик, Город, Цена)`

и содержит множество неключевых полей (`Вид`, `Основа`, `Выход` и т. д.), зависящих лишь от той или иной части первичного ключа. Так поля `Вид` и `Основа` зависят только от поля `Блюдо`, `Статус` — от поля `Поставщик` и т. п. Следова-

тельно, эти поля не связаны с первичным ключом полной функциональной зависимостью.

Ко второй нормальной форме приведены почти все таблицы с рис. 10.1 кроме таблицы `Поставщики`, в которой `Страна` зависит только от поля `Город`, который является частью первичного ключа `(Поставщик, Город)`. Последнее обстоятельство приводит к проблемам при:

- включении данных (пока не появится поставщик из Вильнюса, нельзя зафиксировать, что это город Литвы);
- удалении данных (исключение поставщика может привести к потере информации о местонахождении города);
- обновлении данных (при изменении названия страны приходится просматривать множество строк, чтобы исключить получение противоречивого результата).

Разбивая эту таблицу на две таблицы `Поставщики` и `Города`, можно исключить указанные аномалии.

Что же касается таблиц с рис. 10.2, то ввод в них отсутствующих в предметной области цифровых первичных и внешних ключей формально затрудняет процедуру выявления функциональных связей между этими ключами и остальными полями. Действительно, легко установить связь между атрибутом `Блюдо` и `Вид` (блюда): Харчо — Суп, Лобио — Закуска и т. п., но нет прямой зависимости между полями `БЛ` и `Вид` (блюда), если не помнить, что значение `БЛ` соответствует номеру блюда.

Для упрощения нормализации подобных таблиц целесообразно использовать следующую рекомендацию. При проведении нормализации таблиц, в которые введены цифровые (или другие) заменители составных и (или) текстовых первичных и внешних ключей, следует хотя бы мысленно подменять их на исходные ключи, а после окончания нормализации снова восстанавливать.

При использовании этой рекомендации таблицы с рис. 10.2 временно превращаются в таблицы с рис. 10.1, а после выполнения нормализации и восстановления полей `БЛ`, `ПР` и `ПС` — в нормализованные таблицы.

Таблица находится в *третьей нормальной форме (3НФ)*, если она удовлетворяет определению 2НФ и ни одно из ее неключевых полей не зависит функционально от любого другого неключевого поля.

После разделения таблицы `Поставщики` с рис. 10.1 на две части все таблицы этого проекта удовлетворяют определению 2НФ, а так как в них нет неключевых полей, функционально зависящих друг от друга, то все они находятся в 3НФ.

Как ни странно, этого нельзя сказать об аналогичных таблицах с рис. 10.2. Если забыть рекомендацию о подмене на время нормализации ключей БЛ, ПР и ПС на Блюдо, Продукт и (Поставщик, Город), то среди этих таблиц появятся две, не удовлетворяющие определению 3НФ. Действительно, так как после ввода первичных ключей БЛ и ПР поля Блюдо и Продукт стали неключевыми — появились не существовавшие ранее функциональные зависимости между неключевыми полями:

Блюдо → Вид и Поставщик → Страна.

Следовательно, для приведения таблиц Блюда и Продукты с рис. 10.2 к 3НФ их надо разбить на:

Блюда(БЛ, Блюдо)

Вид_блюда(БЛ, Вид)

Продукты(ПР, Продукт)

хотя интуиция подсказывает, что это лишнее разбиение, совсем не улучшающее проект базы данных.

Столкнувшись с подобными несуразностями, которые могут возникать не только из-за введения кодированных первичных ключей, теоретики реляционных систем Кодд и Бойс обосновали и предложили более строгое определение для 3НФ, которое учитывает, что в таблице может быть несколько *возможных* ключей.

Таблица находится в *нормальной форме Бойса—Кодда (НФБК)*, если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от *возможного* ключа.

В соответствие с этой формулировкой таблицы Блюда и Продукты с рис. 10.2, имеющие по паре возможных ключей (БЛ и Блюдо) и (ПР и Продукт) находятся в НФБК или в 3НФ.

В следующих нормальных формах (4НФ и 5НФ) учитываются не только функциональные, но и многозначные зависимости между полями таблицы. Для их описания познакомимся с понятием полной декомпозиции таблицы.

Полной декомпозицией таблицы называют такую совокупность произвольного числа ее проекций, соединение которых полностью совпадает с содержимым таблицы.

Например, естественным соединением (см. *разд. 3.3*) таблиц с рис. 10.1 можно образовать исходную таблицу, приведенную на рис. 10.2. Следовательно, таблицы с рис. 10.1 и 10.2 являются полными декомпозициями табл. 10.2 (универсального отношения СООК).

Теперь можно дать определения высших нормальных форм. И сначала будет дано определение для последней из предложенных — 5НФ.

Таблица находится в *пятой нормальной форме (5НФ*) тогда и только тогда, когда в каждой ее полной декомпозиции все проекции содержат возможный ключ. Таблица, не имеющая ни одной полной декомпозиции, также находится в 5НФ.

Четвертая нормальная форма (4НФ) является частным случаем 5НФ, когда полная декомпозиция должна быть соединением ровно двух проекций. Весьма не просто подобрать реальную таблицу, которая находилась бы в 4НФ, но не была бы в 5НФ.

11.3. Процедура нормализации

Как уже говорилось, нормализация — это разбиение таблицы на несколько, обладающих лучшими свойствами при обновлении, включении и удалении данных. Теперь можно дать и другое определение: *нормализация* — это процесс последовательной замены таблицы ее полными декомпозициями до тех пор, пока все они не будут находиться в 5НФ. На практике же достаточно привести таблицы к НФБК и с большой гарантией считать, что они находятся в 5НФ. Разумеется, этот факт нуждается в проверке, однако пока не существует эффективного алгоритма такой проверки. Поэтому остановимся лишь на процедуре приведения таблиц к НФБК.

Эта процедура основывается на том, что единственными функциональными зависимостями в любой таблице должны быть зависимости вида K → F, где K — первичный ключ, а F — некоторое другое поле. Заметим, что это следует из определения первичного ключа таблицы, в соответствии с которым K → F всегда имеет место для всех полей данной таблицы. "Один факт в одном месте" говорит о том, что не имеют силы никакие другие функциональные зависимости. Цель нормализации состоит именно в том, чтобы избавиться от всех этих "других" функциональных зависимостей, т. е. таких, которые имеют иной вид, чем K → F.

Если воспользоваться рекомендацией из *разд. 11.2* и подменить на время нормализации коды первичных (внешних) ключей на исходные ключи, то, по существу, следует рассмотреть лишь два случая:

1. Таблица имеет составной первичный ключ вида, скажем, (К1, К2), и включает также поле F, которое функционально зависит от части этого ключа, например, от К2, но не от полного ключа. В этом случае рекомендуется сформировать другую таблицу, содержащую К2 и F (первичный ключ — К2), и удалить F из первоначальной таблицы:

```
Заменить T(K1, K2, F),  первичный ключ (K1, K2), ФЗ K2 → F
на       T1(K1, K2),    первичный ключ (K1, K2),
и        T2(K2, F),     первичный ключ K2.
```

2. Таблица имеет первичный (возможный) ключ К, не являющееся возможным ключом поле F1, которое, конечно, функционально зависит от К, и другое неключевое поле F2, которое функционально зависит от F1. Решение здесь, по существу, то же самое, что и прежде — формируется другая таблица, содержащая F1 и F2, с первичным ключом F1, и F2 удаляется из первоначальной таблицы:

```
Заменить T(K, F1, F2),  первичный ключ K, ФЗ F1 → F2
на      T1(K, F1),      первичный ключ K,
и       T2(F1, F2),     первичный ключ F1.
```

Для любой заданной таблицы, повторяя применение двух рассмотренных правил, почти во всех практических ситуациях можно получить, в конечном счете, множество таблиц, которые находятся в "окончательной" нормальной форме и, таким образом, не содержат каких-либо функциональных зависимостей вида, отличного от K → F.

Для выполнения этих операций необходимо первоначально иметь в качестве входных данных какие-либо "большие" таблицы (например, универсальные отношения). Но нормализация ничего не говорит о том, как получить эти большие таблицы. Далее будет рассмотрена процедура получения таких исходных таблиц, а здесь приведем примеры нормализации.

Пример 11.1. Применим рассмотренные правила для полной нормализации универсального отношения COOK (табл. 10.2).

1. Определение первичного ключа таблицы.

   Предположим, что каждое блюдо имеет уникальное название, относится к единственному виду и приготавливается по единственному рецепту, т. е. название блюда однозначно определяет его вид и рецепт. Предположим также, что название организации поставщика уникально для того города, в котором он расположен, и названия городов уникальны для каждой из стран, т. е. название поставщика и город однозначно определяют этого поставщика, а город — страну его нахождения. Наконец, предположим, что поставщик может осуществлять в один и тот же день только одну поставку каждого продукта, т. е. название продукта, название организации поставщика, город и дата поставки однозначно определяют вес и цену поставленного продукта. Тогда в качестве первичного ключа отношения COOK можно использовать следующий набор атрибутов:

   Блюдо, Продукт, Поставщик, Город, Цена.

2. Выявление полей, функционально зависящих от части составного ключа.

   Поля Вид и Основа функционально зависят только от поля Блюдо, т. е.

   Блюдо K → Вид

   Блюдо K → Основа

Аналогичным образом можно получить зависимости:

(Блюдо, Продукт) К → Вес

Город К → Страна

(Поставщик, Город) К → Цена

3. Формирование новых таблиц.

   Полученные функциональные зависимости определяют состав таблиц, которые можно сформировать из данных универсального отношения:

   Блюда (Блюдо, Вид)

   Состав (Блюдо, Продукт, Вес (г))

   Города (Город, Страна)

   Поставки (Поставщик, Город, К_во, Цена)

4. Корректировка исходной таблицы.

   После выделения из состава универсального отношения указанных ранее таблиц, там остались лишь сведения о поставщиках, для хранения которых целесообразно создать таблицу

   Поставщики (Поставщик, Город)

   т. е. использовать часть исходного первичного ключа, так как остальные его части уже ничего не определяют.

Таким образом, процедура последовательной нормализации позволила получить проект, лучший, чем приведен на рис. 10.1.

Пример 11.2. Для улучшения проекта, приведенного на рис. 10.2, нужно определить первичные ключи таблиц и выявить, нет ли в таблицах полей, зависящих лишь от части этих ключей. Такое поле есть только в одной таблице. Это поле Страна в таблице Поставщики. Выделяя его вместе с ключом Город в таблицу Страны, получим нормализованный проект.

11.4. Построение даталогической (табличной) модели

Рассмотренный в *разд. 10.4* процесс построения концептуальной модели базы данных должен быть продолжен построением ее табличной модели. Для этого необходимо:

1. Представить каждый стержень (независимую сущность) таблицей базы данных (базовой таблицей) и специфицировать первичный ключ этой базовой таблицы.

2. Представить каждую ассоциацию (связь вида "многие-ко-многим" или "многие-ко-многим-ко-многим" и т. д. между сущностями) как базовую таблицу. Использовать в этой таблице внешние ключи для идентификации участников ассоциации и специфицировать ограничения, связанные с каждым из этих внешних ключей.
3. Представить каждую характеристику как базовую таблицу с внешним ключом, идентифицирующим сущность, описываемую этой характеристикой. Специфицировать ограничения на внешний ключ этой таблицы и ее первичный ключ — по всей вероятности, комбинации этого внешнего ключа и свойства, которое гарантирует "уникальность в рамках описываемой сущности".
4. Представить каждое свойство (атрибут) как поле в базовой таблице, представляющей сущность, которая непосредственно описывается этим свойством.
5. Для того чтобы исключить в проекте непреднамеренные нарушения каких-либо принципов нормализации, необходимо выполнить процедуру нормализации (см. *разд. 11.3*).
6. Если в процессе нормализации было произведено разделение каких-либо таблиц, то следует модифицировать инфологическую модель базы данных и повторить перечисленные шаги.
7. Указать ограничения целостности проектируемой базы данных и дать (если это необходимо) краткое описание полученных таблиц и их полей.

Для примера приведем описания таблиц `Блюда` и `Состав` (листинги 11.1 и 11.2).

Листинг 11.1. Блюда ("русифицированный" листинг)

```
СОЗДАТЬ ТАБЛИЦУ  Блюда
( БЛ Число(2),
  Блюдо Текст(60),
  Вид Текст(7),
  Основа Текст(6),
  Выход Число(4,1)
);
КОММЕНТАРИЙ К ТАБЛИЦЕ Блюда - 'Перечень блюд, известных шеф-повару';
КОММЕНТАРИЙ К СТОЛБЦУ Блюда.БЛ     - 'Код блюда';
КОММЕНТАРИЙ К СТОЛБЦУ Блюда.Блюдо  - 'Название блюда';
КОММЕНТАРИЙ К СТОЛБЦУ Блюда.Вид    - 'Вид блюда';
КОММЕНТАРИЙ К СТОЛБЦУ Блюда.Основа - 'Основной продукт в блюде';
КОММЕНТАРИЙ К СТОЛБЦУ Блюда.Выход  - 'Масса порции готового блюда';
```

```
ИЗМЕНИТЬ ТАБЛИЦУ Блюда - ДОБАВИТЬ ОГРАНИЧЕНИЕ
  "Повтор кода блюда !" ПЕРВИЧНЫЙ КЛЮЧ (БЛ);
ИЗМЕНИТЬ ТАБЛИЦУ Блюда - ДОБАВИТЬ ОГРАНИЧЕНИЕ
  "Такое блюдо уже есть !" УНИКАЛЬНЫЙ КЛЮЧ (БЛ, Блюдо);
ИЗМЕНИТЬ ТАБЛИЦУ Блюда - ДОБАВИТЬ ОГРАНИЧЕНИЕ
  "Только Закуска,Суп,Горячее,Десерт,Напиток" ПРОВЕРЯТЬ
  (Вид ИЗ ('Закуска', 'Суп', 'Горячее', 'Десерт', 'Напиток'));
ИЗМЕНИТЬ ТАБЛИЦУ Блюда - ДОБАВИТЬ ОГРАНИЧЕНИЕ
   "Только Кофе,Крупа,Молоко,Мясо,Овощи,Рыба,Фрукты,Яйца !" ПРОВЕРЯТЬ
  (Основа ИЗ ('Кофе', 'Крупа', 'Молоко', 'Мясо', 'Овощи', 'Рыба',
  'Фрукты', 'Яйца'));
```

Листинг 11.2. Состав ("русифицированный" листинг)

```
СОЗДАТЬ ТАБЛИЦУ  Состав
(
  БЛ  Число(2),
  ПР  Число(2),
  Вес Число(3)
);
КОММЕНТАРИЙ К ТАБЛИЦЕ Состав - 'Состав блюд';
КОММЕНТАРИЙ К СТОЛБЦУ БЛ     - 'Код блюда';
КОММЕНТАРИЙ К СТОЛБЦУ ПР     - 'Код продукта';
КОММЕНТАРИЙ К СТОЛБЦУ Вес    - 'Масса продукта в блюде';
ИЗМЕНИТЬ ТАБЛИЦУ Состав - ДОБАВИТЬ ОГРАНИЧЕНИЕ
  "Состав не уникален !" ПЕРВИЧНЫЙ КЛЮЧ (БЛ, ПР);
ИЗМЕНИТЬ ТАБЛИЦУ Состав - ДОБАВИТЬ ОГРАНИЧЕНИЕ
  "Такого блюда нет !" ВНЕШНИЙ КЛЮЧ (БЛ)
  СВЯЗЬ Блюда (БЛ);
ИЗМЕНИТЬ ТАБЛИЦУ Состав - ДОБАВИТЬ ОГРАНИЧЕНИЕ
  "Такого продукта нет !" ВНЕШНИЙ КЛЮЧ (ПР)
  СВЯЗЬ Продукты (ПР);
ИЗМЕНИТЬ ТАБЛИЦУ Состав - ДОБАВИТЬ ОГРАНИЧЕНИЕ
   "Вес должен быть между 1 и 500 !" ПРОВЕРЯТЬ (Вес МЕЖДУ 1 И 500);
```

Естественно, что приведенные нестандартные листинги 11.1 и 11.2 сможет "понять" только гипотетическая СУБД. Понятным любой существующей СУБД может быть листинг, написанный с помощью предложений определения

данных (Data Definition Language, DDL) языка SQL (см. *часть V*). Поэтому перепишем приведенные ранее листинги на язык SQL — листинги 11.3 и 11.4.

Листинг 11.3. Блюда (листинг на языке SQL)

```
CREATE TABLE Блюда
(
  БЛ      NUMBER(2),
  Блюдо   VARCHAR2(60),
  Вид     VARCHAR2(7),
  Основа  VARCHAR2(6),
  Выход   NUMBER(4,1),
);
COMMENT ON TABLE Блюда IS 'Перечень блюд, известных шеф-повару';
COMMENT ON COLUMN Блюда.БЛ IS 'Код блюда';
COMMENT ON COLUMN Блюда.Блюдо IS 'Название блюда';
COMMENT ON COLUMN Блюда.Вид IS 'Вид блюда';
COMMENT ON COLUMN Блюда.Основа IS 'Основной продукт в блюде';
COMMENT ON COLUMN Блюда.Выход IS 'Масса порции готового блюда';
ALTER TABLE Блюда
  ADD CONSTRAINT "Повтор кода блюда !" PRIMARY KEY (БЛ);
ALTER TABLE Блюда
  ADD CONSTRAINT "Такое блюдо уже есть !" UNIQUE (БЛ, Блюдо);
ALTER TABLE Блюда
  ADD CONSTRAINT "Только Закуска,Суп,Горячее,Десерт,Напиток"
CHECK (Вид IN ('Закуска', 'Суп', 'Горячее', 'Десерт', 'Напиток'));
ALTER TABLE Блюда
  ADD CONSTRAINT "Только Кофе,Крупа,Молоко,Мясо,Овощи,Рыба,Фрукты,Яйца !"
CHECK (Основа IN ('Кофе', 'Крупа', 'Молоко', 'Мясо', 'Овощи', 'Рыба',
  'Фрукты', 'Яйца'));
```

Листинг 11.4. Состав (листинг на языке SQL)

```
CREATE TABLE СОСТАВ
(
  БЛ   NUMBER(2),
  ПР   NUMBER(2),
  Вес  NUMBER(3)
);
```

```
COMMENT ON TABLE Состав IS 'Состав блюд';
COMMENT ON COLUMN Состав.БЛ IS 'Код блюда';
COMMENT ON COLUMN Состав.ПР IS 'Код продукта';
COMMENT ON COLUMN Состав.Вес IS 'Масса продукта в блюде';
ALTER TABLE Состав
  ADD CONSTRAINT "Состав не уникален !" PRIMARY KEY (БЛ, ПР);
ALTER TABLE Состав
  ADD CONSTRAINT "Такого блюда нет !" FOREIGN KEY (БЛ)
  REFERENCES Блюда (БЛ);
ALTER TABLE Состав
  ADD CONSTRAINT "Такого продукта нет !" FOREIGN KEY (ПР)
  REFERENCES Продукты (ПР);
ALTER TABLE Состав
  ADD CONSTRAINT "Вес должен быть между 1 и 500 !"
CHECK (Вес BETWEEN 1 AND 500);
```

11.5. Различные советы и рекомендации

Векторы. Представляйте векторы по столбцам, а не по строкам. Например, диаграмму продаж товаров x, y, ... за последние годы лучше представить в виде:

```
ТОВАР    МЕСЯЦ  КОЛ-ВО
-----   ------- ------
  x     ЯНВАРЬ   100
  x     ФЕВРАЛЬ   50
 ...      ...    ...
  x     ДЕКАБРЬ  360
  y     ЯНВАРЬ    75
  y     ФЕВРАЛЬ  144
 ...      ...    ...
  y     ДЕКАБРЬ   35
 ...      ...    ...
```

а не так, как показано далее:

```
ТОВАР   КОЛ-ВО   КОЛ-ВО          КОЛ-ВО
        ЯНВАРЬ   ФЕВРАЛЬ  ...   ДЕКАБРЬ
-----   -------  -------        -------
  x       100       50    ...     360
  y        75      144    ...      35
 ...      ...      ...    ...     ...
```

Одна из причин такой рекомендации заключается в том, что при этом значительно проще записываются обобщенные (параметризованные) запросы. Рассмотрите, например, как выглядит сравнение сведений из диаграммы продаж товара i в месяце с номером m со сведениями для товара j в месяце с номером n, где i, j, m и n — параметры.

Неопределенные значения. Будьте очень внимательны с неопределенными (`NULL`) значениями. В поведении неопределенных значений проявляется много произвола и противоречивости. В разных СУБД при выполнении различных операций (сравнение, объединение, сортировка, группирование и др.) два неопределенных значения могут быть или не быть равными друг другу. Они могут по-разному влиять на результат выполнения операций по определению средних значений и нахождения количества значений. Для исключения ошибок в ряде СУБД существует возможность замены `NULL`-значения нулем при выполнении расчетов, объявление всех `NULL`-значений равными друг другу и т. п.

Глава 12

Пример проектирования базы данных "LIBRARY"

12.1. Назначение и предметная область

База данных предназначена для хранения данных о приобретенных библиотекой изданиях (монографиях, справочниках, сборниках статей и т. п.), информации о местонахождении отдельных экземпляров каждого издания и сведений о читателях.

Для ведения библиотечных каталогов, организации поиска требуемых изданий и библиотечной статистики в базе должны храниться сведения, большая часть которых размещаются в аннотированных каталожных карточках (рис. 12.1).

Д27	***Дейт К. Дж.*** *Введение в системы баз данных, 8-е издание.: Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил. — Парал. тит. англ.*
	ISBN 5-8459-0788-8 (рус.)
	Новое издание фундаментального труда Криса Дейта представляет собой исчерпывающее введение в очень обширную в настоящее время теорию систем баз данных. С помощью этой книги читатель сможет приобрести фундаментальные знания в области технологии баз данных, а также ознакомиться с направлениями, по которым рассматриваемая сфера деятельности, вероятно, будет развиваться в будущем. Книга предназначена для использования в основном в качестве учебника, а не справочника, и поэтому, несомненно, вызовет интерес у программистов-профессионалов, научных работников и студентов, изучающих соответствующие курсы в высших учебных заведениях. В ней сделан акцент на усвоении сути и глубоком понимании излагаемого материала, а не просто на его формальном изложении.
	Книга, безусловно, будет полезна всем, кому приходится работать с базами данных или просто пользоваться ими.
	ББК 32.973.26-018.2.75

Рис. 12.1. Макет аннотированной каталожной карточки

Анализ запросов на литературу показал, что для поиска подходящих изданий (по тематике, автору, художнику, издательству и т. п.) и отбора нужного (например, по аннотации) необходимо выделить следующие атрибуты каталожной карточки, а также выпускных данных книги (рис. 12.2):

1. Автор (фамилия и имена (инициалы) или псевдоним каждого автора издания).
2. Название (заглавие) издания.

Научно-популярное издание

К.Дж. Дейт

Введение в системы баз данных
8-е издание

Литературный редактор *С.Г. Татаренко*
Верстка *М.А. Смолина*
Художественный редактор *С.А. Чернокозинский*
Корректоры *З.В. Александрова, Л.А. Гордиенко, О.В. Мишутина, Л.В. Чернокозинская*

Издательский дом "Вильямс".
101509, Москва, ул. Лесная, д. 43, стр. 1.
Подписано в печать 28.02.2005. Формат 70x100/16.
Гарнитура Times. Печать офсетная.
Усл. печ. л. 118.68 Уч.-изд. л. 86,19.
Тираж 3000 экз. Заказ № 859.

Отпечатано с диапозитивов в ФУГП "Печатный двор"
Министерства РФ по делам печати,
Телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр. 15.

Рис. 12.2. Выпускные данные книги

3. Номер тома (части, книги, выпуска).
4. Вид издания (сборник, справочник, монография, сборник научных трудов, автореферат диссертации, энциклопедический словарь, учебное пособие и пр.).
5. Тип издания (научное и научно-популярное издание, справочное и рекламное издание, учебное издание, литературно-художественное издание и пр.).
6. Составители сборника произведений разных авторов (фамилия и имена (инициалы) каждого из составителей издания).
7. Язык, с которого выполнен перевод издания.
8. Переводчик (фамилия и инициалы каждого переводчика).
9. Данные о лицах, принимавших участие в создании книги (редакторы, художники, корректоры и т. п.).
10. Данные о повторности издания и его характеристики (стереотипное, исправленное, дополненное и т. п.): издание третье, дополненное и переработанное.
11. Данные об утверждении издания в качестве учебника, учебного пособия или официального издания.
12. Издательство (название и адрес издательства).
13. Год выпуска издания.
14. Макет аннотированной каталожной карточки.
15. Международный стандартный номер книги (International Standard Book Number, ISBN).
16. Библиотечный шифр (например, ББК 32.973).
17. Авторский знак (например, Д27).

ISBN, библиотечный шифр и авторский знак используются при составлении каталогов и организации расстановки изданий на полках: по содержанию (в соответствии с библиотечным шифром) и алфавиту (в соответствии с авторским знаком).

Международный стандартный номер книги состоит из аббревиатуры ISBN (независимо от языка издания) и десяти арабских цифр. Последняя цифра кода ISBN — контрольная и может быть римской цифрой X, используемой для обозначения числа 10. Цифровая часть кода ISBN состоит из четырех групп цифр, содержащих различное количество цифровых знаков, и отделяемых друг от друга дефисом. Группы цифр обозначают следующие данные: идентификатор страны, идентификатор издательства, порядковый номер книги, контрольная цифра.

Библиотечно-библиографическая классификация (ББК) распределяет издания по отраслям знания в соответствии с их содержанием. В ней используются цифро-буквенные индексы ступенчатой структуры.

Каждый из девяти классов (1. Марксизм-ленинизм; 2. Естественные науки; 3. Техника. Технические науки; 4. Сельское и лесное хозяйство; 5. Здравоохранение; 6/8. Общественные и гуманитарные науки; 9. Библиографические пособия. Справочные издания. Журналы.) расчленяется на подклассы и следующие ступени деления:

3. Техника. Технические науки.
32 Радиоэлектроника.
32.97 Вычислительная техника.
32.973 Электронные вычислительные машины и устройства.
32.973.2 Электронно-вычислительные машины и устройства дискретного действия.

Шифр ББК используется при выделении хранимым изданиям определенных комнат, стеллажей и полок, а также для составления каталогов и статистических отчетов.

Авторский знак, состоящий из первой буквы фамилии (псевдонима) автора или названия издания (для изданий без автора) и числа, соответствующего слогу, наиболее приближающегося по написанию к первым буквам фамилии (названия), упрощает расстановку книг на полках в алфавитном порядке.

К объектам и атрибутам, позволяющим охарактеризовать отдельные экземпляры изданий, места их хранения и читателей, можно отнести:

1. Номер комнаты (помещения для хранения экземпляров изданий).
2. Номер стеллажа в комнате.
3. Номер полки на стеллаже.
4. Номер (инвентарный номер) экземпляра.
5. Дата приобретения конкретного экземпляра.
6. Дата списания конкретного экземпляра.
7. Цена конкретного экземпляра.
8. Фамилия читателя.
9. Имя читателя.
10. Отчество читателя.
11. Адрес читателя.

12. Телефон читателя.
13. Дата выдачи читателю конкретного экземпляра.
14. Срок, на который конкретный экземпляр выдан читателю.
15. Дата возврата экземпляра.

12.2. Построение инфологической модели

Анализ определенных ранее объектов и атрибутов позволяет выделить сущности проектируемой базы данных и, приняв решение о создании реляционной базы данных, построить ее инфологическую модель (рис. 12.3).

К *стержневым сущностям* можно отнести:

- `Люди (Ид, Фамилия, Имя, Отчество, Псевдоним, Дата рождения, Пол, Адрес, Телефон)`

  Эта сущность отводится для хранения сведений о читателях, а также всех людях, принимавших участие в подготовке и выпуске издания (авторах, составителях, редакторах, корректорах, переводчиках, художниках и пр.). Такое объединение допустимо, так как данные о разных людях выбираются из одного домена (фамилия, имя и пр.), оно исключает дублирование данных (один и тот же человек может быть читателем библиотеки и играть разные роли в подготовке разных изданий). Например, С. Я. Маршак писал стихи ("Сказка о глупом мышонке") и пьесы ("Двенадцать месяцев"), переводил Дж. Байрона, Р. Бернса, Г. Гейне и составлял сборники стихов.

  Так как фамилия и имена (инициалы) читателя библиотеки и (или) создателя могут быть достаточно громоздкими (М. Е. Салтыков-Щедрин, Франсуа Рене де Шатобриан, Остен Жюль Жан-Батист Ипполит и т. п.) и будут многократно встречаться в разных изданиях, то их целесообразно нумеровать и ссылаться на эти номера. Для этого вводится суррогатный ключ (см. *разд. 2.4*) — целочисленный атрибут `Ид`, который будет автоматически наращиваться на единицу при вводе в базу данных нового автора, переводчика или другого создателя.

  Аналогично создаются суррогатные ключи для большинства сущностей модели рис. 12.3.

- `Создатели (ИД, Наименование)`

  Эта сущность отводится для хранения перечня ролей людей, принимающих участие в создании и выпуске книги: автор, переводчик, редактор, художник, корректор и т. п.

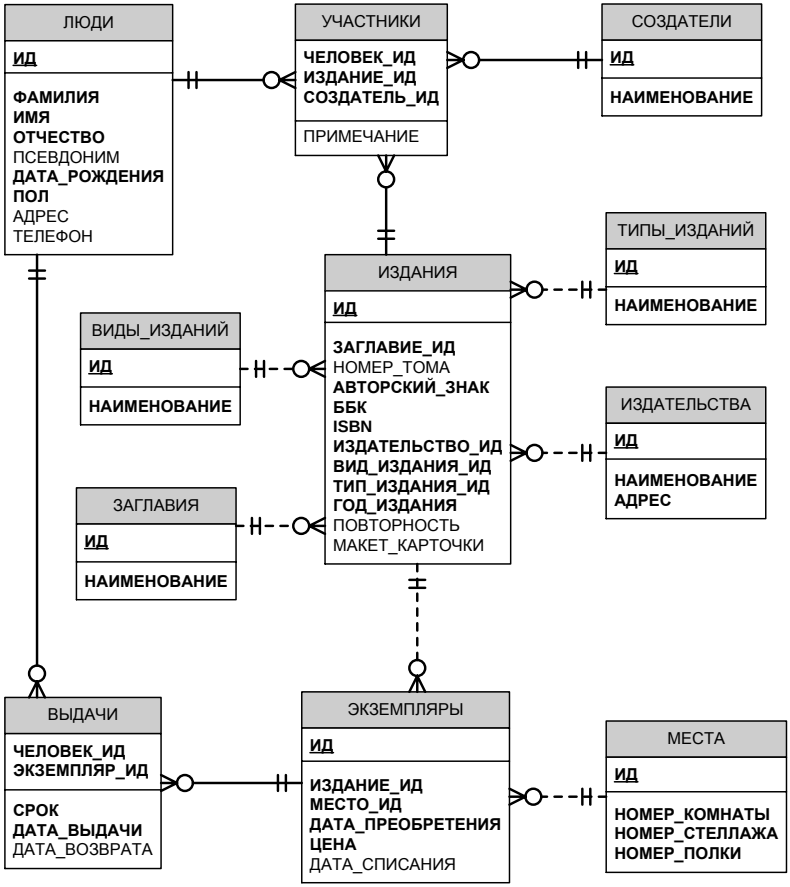


Рис. 12.3. Инфологическая модель базы данных "LIBRARY"

- `Издательства (ИД, Наименование, Адрес)`

  Эта сущность фактически является кратким справочником издательств.

- `Заглавия (ИД, Наименование)`

  Выделение этой сущности позволит сократить объем данных и снизить вероятность возникновения противоречивости (исключается необходимость ввода длинных текстовых названий для различных томов собраний сочинений, повторных изданий, учебников и т. п.).

- `Виды_изданий (ИД, Наименование)`

  Эта сущность отводится для хранения перечня видов изданий (сборник, справочник, монография, сборник научных трудов, автореферат диссертации, энциклопедический словарь, учебное пособие и пр.).

- `Типы_изданий (ИД, Наименование)`

  Эта сущность отводится для хранения перечня типов изданий (научное и научно-популярное издание, справочное и рекламное издание, учебное издание, литературно-художественное издание и пр.).

- `Места (ИД, Номер_комнаты, Номер_стеллажа, Номер_ полки)`

  Одно из значений идентификатора (`Ид`) этой сущности (например, `-1`) отведено для описания обобщенного места, находящегося за стенами хранилища книг (издание выдано читателю, временно передано другой библиотеке или организации).

К *ассоциативным сущностям* можно отнести:

- `Издания (ИД, Заглавие_ид, Номер_тома, Авторский_знак, ББК, ISBN, Издательство_ид, Вид_издания_ид, Тип_издания_ид, Год_издания, Повторность, Макет_карточки)`

  Сущность содержит основные сведения о конкретном издании со ссылками на данные справочников `Заглавия`, `Издательства`, `Виды_изданий`, `Типы_изданий` и `Участники`.

- `Участники (Человек_ид, Создатель_ид, Издание_ид, Примечание)`

  Содержит ссылки на всех лиц, принимавших участие в подготовке и (или) выпуске конкретного издания, с указанием всех их ролей в этой деятельности.

- `Экземпляры (ИД, Издание_ид, Цена, Дата_преобретения, Дата_списания, Место_ид)`

  Содержит сведения о дате приобретения конкретного экземпляра издания и его цене на момент приобретения, месте размещения экземпляра в библиотеке и дате списания экземпляра, если таковое произошло.

- `Выдачи (Человек_ид, Экземпляр_ид, Срок, Дата_выдачи, Дата_возврата)`

  Если не пытаться сокращать названия сущностей и создаваемых по ним таблиц, то `Выдачи` надо было бы назвать "Библиотечный абонемент", предоставляющий читателю (с номером `ИД`) право получить экземпляр издания для работы с ним в читальном зале или в любом месте на определенный срок.

12.3. Построение даталогической модели

Теперь следует проверить, не нарушены ли в данном проекте какие-либо принципы нормализации (*разд. 11.2*), т. е. что любое неключевое поле каждой таблицы:

- функционально зависит от полного первичного ключа, а не от его части (если ключ составной);
- не имеет функциональной зависимости от другого неключевого поля.

Сущности Люди, Создатели, Типы_изданий, Издательства, Виды_изданий, Заглавия и Места, состоящие из суррогатного ключа ИД и не связанных между собой неключевых полей, безусловно нормализованы.

Анализ ассоциативных сущностей Издания и Экземпляры, состоящих из суррогатного ключа ИД и не связанных между собой неключевых полей, показал, что в них нет функциональных связей между неключевыми полями.

И наконец, анализ ассоциативных сущностей Участники и Выдачи, состоящих из составного ключа и неключевых полей, показал, что в них также нет функциональных связей между неключевыми полями. Последние же не зависят функционально от какой-либо части составного ключа.

Теперь перейдем к построению даталогической (табличной) модели.

Для сокращения текста книги, приведем здесь описания ассоциативных сущностей и только трех стержневых: Люди, Создатели и Места. Описания же остальных стержневых сущностей (Типы_изданий, Издательства, Виды_изданий и Заглавия) практически не отличаются от, например, описания таблицы Создатели.

В описаниях (листинги 12.1—12.7) будут использоваться предложения определения данных (Data Definition Language, DDL) языка SQL, подробно рассматриваемые в *частях V* и *VI* этой книги.

Листинг 12.1. Люди

```
-- Предложение для создания таблицы
--
CREATE TABLE ЛЮДИ
(
  ИД NUMBER(6,0) NOT NULL CONSTRAINT "ЛЮДИ_PK" PRIMARY KEY,
  ФАМИЛИЯ       NVARCHAR2(50) NOT NULL,
  ИМЯ           NVARCHAR2(50) NOT NULL,
  ОТЧЕСТВО      NVARCHAR2(50) NOT NULL,
  ПСЕВДОНИМ     NVARCHAR2(50),
  ДАТА_РОЖДЕНИЯ DATE NOT NULL,
  ПОЛ NCHAR(1) NOT NULL CONSTRAINT "Пол может быть М или Ж"
                        CHECK (ПОЛ IN ('М','Ж')),
  АДРЕС         NVARCHAR2(200),
  ТЕЛЕФОН       NVARCHAR2(50),
 CONSTRAINT "ЛЮДИ_UK" UNIQUE (ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО, ДАТА_РОЖДЕНИЯ, ПОЛ)
);
```

```
-- Комментарии для таблицы и ее столбцов
--
COMMENT ON TABLE ЛЮДИ  IS 'Список читателей и/или создателей изданий';
COMMENT ON COLUMN ЛЮДИ.ИД IS 'Уникальный номер человека';
COMMENT ON COLUMN ЛЮДИ.ФАМИЛИЯ IS 'Фамилия человека';
COMMENT ON COLUMN ЛЮДИ.ИМЯ IS 'Имя человека';
COMMENT ON COLUMN ЛЮДИ.ОТЧЕСТВО IS 'Отчество человека';
COMMENT ON COLUMN ЛЮДИ.ПСЕВДОНИМ IS 'Псевдоним человека';
COMMENT ON COLUMN ЛЮДИ.ДАТА_РОЖДЕНИЯ IS 'Дата рождения человека';
COMMENT ON COLUMN ЛЮДИ.ПОЛ IS 'Пол человека';
COMMENT ON COLUMN ЛЮДИ.АДРЕС IS 'Адрес человека';
COMMENT ON COLUMN ЛЮДИ.ТЕЛЕФОН IS 'Телефон человека';

-- Создание генератора уникальной последовательности значений, которые
-- будут использоваться в триггере ЛЮДИ_BIR для выработки значений
-- первичного ключа таблицы Люди
--
CREATE SEQUENCE ЛЮДИ_ПОСЛ INCREMENT BY 1 START WITH 1;

-- Создание триггера для выработки значений первичного ключа таблицы,
-- преобразования с помощью функции InitCap первых букв фамилии, имени и
-- отчества в прописные, а также ввода в столбец ДАТА_РОЖДЕНИЯ даты
-- 11.11.1111, если при вводе в него не вводилось никакого значения
--
CREATE OR REPLACE TRIGGER ЛЮДИ_BIR
 BEFORE INSERT ON люди
 FOR EACH ROW
BEGIN
  SELECT ЛЮДИ_ПОСЛ.NEXTVAL INTO :new.ид FROM dual;
  :new.фамилия := InitCap(:new.фамилия);
  :new.имя := InitCap(:new.имя);
  :new.отчество := InitCap(:new.отчество);
  IF :new.дата_рождения IS NULL THEN
     :new.дата_рождения := to_date('11.11.1111','DD.MM.YYYY');
  END IF;
END люди_bir;
/
```

Листинг 12.2. Создатели

```
-- Предложение для создания таблицы
--
CREATE TABLE СОЗДАТЕЛИ
(
  ИД NUMBER(6,0) NOT NULL CONSTRAINT "СОЗДАТЕЛИ_PK" PRIMARY KEY,
  НАИМЕНОВАНИЕ NVARCHAR2(200) NOT NULL,
  CONSTRAINT "СОЗДАТЕЛИ_UK" UNIQUE (НАИМЕНОВАНИЕ)
);

-- Комментарии для таблицы и ее столбцов
--
COMMENT ON TABLE СОЗДАТЕЛИ IS 'Перечень создателей издания';
COMMENT ON COLUMN СОЗДАТЕЛИ.ИД IS 'Искусственный первичный уникальный
идентификатор';
COMMENT ON COLUMN СОЗДАТЕЛИ.НАИМЕНОВАНИЕ IS 'Наименование создателя';

-- Создание генератора уникальной последовательности значений, которые
-- будут использоваться в триггере СОЗДАТЕЛИ_BIR для выработки значений
-- первичного ключа таблицы Создатели
--
CREATE SEQUENCE СОЗДАТЕЛИ_ПОСЛ INCREMENT BY 1 START WITH 1;

-- Создание триггера для выработки значений первичного ключа таблицы
--
CREATE OR REPLACE TRIGGER СОЗДАТЕЛИ_BIR
  BEFORE INSERT ON создатели
 FOR EACH ROW
BEGIN
  SELECT создатели_посл.NEXTVAL INTO :new.ид FROM dual;
end создатели_bir;
/
```

Листинг 12.3. Места

```
-- Предложение для создания таблицы
--
CREATE TABLE МЕСТА
```

```
(
  ИД NUMBER(6,0) NOT NULL CONSTRAINT "МЕСТА_PK" PRIMARY KEY,
  НОМЕР_КОМНАТЫ  NVARCHAR2(20) NOT NULL,
  НОМЕР_СТЕЛЛАЖА NVARCHAR2(20) NOT NULL,
  НОМЕР_ПОЛКИ    NVARCHAR2(20) NOT NULL,
CONSTRAINT "МЕСТА_UK" UNIQUE (НОМЕР_КОМНАТЫ, НОМЕР_СТЕЛЛАЖА, НОМЕР_ПОЛКИ)
);

-- Комментарии для таблицы и ее столбцов
--
COMMENT ON TABLE МЕСТА IS 'Перечень мест хранения экземпляров';
COMMENT ON COLUMN МЕСТА.ИД IS 'Искусственный первичный уникальный
идентификатор';
COMMENT ON COLUMN МЕСТА.НОМЕР_КОМНАТЫ IS 'Номер комнаты';
COMMENT ON COLUMN МЕСТА.НОМЕР_СТЕЛЛАЖА IS 'Номер стеллажа';
COMMENT ON COLUMN МЕСТА.НОМЕР_ПОЛКИ IS 'Номер полки';

-- Создание генератора уникальной последовательности значений, которые
-- будут использоваться в триггере МЕСТА_BIR для выработки значений
-- первичного ключа таблицы Места
--
CREATE SEQUENCE МЕСТА_ПОСЛ INCREMENT BY 1 START WITH 1;

-- Создание триггера для выработки значений первичного ключа таблицы
--
CREATE OR REPLACE TRIGGER МЕСТА_BIR
  BEFORE INSERT ON места
 FOR EACH ROW
BEGIN
  SELECT места_посл.NEXTVAL INTO :new.ид FROM dual;
end места_bir;
/
```

Листинг 12.4. Издания

```
-- Предложение для создания таблицы
--
CREATE TABLE ИЗДАНИЯ
(
```

```
ИД NUMBER(6,0) NOT NULL CONSTRAINT "ИЗДАНИЯ_PK" PRIMARY KEY,
  ЗАГЛАВИЕ_ИД     NUMBER(6,0) NOT NULL CONSTRAINT "ЗАГЛАВИЯ_FK"
                                       REFERENCES ЗАГЛАВИЯ (ИД),
  НОМЕР_ТОМА      NVARCHAR2(10),
  АВТОРСКИЙ_ЗНАК  NVARCHAR2(6) NOT NULL,
  ББК             NVARCHAR2(20) NOT NULL,
  ISBN            NVARCHAR2(20) NOT NULL,
  ИЗДАТЕЛЬСТВО_ИД NUMBER(6,0) NOT NULL CONSTRAINT "ИЗДАТЕЛЬСТВА_FK"
                                       REFERENCES ИЗДАТЕЛЬСТВА (ИД),
  ВИД_ИЗДАНИЯ_ИД  NUMBER(6,0) NOT NULL CONSTRAINT "ВИДЫ_ИЗДАНИЙ_FK"
                                       REFERENCES ВИДЫ_ИЗДАНИЙ (ИД),
  ТИП_ИЗДАНИЯ_ИД  NUMBER(6,0) NOT NULL CONSTRAINT "ТИПЫ_ИЗДАНИЙ_FK"
                                       REFERENCES ТИПЫ_ИЗДАНИЙ (ИД),
  ГОД_ИЗДАНИЯ     NUMBER(4,0) NOT NULL,
  ПОВТОРНОСТЬ     NVARCHAR2(200),
  МАКЕТ_КАРТОЧКИ  NVARCHAR2(4000),
CONSTRAINT "ИЗДАНИЯ_UK" UNIQUE (ЗАГЛАВИЕ_ИД, НОМЕР_ТОМА, АВТОРСКИЙ_ЗНАК,
ББК, ISBN, ИЗДАТЕЛЬСТВО_ИД, ВИД_ИЗДАНИЯ_ИД, ТИП_ИЗДАНИЯ_ИД, ГОД_ИЗДАНИЯ,
ПОВТОРНОСТЬ, МАКЕТ_КАРТОЧКИ)
);


-- Комментарии для таблицы и ее столбцов
--
COMMENT ON TABLE ИЗДАНИЯ IS 'Перечень изданий';
COMMENT ON COLUMN ИЗДАНИЯ.ИД IS 'Искусственный первичный уникальный
идентификатор';
COMMENT ON COLUMN ИЗДАНИЯ.ЗАГЛАВИЕ_ИД IS 'Внешний ключ к таблице
Заглавия';
COMMENT ON COLUMN ИЗДАНИЯ.НОМЕР_ТОМА IS 'Номер тома';
COMMENT ON COLUMN ИЗДАНИЯ.АВТОРСКИЙ_ЗНАК IS 'Авторский знак ';
COMMENT ON COLUMN ИЗДАНИЯ.ББК IS 'Шифр библиотечно-библиографической
классификации';
COMMENT ON COLUMN ИЗДАНИЯ.ISBN IS 'Международный стандартный номер
книги';
COMMENT ON COLUMN ИЗДАНИЯ.ИЗДАТЕЛЬСТВО_ИД IS 'Внешний ключ к таблице
Издательства';
COMMENT ON COLUMN ИЗДАНИЯВИД_ИЗДАНИЯ_ИД IS 'Внешний ключ к таблице Виды
изданий';
COMMENT ON COLUMN ИЗДАНИЯ.ТИП_ИЗДАНИЯ_ИД IS 'Внешний ключ к таблице Типы
изданий';
COMMENT ON COLUMN ИЗДАНИЯ.ГОД_ИЗДАНИЯ IS 'Год издания книги';
COMMENT ON COLUMN ИЗДАНИЯ.ПОВТОРНОСТЬ IS 'Повторность издания';
```

```
COMMENT ON COLUMN ИЗДАНИЯ.МАКЕТ_КАРТОЧКИ IS 'Макет аннотированной ката-
ложной карточки';

-- Создание генератора уникальной последовательности значений, которые
-- будут использоваться в триггере ИЗДАНИЯ_BIR для выработки значений
-- первичного ключа таблицы Издания
--
CREATE SEQUENCE ИЗДАНИЯ_ПОСЛ INCREMENT BY 1 START WITH 1;

-- Создание триггера для выработки значений первичного ключа таблицы
--
CREATE OR REPLACE TRIGGER ИЗДАНИЯ_BIR
  BEFORE INSERT ON издания
 FOR EACH ROW
BEGIN
  SELECT издания_посл.NEXTVAL INTO :new.ид FROM dual;
end издания_bir;
/
```

Листинг 12.5. Участники

```
-- Предложение для создания таблицы
--
CREATE TABLE УЧАСТНИКИ
(
  ЧЕЛОВЕК_ИД NUMBER(6,0) NOT NULL CONSTRAINT "ЛЮДИ_FK"
                                  REFERENCES ЛЮДИ (ИД),
  ИЗДАНИЕ_ИД NUMBER(6,0) NOT NULL CONSTRAINT "ИЗДАНИЕ_FK"
                                  REFERENCES ИЗДАНИЯ (ИД),
  СОЗДАТЕЛЬ_ИД NUMBER(6,0) NOT NULL CONSTRAINT "СОЗДАТЕЛИ_FK"
                                  REFERENCES СОЗДАТЕЛИ (ИД),
  ПРИМЕЧАНИЕ   NVARCHAR2(200),
CONSTRAINT "УЧАСТНИКИ_PK" PRIMARY KEY (ИЗДАНИЕ_ИД, СОЗДАТЕЛЬ_ИД,
ЧЕЛОВЕК_ИД)
);

-- Комментарии для таблицы и ее столбцов
--
COMMENT ON TABLE УЧАСТНИКИ IS 'Люди, принимавшие участие в создании издания';
```

```
COMMENT ON COLUMN УЧАСТНИКИ.ЧЕЛОВЕК_ИД IS 'Внешний ключ к таблице Люди';
COMMENT ON COLUMN УЧАСТНИКИ.ИЗДАНИЕ_ИД IS 'Внешний ключ к таблице Издания';
COMMENT ON COLUMN УЧАСТНИКИ.СОЗДАТЕЛЬ_ИД IS 'Внешний ключ к таблице Создатели';
COMMENT ON COLUMN УЧАСТНИКИ.ПРИМЕЧАНИЕ IS 'Примечание';
```

Листинг 12.6. Экземпляры

```
-- Предложение для создания таблицы
--
CREATE TABLE ЭКЗЕМПЛЯРЫ
(
  ИД NUMBER(6,0) NOT NULL CONSTRAINT "ЭКЗЕМПЛЯРЫ_PK" PRIMARY KEY,
  ИЗДАНИЕ_ИД NUMBER(6,0) NOT NULL CONSTRAINT "ИЗДАНИЯ_FK"
                                  REFERENCES ИЗДАНИЯ (ИД),
  МЕСТО_ИД   NUMBER(6,0) NOT NULL CONSTRAINT "МЕСТА_FK"
                                  REFERENCES МЕСТА (ИД),
  ДАТА_ПРЕОБРЕТЕНИЯ DATE NOT NULL,
  ЦЕНА NUMBER(9,2) NOT NULL,
  ДАТА_СПИСАНИЯ DATE,
CONSTRAINT "ЭКЗЕМПЛЯРЫ_UK" UNIQUE (ИЗДАНИЕ_ИД, МЕСТО_ИД,
ДАТА_ПРЕОБРЕТЕНИЯ, ЦЕНА)
);

-- Описания назначения таблицы и ее столбцов
--
COMMENT ON TABLE ЭКЗЕМПЛЯРЫ IS 'Перечень экземпляров';
COMMENT ON COLUMN ЭКЗЕМПЛЯРЫ.ИД IS 'Искусственный первичный уникальный
идентификатор';
COMMENT ON COLUMN ЭКЗЕМПЛЯРЫ.ИЗДАНИЕ_ИД IS 'Внешний ключ к таблице Издания';
COMMENT ON COLUMN ЭКЗЕМПЛЯРЫ.МЕСТО_ИД IS 'Внешний ключ к таблице Места';
COMMENT ON COLUMN ЭКЗЕМПЛЯРЫ.ДАТА_ПРЕОБРЕТЕНИЯ IS 'Дата преобретения
экземпляра';
COMMENT ON COLUMN ЭКЗЕМПЛЯРЫ.ЦЕНА IS 'Цена экземпляра';
COMMENT ON COLUMN ЭКЗЕМПЛЯРЫ.ДАТА_СПИСАНИЯ IS 'Дата списания экземпляра';

-- Создание генератора уникальной последовательности значений, которые
-- будут использоваться в триггере ЭКЗЕМПЛЯРЫ_BIR для выработки значений
-- первичного ключа таблицы Экземпляры
--
```

```
CREATE SEQUENCE ЭКЗЕМПЛЯРЫ_ПОСЛ INCREMENT BY 1 START WITH 1;

-- Создание триггера для выработки значений первичного ключа таблицы
--
CREATE OR REPLACE TRIGGER ЭКЗЕМПЛЯРЫ_BIR
  BEFORE INSERT ON экземпляры
 FOR EACH ROW
BEGIN
  SELECT экземпляры_посл.NEXTVAL INTO :new.ид FROM dual;
end экземпляры_bir;
/
```

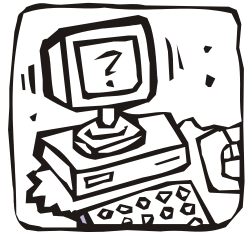
Листинг 12.7. Выдачи

```
-- Предложение для создания таблицы
--
CREATE TABLE ВЫДАЧИ
(
  ЧЕЛОВЕК_ИД    NUMBER(6,0) NOT NULL CONSTRAINT "ЛЮДИ_ВЫДАЧИ_FK"
                                     REFERENCES ЛЮДИ (ИД),
  ЭКЗЕМПЛЯР_ИД  NUMBER(6,0) NOT NULL CONSTRAINT "ЭКЗЕМПЛЯРЫ_FK"
                                     REFERENCES ЭКЗЕМПЛЯРЫ (ИД),
  СРОК          NUMBER(3,0) NOT NULL,
  ДАТА_ВЫДАЧИ   DATE DEFAULT sysdate NOT NULL,
  ДАТА_ВОЗВРАТА DATE,
CONSTRAINT "ВЫДАЧИ_UK" UNIQUE (ЭКЗЕМПЛЯР_ИД, ЧЕЛОВЕК_ИД, ДАТА_ВЫДАЧИ)
);

-- Комментарии для таблицы и ее столбцов
--
COMMENT ON TABLE ВЫДАЧИ IS 'Читательские билеты';
COMMENT ON COLUMN ВЫДАЧИ.ЧЕЛОВЕК_ИД IS 'Внешний ключ к таблице Люди';
COMMENT ON COLUMN ВЫДАЧИ.ЭКЗЕМПЛЯР_ИД IS 'Внешний ключ к таблице
Экземпляры';
COMMENT ON COLUMN ВЫДАЧИ.СРОК IS 'Срок, на который выдан экземпляр
(в днях)';
COMMENT ON COLUMN ВЫДАЧИ.ДАТА_ВЫДАЧИ IS 'Дата выдачи экземпляра';
COMMENT ON COLUMN ВЫДАЧИ.ДАТА_ВОЗВРАТА IS 'Дата возврата экземпляра';
```

ЧАСТЬ V

ЯЗЫК SQL. СОЗДАНИЕ БАЗЫ ДАННЫХ



Глава 13

Создание базы данных и ее основных объектов

13.1. О языке определения данных (DDL)

Большинство пользователей работает с существующими базами данных с помощью каких-либо приложений. Однако некоторым из них иногда требуется создавать как сами базы данных, так и новые объекты в существующих базах данных. Для этого необходимо познакомиться с языком определения данных, с помощью которого можно:

- создать новую базу данных;
- создать так называемую схему в существующей базе данных;
- определить структуру новой таблицы и создать ее;
- удалить таблицу, которая больше не нужна;
- изменить определение существующей таблицы;
- создать индексы для ускорения доступа к таблицам;
- управлять физическим размещением данных;
- создать процедуру, функцию или триггер.

В большинстве случаев предложения DDL обеспечивают высокий уровень доступа к данным и позволяют пользователю не вникать в детали хранения информации в базе данных на физическом уровне. Они оперируют абстрактными объектами базы данных, такими как таблицы и столбцы. Однако DDL не может совершенно не затрагивать вопросов, связанных с физической памятью. Инструкции и предложения DDL, управляющие физической памятью, могут быть разными в различных СУБД. Ядро языка определения данных образуют три глагола:

- `CREATE` (создать) — позволяет определить и создать объект базы данных;
- `DROP` (удалить) — служит для удаления существующего объекта базы данных;

- ALTER (изменить) — с его помощью можно изменить определение объекта базы данных.

Все основные реляционные СУБД позволяют использовать три указанных глагола во время своей работы. Таким образом, структура реляционной базы данных является динамической. Например, СУБД может создавать, удалять или изменять таблицы, одновременно с этим обеспечивая доступ пользователей к базе данных. Это одно из главных преимуществ реляционных баз данных по сравнению с более ранними системами, в которых изменять структуру базы данных можно было только после прекращения работы СУБД. Это означает, что с течением времени база данных может расти и изменяться. Ее промышленная эксплуатация может продолжаться в то время, когда в базу данных добавляются все новые таблицы и модули.

Хотя DDL и DML являются двумя отдельными частями SQL, в большинстве реляционных СУБД такое разделение существует лишь на абстрактном уровне. Обычно инструкции DDL и DML в СУБД абсолютно равноправны, и их можно произвольно чередовать как в интерактивном, так и в программном SQL. Если программе или пользователю требуется таблица для временного хранения результатов, они могут создать эту таблицу, заполнить ее, проделать с данными необходимую работу и затем удалить таблицу.

13.2. Создание базы данных и схем

В СУБД крупных корпораций новые базы данных создаются администратором. В СУБД, установленных на серверах более низкого уровня, отдельным пользователям разрешается создавать собственные базы данных, но обычно в таких СУБД базы данных создаются централизованно, а пользователи затем работают с ними. Если же вы сами создаете базу данных на персональном компьютере, то вы являетесь и ее администратором и пользователем. Следует отметить, что в этом абзаце речь идет о физической базе данных (наборе физических файлов операционной системы, ассоциирующихся с именем базы данных).

При создании базы данных, например, в СУБД Oracle с помощью предложения

```
CREATE DATABASE [имя_базы_данных]
  { USER SYS IDENTIFIED BY пароль
  | USER SYSTEM IDENTIFIED BY пароль
  }... ;
```

создается новая пустая база данных с указанным именем *имя_базы_данных* (до восьми символов на латинице) и двумя пользователями с именами SYS

и `SYSTEM`, имеющими пароли по умолчанию `change_on_install` и `manager` соответственно. Если имя опускается, то его создает сама СУБД, а пароли указанных суперпользователей могут быть затем изменены администратором.

Полные тексты команд `CREATE DATABASE` и `ALTER DATABASE` достаточно сложны (в Oracle они занимают более 50 страниц документации), и мы не будем их обсуждать. Создание же базы данных для поставляемого на компакт-диске Oracle Database 10*g* Express Edition подробно описано в *приложении А*.

К сожалению, под словосочетанием "база данных" нередко подразумевают или саму СУБД, или схему базы данных — именованный набор объектов базы данных (таблиц, представлений, процедур, триггеров, функций и др.), как правило, ассоциирующийся с каким-либо пользователем. Например, при создании базы данных были определены два таких суперпользователя и их схемы, где хранятся системные данные. Затем можно в этой базе данных создать множество необходимых схем (пользователей).

В Oracle понятия "схема" и "пользователь" слились воедино. Формально два разных слова "user" и "schema" используются в Oracle для обозначения одного и того же: "схемы-пользователя". Документация на этот счет стыдливо говорит, что "при заведении пользователя

`CREATE USER` *`имя_пользователя`* `IDENTIFIED BY` *`пароль`*

автоматически создается схема с таким же именем", т. е. в системе понятий Oracle "схема" = "пользователь".

Например, администратор базы данных СПбГУ ИТМО создал пустые схемы для студентов, которым необходим при обучении доступ к учебной базе данных, задав их имена (идентификатор студента) и сгенерировав пароли. Кроме того, этим студентам даны права на создание объектов в своих схемах и права на чтение таблиц схемы с именем учебной базы данных (см. *часть VI*), в которой хранится иллюстрационная база данных, используемая в лабораторном практикуме.

13.3. Создание таблиц

Манипулирование таблицами — наиболее распространенный вид деятельности, который администраторы баз данных и пользователи осуществляют при работе с объектами баз данных. В отличие от рассмотренных ранее предложений языка DML, предложения `CREATE TABLE` для различных СУБД, достаточно сильно отличаются от стандарта, хотя в основном поддерживают его функции.

Рассмотрим для примера подробное описание таблицы `Люди` базы данных (схемы) "LIBRARY" (см. *разд. 12.3*), созданной в СУБД Oracle.

Листинг 13.1. Пример создания таблицы Люди с физическими параметрами хранения

```
CREATE TABLE ЛЮДИ
 (
   ИД NUMBER(6,0) NOT NULL CONSTRAINT "ЛЮДИ_PK" PRIMARY KEY
    USING INDEX
    TABLESPACE "USERS"
    PCTFREE 10
    INITRANS 2
    MAXTRANS 255
    STORAGE
     (
       INITIAL 64K
       NEXT 1024K
       MINEXTENTS 1
       MAXEXTENTS unlimited
       PCTINCREASE 0
       FREELISTS 1
       FREELIST GROUPS 1
       BUFFER_POOL DEFAULT
     ),
  ФАМИЛИЯ VARCHAR2(50) NOT NULL,
       ИМЯ VARCHAR2(50) NOT NULL,
       ОТЧЕСТВО VARCHAR2(50) NOT NULL,
       ПСЕВДОНИМ VARCHAR2(50),
       ДАТА_РОЖДЕНИЯ DATE NOT NULL,
       ПОЛ CHAR(1) NOT NULL CONSTRAINT "Пол может быть М или Ж" CHECK
(пол in ('М','Ж')),
       АДРЕС VARCHAR2(200),
       ТЕЛЕФОН VARCHAR2(50),
        CONSTRAINT "ЛЮДИ_UK" UNIQUE (ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО,
ДАТА_РОЖДЕНИЯ, ПОЛ)
    USING INDEX
    TABLESPACE "USERS"
    PCTFREE 10
    INITRANS 2
    MAXTRANS 255
    STORAGE
     (
       INITIAL 64K
       NEXT 1024K
       MINEXTENTS 1
```

```
      MAXEXTENTS unlimited
      PCTINCREASE 0
      FREELISTS 1
      FREELIST GROUPS 1
      BUFFER_POOL DEFAULT
    )
  )
   TABLESPACE "USERS"
   PCTFREE 10
   PCTUSED 40
   INITRANS 2
   MAXTRANS 255
   NOCOMPRESS LOGGING
   STORAGE
    (
      INITIAL 64K
      NEXT 1024K
      MINEXTENTS 1
      MAXEXTENTS unlimited
      PCTINCREASE 0
      FREELISTS 1
      FREELIST GROUPS 1
      BUFFER_POOL DEFAULT
    );
```

В листинг 13.1 включено полное описание таблицы, содержащее физические параметры хранения.

Перейдем теперь к описанию упрощенного синтаксиса этого предложения для СУБД Oracle, достаточно близкого к стандарту.

13.3.1. Описание таблицы

В приведенном синтаксисе опущены фразы, касающиеся определения физических атрибутов таблицы и их параметров, параметров хранения (`TABLESPACE`, `CLUSTER` и пр.), параметров секционирования, внешних и сжатых таблиц, т. е. всего того, что создается СУБД по умолчанию и при желании может быть изменено после детального знакомства с документацией (см. прилагаемый к книге компакт-диск). Там же можно познакомиться с синтаксисом объектных таблиц и таблиц XML.

При выполнении предложения `CREATE TABLE` создается постоянная или глобальная временная таблица либо путем объявления структуры, либо путем

ссылки на существующую таблицу. Упрощенный синтаксис этого предложения имеет вид:

```
CREATE TABLE [ GLOBAL TEMPORARY ] TABLE [имя_схемы.]имя_таблицы
( { имя_столбца тип_данных[(длина)] [DEFAULT выражение]
  [ ENCRYPT [ USING 'алгоритм_шифрования' ] [ IDENTIFIED BY пароль ]
  [ [NO] SALT ] ]
[ограничения_для_столбца] ... }
[ , { имя_столбца тип_данных[(длина)] [DEFAULT выражение]
  [ ENCRYPT [ USING 'алгоритм_шифрования' ] [ IDENTIFIED BY пароль ]
  [ [NO] SALT ] ]
[ограничения_для_столбца] ... } ]
[ {, ограничение_для_таблицы } ... ] )
[AS подзапрос];
```

где

`[GLOBAL TEMPORARY]`

Объявляется временная (`GLOBAL TEMPORARY`) таблица с глобальной областью действия. Глобальные временные таблицы доступны из всех активных сеансов, но они автоматически удаляются, когда завершается создавший их сеанс.

`имя_столбца`

Указывается имя столбца.

`тип_данных`

Связывает со столбцом с именем *`имя_столбца`* определенный тип данных. Для тех типов, которые позволяют указать их длину, существует дополнительный параметр *`длина`*, например `VARCHAR2(255)`.

`DEFAULT` *`выражение`*

Столбец будет использовать значение выражения, если предложение `INSERT` или `UPDATE` не вводит никакого значения. Выражение может представлять собой строку или число, пользовательскую или системную функцию (см. *разд. 4.6* и *18.3*), но не может содержать подзапрос, ссылки на другие столбцы и псевдостолбцы `CURRVAL`, `NEXTVAL`, `LEVEL` и `ROWNUM`.

`ENCRYPT`

Позволяет зашифровать данные столбца, имеющего один из следующих типов: `CHAR`, `NCHAR`, `VARCHAR2`, `NVARCHAR2`, `NUMBER` и `DATE`.

`USING '`*`алгоритм_шифрования`*`'`

Позволяет указать один из алгоритмов шифрования данных. Если эта опция не указана, то для шифрования используется алгоритм AES (Advanced Encryption Standard, усовершенствованный стандарт шифрования) с 192-битовым ключом.

`IDENTIFIED BY` *`пароль`*

При указании пароля СУБД формирует из него ключ шифрования для этого столбца.

`[NO] SALT`

Если в шифруемых данных много повторяющихся значений и, следовательно, их можно угадать, то для предотвращения этого к данным добавляется "соль" (salt), которая делает зашифрованные значения разными, даже если входные данные совпадают. По умолчанию "соль" применяется.

Так как столбец с "соль" нельзя индексировать, то для создания индексируемого столбца необходимо указывать `NO SALT`.

`ограничения_для_столбца`

Необязательный параметр, с помощью которого задается одно или несколько (в синтаксисе указано с помощью "...") ограничений целостности на значения столбца (синтаксис этого выражения приведен в *разд. 13.3.2*).

`ограничение_для_таблицы`

Необязательный параметр, с помощью которого задаются разделенные запятыми ограничения целостности таблицы (синтаксис этого выражения приведен в *разд. 13.3.2*).

`AS-подзапрос`

Необязательный параметр, с помощью которого можно вставить в создаваемую таблицу строки данных в соответствии с подзапросом; может также использоваться для копирования как описания (без ограничений целостности), так и содержимого существующей таблицы (копирование).

13.3.2. Ограничения целостности

Как уже отмечалось ранее, ограничения целостности задаются с помощью параметров *`ограничения_для_столбца`* и *`ограничение_для_таблицы`*. Они описывают правила, применимые к таблицам при их создании и оговаривают допустимые значения столбцов и (или) допустимые сочетания этих значений. Кроме того, ряд ограничений целостности можно создавать с помощью триггеров.

Ограничения целостности для столбца

Синтаксис параметра *`ограничения_для_столбца`* имеет вид:

```
[CONSTRAINT имя_ограничения]
{ [NOT] NULL | UNIQUE | PRIMARY KEY
  | REFERENCES [имя_схемы.]имя_таблицы [(имя_столбца)]
```

```
    [ON DELETE { CASCADE | SET NULL } ]
    | CHECK (условие_поиска) }
```

где

`CONSTRAINT` *имя_ограничения*

Необязательный параметр, используемый для задания имени ограничения, если имя не задано, то СУБД присваивает ограничению свое имя, вывод которого на экран при ошибке (например, отсутствии уникальности вводимого значения) резко усложняет процесс идентификации места ошибки.

`PRIMARY KEY`

Первичный ключ (не может быть назначен для столбцов с типом данных `BLOB`, `CLOB`, `NCLOB`, `ARRAY`); в столбце, объявленном первичным ключом, значения уставливаются уникальными и не пустыми (`NOT NULL`); по значениям такого столбца автоматически строится индекс.

`UNIQUE`

Отличается от `PRIMARY KEY` тем, что в связанном с ним столбце допускаются пустые (`NULL`) значения.

`[NOT] NULL`

Указывает, что в этот столбец можно поместить пустое (`NULL`) или некоторое непустое (`NOT NULL`) значение.

`REFERENCES`

Ссылка на таблицу `[`*имя_схемы*`.]`*имя_таблицы*, в которой находится первичный или уникальный ключ (когда имя столбца опущено, автоматически выбирается первичный ключ). `REFERENCES` допускает ввод пустых значений (для их исключения надо одновременно вводить ограничение `NOT NULL`).

`ON DELETE`

Определяет действия, которые выполняет СУБД для обеспечения ссылочной целостности внешнего ключа, когда удаляется значение связанного первичного или уникального ключа.

`CASCADE`

Указывает, что в том случае, если удаляется значение первичного или уникального ключа, СУБД выполняет то же самое действие над внешним ключом. Если `ON DELETE CASCADE` опущено, то строки с ключевыми значениями в таблице с первичным или уникальным ключом не могут быть удалены, пока не будут удалены все ссылающиеся на них строки из данной таблицы (таблицы с внешним ключом).

`SET NULL`

Указывает, что в случае удаления первичного или уникального ключа база данных установит значение внешнего ключа в `NULL`.

`CHECK`

Используется для контроля данных по условию, текст которого размещается в скобках. CHECK допускает ввод пустых (NULL) значений и не может содержать:

- запросов на обращение к другим таблицам или другим строкам данной таблицы;
- обращений к системным переменным и константам (например, SYSDATE — текущая дата).

Ограничения целостности для таблицы

В тех случаях, когда ограничение относится не к одному столбцу, а к их комбинации, используются ограничения на уровне всей таблицы (листинг 13.2).

Синтаксис параметра *ограничения_для_таблицы* имеет вид:

```
[ CONSTRAINT имя_ограничения ]
{ UNIQUE (имя_столбца [,имя_столбца ]...)
| PRIMARY KEY (имя_столбца [,имя_столбца ]...)
| FOREIGN KEY (имя_столбца [,имя_столбца ]...)
    REFERENCES [имя_схемы.]имя_таблицы [(имя_столбца[,имя_столбца ]...)]
       [ON DELETE { CASCADE | SET NULL } ]
| CHECK (условие_поиска) }
```

где параметры имеют тот же смысл, что и для *ограничений_для_столбца*.

Листинг 13.2. Пример создания таблицы Меню

```
CREATE TABLE МЕНЮ
(
  СТРОКА       NUMBER(9) CONSTRAINT "Строка меню не уникальна !"
                         PRIMARY KEY,
  КОД_ТРАПЕЗЫ NUMBER(1) NOT NULL CONSTRAINT "Номер трапезы в Меню ?"
                                 REFERENCES ТРАПЕЗЫ (КОД_ТРАПЕЗЫ),
  КОД_БЛЮДА    NUMBER(2) NOT NULL CONSTRAINT "Код блюда в Меню ? "
                                 REFERENCES БЛЮДА (КОД_БЛЮДА),
  ДАТА         DATE not NULL,
  CONSTRAINT "Данные строки не уникальны !" UNIQUE (КОД_ТРАПЕЗЫ,
КОД_БЛЮДА, ДАТА)
);
```

Ограничения целостности, создаваемые триггерами

С помощью команд CREATE TABLE и ALTER TABLE нельзя реализовать любые ограничения целостности. Например, мы не сможем обеспечить с их помощью

такие ограничения таблицы Люди, как: "дата рождения должна быть в пределах от 80 лет до 15 лет" и "первые буквы фамилии, имени и отчества должны быть большими, а остальные — малыми". Действительно, в CHECK нельзя включать обращения к текущей дате (SYSDATE) и сравнительно сложным процедурам для проверки правильности ввода, например, таких имен как Смирнов-Сокольский, Жан-Жак Руссо, Гай Юлий Цезарь и т. п.

Указанные ранее и многие другие ограничения целостности можно вводить с помощью триггеров (см. *часть VI*). *Триггер* — это сочетание хранимой в базе данных программы и события, которое заставляет ее выполняться. Такими событиями могут быть: ввод новой строки таблицы, изменение значений одного или нескольких ее столбцов и (или) удаление строки таблицы. При любом из этих событий автоматически запускаются один или несколько заранее созданных триггеров, которые производят проверку запрограммированных в них условий и, если они не выполняются, отменяют ввод, изменение или удаление, посылая об этом заранее подготовленное сообщение пользователю.

Только с помощью триггеров мы сможем обеспечить "автоматическое" наращивание значений суррогатного первичного ключа или проверки правильности ввода фамилии, как это сделано в тексте триггера ЛЮДИ_BIR в листинге 12.1.

Для написания текста триггера надо овладеть языком PL/SQL (см. *часть VI*).

13.3.3. Комментарии к описанию таблицы

Для сохранения в базе данных комментария к таблице и (или) любому ее столбцу используется предложение COMMENT, синтаксис которого имеет вид:

```
COMMENT ON { TABLE [ имя_схемы.]{имя_таблицы | имя_представления}
  | COLUMN [ имя_схемы.]{имя_таблицы. | имя_представления.} имя_столбца }
IS 'текст_комментария';
```

где текст_комментария — любой набор символов (кроме апострофов), заключенный в апострофы ('), длина которого не превышает 4000 байт. Правила создания строковых констант описаны в *разд. 4.4.3*.

Кроме предложения COMMENT для включения пояснений в любое предложение SQL и командные блоки PL/SQL можно либо расположить текст пояснения между парами символов **/*** и ***/**, либо предварить его двумя дефисами (--).

С помощью первого способа можно включать многострочные пояснения. Для включения многострочных пояснений вторым способом приходится помещать -- перед каждой их строкой. Следует заметить, что комментарии,

отмеченные --, могут располагаться либо в отдельных строках, либо в конце строк текста комментируемого предложения (листинг 13.3).

Листинг 13.3. Пример создания таблицы Меню с комментариями

```
CREATE TABLE МЕНЮ
(
  СТРОКА       NUMBER(9) CONSTRAINT "Строка меню не уникальна !"
                         PRIMARY KEY,
  КОД_ТРАПЕЗЫ NUMBER(1) NOT NULL CONSTRAINT "Номер трапезы в Меню ?"
                                 REFERENCES ТРАПЕЗЫ (КОД_ТРАПЕЗЫ),
  КОД_БЛЮДА    NUMBER(2) NOT NULL CONSTRAINT "Код блюда в Меню ? "
                                 REFERENCES БЛЮДА (КОД_БЛЮДА),
  ДАТА         DATE not NULL,
-- табличное ограничение
  CONSTRAINT "Данные строки не уникальны !" UNIQUE (КОД_ТРАПЕЗЫ,
КОД_БЛЮДА, ДАТА)
);
   COMMENT ON TABLE МЕНЮ  IS 'Меню, предлагаемое шеф-поваром на ...';
   COMMENT ON COLUMN МЕНЮ.СТРОКА IS 'Номер строки в меню';
   COMMENT ON COLUMN МЕНЮ.КОД_ТРАПЕЗЫ IS 'Номер трапезы';
   COMMENT ON COLUMN МЕНЮ.КОД_БЛЮДА IS 'Код блюда';
   COMMENT ON COLUMN МЕНЮ.ДАТА IS 'Меню составлено на дату';
```

13.4. Изменение таблиц

Так же, как и для создания таблиц, синтаксис их изменения для различных СУБД достаточно сильно отличается от стандарта, хотя в основном поддерживает его функции. Поэтому приведем здесь упрощенный синтаксис для СУБД Oracle, достаточно близкий к стандарту. Опустим фразы, касающиеся изменения физических атрибутов таблицы и их параметров, параметров хранения (`TABLESPACE`, `CLUSTER` и пр.), параметров секционирования, внешних и сжатых таблиц, т. е. всего того, что создается СУБД по умолчанию и при желании может быть изменено после детального знакомства с документацией (см. прилагаемый к книге компакт-диск). Там же можно познакомиться с синтаксисом объектных таблиц и таблиц XML.

При выполнении предложения `ALTER TABLE` изменяется описание таблицы. Упрощенный синтаксис имеет вид:

```
ALTER TABLE [имя_схемы.]имя_таблицы
-- изменение имени таблицы
RENAME TO новое_имя_таблицы
```

```
-- добавление новых столбцов и их параметров
| ADD ( { имя_столбца тип_данных[(длина)] [DEFAULT выражение]
  [ ENCRYPT [ USING 'алгоритм_шифрования' ] [ IDENTIFIED BY пароль ]
  [ [NO] SALT ] ]
[ограничения_для_столбца] ... }
[ , { имя_столбца тип_данных[(длина)] [DEFAULT выражение]
  [ ENCRYPT [ USING 'алгоритм_шифрования' ] [ IDENTIFIED BY пароль ]
  [ [NO] SALT ] ]
[ограничения_для_столбца] ... } ]
-- добавление табличных ограничений
| ADD { ограничение_для_таблицы [ограничение_для_таблицы]... }
-- изменение параметров столбца в существующей таблице
| MODIFY ( { имя_столбца тип_данных[(длина)] [DEFAULT выражение]
  [ ENCRYPT [ USING 'алгоритм_шифрования' ] [ IDENTIFIED BY пароль ]
  [ [NO] SALT ] | DECRYPT ]
[ограничения_для_столбца] ... }
[ , { имя_столбца тип_данных[(длина)] [DEFAULT выражение]
  [ ENCRYPT [ USING 'алгоритм_шифрования' ] [ IDENTIFIED BY пароль ]
  [ [NO] SALT ] | DECRYPT ]
[ограничения_для_столбца] ... } ]
-- удаление и изменение параметров таблицы
| DROP { COLUMN имя_столбца | (имя_столбца [,имя_столбца ]...) }
  [ { CASCADE CONSTRAINTS | INVALIDATE }
      [ CASCADE CONSTRAINTS | INVALIDATE ]... ]
| MODIFY { CONSTRAINT имя_ограничения | PRIMARY KEY
          | UNIQUE (имя_столбца [,имя_столбца ]...) } ENABLE | DISABLE
| RENAME CONSTRAINT старое_имя TO новое_имя
| DROP { { PRIMARY KEY | UNIQUE (имя_столбца [,имя_столбца ]...) }
     [ CASCADE ] [ { KEEP | DROP } INDEX ] | CONSTRAINT имя_ограничения
     [ CASCADE ] }
```

Здесь большинство параметров совпадает с аналогичными параметрами предложения `CREATE TABLE` и было рассмотрено в *разд. 13.3*. Опишем новые.

`INVALIDATE`

При указании этого ключевого слова любой объект, зависящий от удаленного объекта, становится нерабочим и неиспользуемым до перекомпиляции или следующего использования.

`{KEEP I DROP) INDEX`

Позволяет сохранить (`KEEP`) или удалить (`DROP`) индекс, связанный с уникальным или первичным ключом (листинг 13.4).

Листинг 13.4. Пример создания таблицы `Люди` с ограничениями целостности, добавленными предложениями `ALTER TABLE`

```
-- Предложение для создания таблицы
--
CREATE TABLE ЛЮДИ
( ИД             NUMBER(6,0) NOT NULL,
  ФАМИЛИЯ        NVARCHAR2(50) NOT NULL,
  ИМЯ            NVARCHAR2(50) NOT NULL,
  ОТЧЕСТВО       NVARCHAR2(50) NOT NULL,
  ПСЕВДОНИМ      NVARCHAR2(50),
  ДАТА_РОЖДЕНИЯ DATE NOT NULL,
  ПОЛ            NCHAR(1) NOT NULL,
  АДРЕС          NVARCHAR2(200),
  ТЕЛЕФОН        NVARCHAR2(50)
);

-- Описания назначения таблицы и ее столбцов
--
COMMENT ON TABLE ЛЮДИ  IS 'Список читателей и/или создателей изданий';
COMMENT ON COLUMN ЛЮДИ.ИД IS 'Уникальный номер человека';
COMMENT ON COLUMN ЛЮДИ.ФАМИЛИЯ IS 'Фамилия человека';
COMMENT ON COLUMN ЛЮДИ.ИМЯ IS 'Имя человека';
COMMENT ON COLUMN ЛЮДИ.ОТЧЕСТВО IS 'Отчество человека';
COMMENT ON COLUMN ЛЮДИ.ПСЕВДОНИМ IS 'Псевдоним человека';
COMMENT ON COLUMN ЛЮДИ.ДАТА_РОЖДЕНИЯ IS 'Дата рождения человека';
COMMENT ON COLUMN ЛЮДИ.ПОЛ IS 'Пол человека';
COMMENT ON COLUMN ЛЮДИ.АДРЕС IS 'Адрес человека';
COMMENT ON COLUMN ЛЮДИ.ТЕЛЕФОН IS 'Телефон человека';

-- Предложения для изменения таблицы
-- 1. Создание первичного ключа
ALTER TABLE ЛЮДИ ADD CONSTRAINT "ЛЮДИ_PK" PRIMARY KEY (ИД);
-- 2. Создание ограничения для проверки уникальности комбинации значений
ALTER TABLE ЛЮДИ ADD CONSTRAINT "ЛЮДИ_UK" UNIQUE (ФАМИЛИЯ, ИМЯ, ОТЧЕСТВО,
ДАТА_РОЖДЕНИЯ, ПОЛ);
-- 3. Создание ограничения для проверки правильности описания пола
ALTER TABLE ЛЮДИ ADD CONSTRAINT "Пол может быть М или Ж" CHECK (ПОЛ IN
('М','Ж'));
```

13.5. Удаление таблиц

При выполнении предложения

```
DROP TABLE имя_таблицы [CASCADE CONSTRAINTS] [PURGE]
```

удаляется указанная таблица, очищаются все ее данные, удаляются все индексы и триггеры, созданные на ее основе (даже те, которые принадлежат к другим схемам), становятся недействительными все права доступа и все зависимые объекты (такие как представления, хранимые процедуры и т. п.).

Фраза `CASCADE CONSTRAINTS` используется для удаления по всей базе данных всех ограничений ссылочной целостности, которые зависят от первичного или уникального ключа удаленной таблицы. Без этой фразы нельзя удалить таблицу с зависимыми ссылочными ограничениями.

Предложение `DROP TABLE` эффективно для стандартных таблиц, индекс-таблиц и объектных таблиц. Удаляемая таблица перемещается в корзину, если только не указано ключевое слово `PURGE`, которое заставляет систему Oracle немедленно освободить все место, занятое таблицей.

13.6. Создание последовательностей

Последовательность (sequence) генерирует уникальные порядковые номера, которые могут использоваться как значения числовых столбцов таблиц базы данных. Последовательности упрощают решение многих задач, когда требуется генерация уникальных числовых значений для строк одной или нескольких таблиц.

Для создания последовательностей используется команда `CREATE SEQUENCE`, упрощенный синтаксис которой имеет вид:

```
CREATE SEQUENCE [схема.]имя_последовательности
     [INCREMENT BY целое_число]
     [START WITH целое_число];
```

где *схема* — это схема, в которой создается последовательность (если она опущена, то последовательность создается в схеме текущего пользователя).

имя_последовательности — имя создаваемой последовательности.

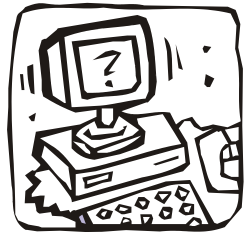
`INCREMENT BY` задает интервал между значениями последовательности. Это значение может быть любым (отличным от нуля) положительным или отрицательным целым числом (для возрастающей или убывающей последовательности), по умолчанию принимается интервал, равный 1.

START WITH определяет первый генерируемый номер последовательности (по умолчанию он равен 1).

Для выбора следующего уникального значения последовательности используется псевдостолбец *имя_последовательности*.NEXTVAL, а для текущего — *имя_последовательности*.CURRVAL.

Например, создадим последовательность с именем люди_посл для формирования первичного ключа таблицы Люди (см. листинг 12.1).

```
create sequence люди_посл
start with 1
increment by 1;
```



Глава 14

Системный каталог (словарь данных)

14.1. Что такое системный каталог

Для того чтобы эффективно управлять данными, СУБД должна отслеживать большое количество информации, определяющей структуру базы данных. В реляционной базе данных эта информация обычно хранится *в системном каталоге* — совокупности системных таблиц, используемых СУБД для внутренних целей. В файлах, хранящихся в системном каталоге, содержатся описания таблиц, представлений, столбцов, привилегий и других структурных элементов базы данных.

Несмотря на то, что словарь данных предназначен главным образом для внутреннего применения, пользователи базы данных могут получить доступ к системным таблицам с помощью стандартных запросов SQL. Таким образом, реляционная база данных содержит описание своей структуры и пользователь может получить его, выполняя запросы к системным таблицам. Это используется в таких клиентских приложениях общего назначения, как модули формирования запросов и программы генерации отчетов, где для упрощения доступа к базе данных пользователям предоставляется на выбор список таблиц и столбцов.

Таблицы системного каталога создаются автоматически при создании базы данных. Обычно они объединяются под специальным "системным идентификатором пользователя" с таким именем, как `SYSTEM`, `SYS` и т. п.

При обработке инструкций SQL СУБД постоянно обращается к данным системного каталога. Например, чтобы обработать запрос на получение данных из нескольких таблиц, СУБД должна:

- ❒ проверить, существуют ли указанные таблицы;
- ❒ убедиться, что пользователь имеет разрешение на доступ к ним;
- ❒ проверить, существуют ли столбцы, на которые делаются ссылки в запросе;

- установить, к каким таблицам относятся неполные имена столбцов;
- определить тип данных каждого столбца.

Так как информация о структуре базы данных хранится в системных таблицах, СУБД может использовать свои собственные методы и алгоритмы, чтобы быстро и эффективно извлекать информацию, необходимую для выполнения этих задач (см. *главу 15*).

Если бы системные таблицы служили только для удовлетворения внутренних потребностей СУБД, то для пользователей базы данных они не представляли бы никакого интереса. Однако, как уже отмечалось, системные таблицы (или созданные на их основе представления), как правило, доступны и для пользователей. Запросы к системным каталогам разрешены почти во всех СУБД, если их не ограничил администратор базы данных с целью обеспечения безопасности базы данных. С помощью запросов к словарю данных можно получить информацию о структуре базы данных, даже если вы никогда раньше с ней не работали.

Пользователи могут только извлекать информацию из системного каталога. СУБД запрещает пользователям модифицировать системные таблицы непосредственно, так как это может нарушить целостность базы данных. СУБД сама вставляет, удаляет и обновляет строки системных таблиц во время модификации структуры базы данных. Изменения в системных таблицах происходят также в качестве побочного результата выполнения таких инструкций DDL, как `CREATE`, `ALTER`, `DROP`, `COMMIT` и `REVOKE`. В некоторых СУБД даже инструкции DML, например `INSERT` и `DELETE`, могут модифицировать содержимое системных таблиц, отслеживающих количество записей в таблицах.

Для определенности будем далее описывать системный каталог СУБД Oracle, который называется *словарем данных*.

14.2. Словарь данных Oracle

Каждая таблица словаря данных содержит информацию об отдельном структурном элементе базы данных. В состав почти всех коммерческих реляционных СУБД входят, с небольшими различиями, системные таблицы, каждая из которых описывает один из следующих пяти объектов:

- *таблицы* — описывается каждая таблица базы данных: указывается ее имя, владелец, число содержащихся в ней столбцов, их размер и т. д.;
- *столбцы* — описывается каждый столбец базы данных, приводится имя столбца, имя таблицы, которой он принадлежит, тип данных столбца, его размер, разрешены ли значения `NULL` и т. д.;

- *пользователи* — описывается каждый зарегистрированный пользователь базы данных: указываются имя пользователя, его пароль в зашифрованном виде и другие данные;
- *представления* — описывается каждое представление, имеющееся в базе данных: указываются его имя, имя владельца, запрос, являющийся определением представления, и т. д.;
- *привилегии* — описывается каждый набор привилегий, предоставленных в базе данных: приводятся имена тех, кто предоставил привилегии, и тех, кому они предоставлены, указываются сами привилегии, объект, на которые они распространяются, и т. д.

14.2.1. Структура словаря данных

В состав словаря данных базы данных входят *базовые таблицы* и *доступные пользователю представления*.

Основу словаря данных составляет совокупность базовых таблиц, хранящих информацию о базе данных. Эти таблицы читаются и пишутся на системном уровне; они редко используются непосредственно пользователями.

Словарь данных содержит доступные пользователю представления, которые суммируют и отображают в удобном формате информацию из базовых таблиц словаря. Эти представления декодируют информацию базовых таблиц, представляя ее в полезном виде, таком как имена пользователей или таблиц, и используют соединения и фразы `WHERE`, чтобы упростить информацию. Большинство пользователей имеют доступ к этим представлениям вместо базовых таблиц словаря.

Словарь данных базы данных Oracle имеет два основных применения:

- Oracle обращается к словарю данных каждый раз, когда выдается предложение DDL;
- каждый пользователь Oracle может использовать словарь данных как только читаемый справочник по базе данных.

Словарь данных всегда доступен при открытой базе данных. Он размещается в табличном пространстве `SYSTEM`, которое всегда находится в состоянии online, когда база данных открыта.

Словарь данных состоит из нескольких наборов представлений. Во многих случаях такой набор состоит из трех представлений, содержащих аналогичную информацию и отличающихся друг от друга своими префиксами:

- `USER` — представление для пользователя;

- ALL — расширенное представление для пользователя;
- DBA — представление администратора.

Столбцы в каждом представлении набора идентичны, но имеются исключения. В представлениях с префиксом USER обычно нет столбца с именем OWNER (владелец); в представлениях USER под владельцем подразумевается пользователь, выдавший запрос. Некоторые представления DBA имеют дополнительные столбцы, которые содержат информацию, полезную для АБД.

Представления с префиксом USER:

- отражают окружение пользователя в базе данных, включая информацию о созданных им объектах, предоставленных им грантах и т. д.;
- выдают только строки, имеющие отношение к пользователю;
- имеют столбцы, идентичные с другими представлениями, с тем исключением, что столбец OWNER подразумевается (текущий пользователь);
- возвращают подмножество информации, предоставляемой представлениями ALL;
- могут иметь сокращенные общие синонимы для удобства.

Представления с префиксом ALL отражают общее представление о базе данных со стороны пользователя. Эти представления возвращают информацию об объектах, к которым пользователь имеет доступ через общие или явные гранты, помимо тех объектов, которыми владеет этот пользователь.

Представления с префиксом DBA показывают общее представление о базе данных, и предназначены для администраторов базы данных.

Во время своей работы Oracle поддерживает набор "виртуальных" таблиц, в которых регистрируется текущая информация о базе данных. Эти таблицы называются *динамическими таблицами производительности*. Так как динамические таблицы производительности не являются истинными таблицами, большинство пользователей не должно обращаться к ним. Динамические таблицы производительности принадлежат схеме SYS, а их имена начинаются с V_$. По этим таблицам создаются представления, а для представлений создаются синонимы, имена которых начинаются с V$.

14.2.2. Краткое содержимое словаря данных

Словарь данных Oracle 10g включает более 1000 статических представлений и около 500 — динамических. С ними можно познакомиться в документации, приведенной в *приложении А*. Здесь же мы покажем наиболее часто используемые представления (табл. 14.1) и несколько примеров работы с ними.

В табл. 14.1 после некоторых из имен представлений в скобках расположен перечень букв: (a,u), (d,u) или (a,d,u). Это означает, что такое имя является основой для имен двух или трех словарей, получаемых за счет добавки префикса ALL_, DBA_ или USER_. Например, TABLES(a,d,u) — основа для имен словарей: ALL_TABLES, DBA_TABLES и USER_TABLES. Кроме того, некоторые представления имеют синоним (столбец "Синоним"), заменяющий соответствующее представление с префиксом USER_ (например, CAT вместо USER_CATALOG).

Таблица 14.1. *Ряд наиболее используемых представлений словаря данных*

Представление	Синоним	Описание
Таблицы, представления, синонимы, последовательности		
ALL_TABLES(a,d,u)		Описывает все объектные и реляционные таблицы
CATALOG(a,d,u)	CAT	Информация обо всех объектах базы данных
COL		Список столбцов в таблицах пользователя
COL_COMMENTS(a,d,u)		Комментарии для столбцов таблиц и представлений
CONSTRAINTS(a,d,u)		Информация об ограничениях ссылочной целостности в БД
CONS_COLUMNS(a,d,u)		Информация о столбцах, участвующих в ограничениях ссылочной целостности
SEQUENCES(a,d,u)	SEQ	Информация о последовательностях БД
SYNONYMS(a,d,u)	SYN	Информация о синонимах БД
TAB		Таблицы пользователя
TAB_COLUMNS(a,d,u)	COLS	Информация о столбцах таблиц и представлений БД
TAB_COMMENTS(a,d,u)		Комментарии для таблиц и представлений БД
TABLES(a,d,u)	TABS	Информация о таблицах БД
VIEWS(a,d,u)		Информация о представлениях БД

Таблица 14.1 *(продолжение)*

Представление	Синоним	Описание
Объекты		
ARGUMENTS (a,u)		Список всех параметров процедур и функций
ERRORS(a,d,u)		Информация об ошибках компиляции, обнаруженных в БД для процедур, функций, спецификаций пакета и тел пакета
DEPENDENCIES(a,d,u)		Информация о зависимостях объекта в БД
OBJECTS(a,d,u)	OBJ	Информация об объектах базы данных
OBJECT_SIZE(d,u)		Информация о размерах процедур, функций, спецификаций пакетов и тел пакетов в БД
PUBLIC_DEPENDENCY		Информация о зависимостях объекта
SOURCE(a,d,u)		Исходный код процедур, функций, спецификаций пакетов и тел пакетов в БД
TRIGGERS(a,d,u)		Информация о триггерах БД
Привилегии		
COL_PRIVS(a,d,u)		Полномочия для столбцов
DBA_PROFILES		Все профили ограничений ресурсов в БД
DBA_ROLES		Информация о ролях в БД
ROLE_PRIVS(d,u)		Роли, назначенные пользователю
ROLE_ROLE_PRIVS		Роли, назначенные другим ролям
ROLE_SYS_PRIVS		Системные полномочия, предоставляемые роли

Таблица 14.1 *(продолжение)*

Представление	Синоним	Описание
ROLE_TAB_PRIVS		Полномочия на объект, предоставляемые роли
SESSION_PRIVS		Полномочия, предоставленные сеансу
SESSION_ROLES		Информация о доступных для сеанса ролях
SYS_PRIVS(d,u)		Предоставленные пользователю системные полномочия
TAB_PRIVS(a,d,u)		Информация о заданных на объекты полномочиях
TABLE_PRIVILEGES		Информация о предоставленных на объекты полномочиях
USERS(a,d,u)		Информация о пользователях БД
USER_RESOURCE_LIMITS		Информация о лимитах ресурсов для текущего пользователя
Индексы		
INDEXES(a,d,u)	IND	Информация об индексах БД
IND_COLUMNS(a,d,u)		Информация об индексах, соответствующих индексам таблицы
Табличные пространства, кластеры, экстенты, файлы		
CLUSTERS(a,d,u)	CLU	Информация о всех кластерах базы данных
CLUSTER_HASH_EXPRESSIONS(a,d,u)		Список хеш-функций всех хешированных кластеров в БД
CLU_COLUMNS(d,u)		Отношение столбцов таблицы к ключам кластера

Таблица 14.1 *(окончание)*

Представление	Синоним	Описание
DBA_DATA_FILES		Информация о файлах данных
EXTENTS(d,u)		Информация об экстентах объектов в БД
FREE_SPACE(d,u)		Информация о свободных экстентах в табличной области БД
SEGMENTS(d,u)		Информация о сегментах БД
TABLESPACES(d,u)		Информация о табличных областях БД
Словарь		
DICTIONARY	DICT	Информация о таблицах и представлениях словаря данных
DICT_COLUMNS		Информация о столбцах словаря данных

14.2.3. Примеры использования словаря данных

Все результаты примеров этого раздела показаны для текущего пользователя (схемы) "COOK".

Пример 14.1. Получить перечень всех объектов текущей схемы, используя для этого представление USER_OBJECTS словаря данных. Для этого создадим запрос

```
SELECT object_name, object_type,created FROM user_objects;
```

включив в него следующий набор столбцов: имя объекта, тип объекта и дата его создания. Результат запроса имеет вид:

```
OBJECT_NAME                    OBJECT_TYPE         CREATED
------------------------------ ------------------- -----------
БЛЮДА                          TABLE               20.09.2007
Блюдо не уникально !           INDEX               20.09.2007
ВИДЫ_БЛЮД                      TABLE               20.09.2007
ВЫБОР                          TABLE               04.10.2007
```

```
Вид блюда не уникален !       INDEX              20.09.2007
Выбор не уникален !           INDEX              04.10.2007
Данные строки не уникальны ! INDEX               04.10.2007
МЕНЮ                          TABLE              04.10.2007
НАЛИЧИЕ                       VIEW               06.11.2007
ПОСТАВКИ                      TABLE              20.09.2007
ПОСТАВЩИКИ                    TABLE              20.09.2007
ПРОДУКТЫ                      TABLE              20.09.2007
Повтор кода блюда !           INDEX              20.09.2007
Повтор кода вида блюда !      INDEX              20.09.2007
Повтор кода поставщика !      INDEX              20.09.2007
Повтор кода продукта !        INDEX              20.09.2007
Повтор кода рецепта !         INDEX              08.10.2007
Повтор номера трапезы !       INDEX              20.09.2007
Поставка не уникальна !       INDEX              20.09.2007
Поставщик не уникален !       INDEX              20.09.2007
Продукт не уникален !         INDEX              20.09.2007
РЕЦЕПТЫ                       TABLE              08.10.2007
СОСТАВ                        TABLE              20.09.2007
Состав не уникален !          INDEX              20.09.2007
Строка меню не уникальна !    INDEX              04.10.2007
ТРАПЕЗЫ                       TABLE              20.09.2007
Трапеза не уникальна !        INDEX              20.09.2007
```

Пример 14.2. Получить перечень таблиц текущей схемы с указанием количества столбцов в каждой из них.

Рассмотрим, не подойдет ли для этой цели представление `USER_TAB_COLUMNS`. Если, находясь в схеме `COOK`, воспользоваться запросом

```
SELECT * FROM USER_TAB_COLUMNS;
```

то получим строки, приведенные далее (здесь сохранены данные только из пяти столбцов, которые, с нашей точки зрения, могут подойти для решения поставленной задачи и задачи из следующего примера).

```
TABLE_NAME COLUMN_NAME    DATA_TYPE ... DATA_LENGTH DATA_PRECISION ...
---------- -------------- --------- ... ----------- -------------- ...
БЛЮДА      КОД_БЛЮДА      NUMBER                 22              2
БЛЮДА      БЛЮДО          VARCHAR2               16
БЛЮДА      КОД_ВИДА       NUMBER                 22              1
БЛЮДА      ОСНОВА         VARCHAR2                6
```

БЛЮДА	ВЫХОД	NUMBER	22	4
БЛЮДА	ТРУД	NUMBER	22	3
ВИДЫ_БЛЮД	КОД_ВИДА	NUMBER	22	1
ВИДЫ_БЛЮД	ВИД	VARCHAR2	7	
ВЫБОР	МЕСТО	NUMBER	22	2
ВЫБОР	СТРОКА	NUMBER	22	9
МЕНЮ	СТРОКА	NUMBER	22	9
МЕНЮ	КОД_ТРАПЕЗЫ	NUMBER	22	1
МЕНЮ	КОД_БЛЮДА	NUMBER	22	2
МЕНЮ	ДАТА	DATE	7	
НАЛИЧИЕ	КОД_ПРОДУКТА	NUMBER	22	2
НАЛИЧИЕ	К_ВО	NUMBER	22	
НАЛИЧИЕ	СТОИМОСТЬ	NUMBER	22	
ПОСТАВКИ	КОД_ПОСТАВЩИКА	NUMBER	22	2
ПОСТАВКИ	КОД_ПРОДУКТА	NUMBER	22	2
ПОСТАВКИ	ЦЕНА	NUMBER	22	5
ПОСТАВКИ	К_ВО	NUMBER	22	4
ПОСТАВКИ	ДАТА	DATE	7	
ПОСТАВЩИКИ	КОД_ПОСТАВЩИКА	NUMBER	22	2
ПОСТАВЩИКИ	НАЗВАНИЕ	VARCHAR2	8	
ПОСТАВЩИКИ	СТАТУС	VARCHAR2	10	
ПОСТАВЩИКИ	ГОРОД	VARCHAR2	9	
ПОСТАВЩИКИ	АДРЕС	VARCHAR2	14	
ПОСТАВЩИКИ	ТЕЛЕФОН	VARCHAR2	7	
ПРОДУКТЫ	КОД_ПРОДУКТА	NUMBER	22	2
ПРОДУКТЫ	ПРОДУКТ	VARCHAR2	8	
ПРОДУКТЫ	БЕЛКИ	NUMBER	22	4
ПРОДУКТЫ	ЖИРЫ	NUMBER	22	4
ПРОДУКТЫ	УГЛЕВ	NUMBER	22	4
ПРОДУКТЫ	К	NUMBER	22	4
ПРОДУКТЫ	СА	NUMBER	22	4
ПРОДУКТЫ	NA	NUMBER	22	4
ПРОДУКТЫ	B2	NUMBER	22	4
ПРОДУКТЫ	PP	NUMBER	22	4
ПРОДУКТЫ	C	NUMBER	22	4
РЕЦЕПТЫ	ИД	NUMBER	22	9
РЕЦЕПТЫ	КОД_БЛЮДА	NUMBER	22	2
РЕЦЕПТЫ	РЕЦЕПТ	VARCHAR2	560	
РЕЦЕПТЫ	ВАРИАНТ	NUMBER	22	2

```
СОСТАВ      КОД_БЛЮДА       NUMBER          22          2
СОСТАВ      КОД_ПРОДУКТА    NUMBER          22          2
СОСТАВ      ВЕС             NUMBER          22          3
ТРАПЕЗЫ     КОД_ТРАПЕЗЫ     NUMBER          22          1
ТРАПЕЗЫ     ТРАПЕЗА         VARCHAR2         7

48 rows selected
```

Легко заметить, что для решения поставленной задачи достаточно данных из двух первых столбцов. Однако из этих данных надо удалить три строки, описывающих представление (VIEW) НАЛИЧИЕ (см. объекты примера 14.1). Это можно сделать при помощи запроса:

```
SELECT table_name, COUNT(column_name) Столбцов
  FROM USER_TAB_COLUMNS
  WHERE table_name NOT IN (SELECT view_name FROM USER_VIEWS)
  GROUP BY table_name;
```

в котором исключаются строки, описывающие представления, и производится группировка по именам таблиц с получением количества столбцов в них:

```
TABLE_NAME       СТОЛБЦОВ
-------------- ----------
БЛЮДА                 6
ВИДЫ_БЛЮД             2
ВЫБОР                 2
МЕНЮ                  4
ПОСТАВКИ              5
ПОСТАВЩИКИ            6
ПРОДУКТЫ             11
РЕЦЕПТЫ               4
СОСТАВ                3
ТРАПЕЗЫ               2
10 rows selected
```

Пример 14.3. Получить перечень столбцов всех таблиц текущей схемы с указанием комментариев и типов хранимых в них данных. Для этого воспользуемся представлениями словаря данных USER_TAB_COLUMNS, USER_OBJECTS и USER_COL_COMMENTS:

```
SELECT t.table_name,t.column_name,data_type||
       CASE
            WHEN data_type = 'VARCHAR2'  THEN ' ('||data_length||')'
            WHEN data_type = 'NUMBER'    THEN ' ('||data_precision||')'
            ELSE NULL
       END ТИП_ДАННЫХ, c.comments
```

```
FROM USER_TAB_COLUMNS t, user_objects o, USER_COL_COMMENTS c
WHERE t.TABLE_NAME = o.object_name
AND t.COLUMN_NAME = c.column_name
AND t.TABLE_NAME = c.table_name
AND object_type = 'TABLE';
```

TABLE_NAME	COLUMN_NAME	ТИП_ДАННЫХ	COMMENTS
БЛЮДА	КОД_БЛЮДА	NUMBER (2)	Код блюда
БЛЮДА	БЛЮДО	VARCHAR2 (16)	Название блюда
БЛЮДА	КОД_ВИДА	NUMBER (1)	Код вида блюда
БЛЮДА	ОСНОВА	VARCHAR2 (6)	Основной продукт в блюде
БЛЮДА	ВЫХОД	NUMBER (4)	Масса порции готового блюда
БЛЮДА	ТРУД	NUMBER (3)	Стоимость приготовления блюда (коп)
ВИДЫ_БЛЮД	КОД_ВИДА	NUMBER (1)	Код вида блюда
ВИДЫ_БЛЮД	ВИД	VARCHAR2 (7)	Вид блюда (Закуска, Суп, ...)
ВЫБОР	МЕСТО	NUMBER (2)	Номер места в столовой пансионата
ВЫБОР	СТРОКА	NUMBER (9)	Строка меню
МЕНЮ	СТРОКА	NUMBER (9)	Номер строки в меню
МЕНЮ	КОД_ТРАПЕЗЫ	NUMBER (1)	Номер трапезы
МЕНЮ	КОД_БЛЮДА	NUMBER (2)	Код блюда
МЕНЮ	ДАТА	DATE	Меню составлено на дату
ПОСТАВКИ	КОД_ПОСТАВЩИКА	NUMBER (2)	Код поставщика
ПОСТАВКИ	КОД_ПРОДУКТА	NUMBER (2)	Код продукта
ПОСТАВКИ	ЦЕНА	NUMBER (5)	Договорная цена продукта (руб)
ПОСТАВКИ	К_ВО	NUMBER (4)	Наличие продукта у поставщика (кг)
ПОСТАВКИ	ДАТА	DATE	Дата поставки продукта
ПОСТАВЩИКИ	КОД_ПОСТАВЩИКА	NUMBER (2)	Код поставщика
ПОСТАВЩИКИ	НАЗВАНИЕ	VARCHAR2 (8)	Название поставщика
ПОСТАВЩИКИ	СТАТУС	VARCHAR2 (10)	(Кооператив, совхоз, база, ...)
ПОСТАВЩИКИ	ГОРОД	VARCHAR2 (9)	Город, где расположена контора
ПОСТАВЩИКИ	АДРЕС	VARCHAR2 (14)	Улица и номер дома конторы
ПОСТАВЩИКИ	ТЕЛЕФОН	VARCHAR2 (7)	Телефон конторы
ПРОДУКТЫ	КОД_ПРОДУКТА	NUMBER (2)	Код продукта
ПРОДУКТЫ	ПРОДУКТ	VARCHAR2 (8)	Название продукта
ПРОДУКТЫ	БЕЛКИ	NUMBER (4)	Белков в 1000 г продукта (г)
ПРОДУКТЫ	ЖИРЫ	NUMBER (4)	Жиров в 1000 г продукта (г)
ПРОДУКТЫ	УГЛЕВ	NUMBER (4)	Углеводов в 1000 г продукта (г)
ПРОДУКТЫ	К	NUMBER (4)	Калия в 1000 г продукта (мг)
ПРОДУКТЫ	CA	NUMBER (4)	Кальция в 1000 г продукта (мг)
ПРОДУКТЫ	NA	NUMBER (4)	Натрия в 1000 г продукта (мг)

ПРОДУКТЫ	B2	NUMBER (4)	Рибофлавина в 1000 г продукта (мг)
ПРОДУКТЫ	PP	NUMBER (4)	Ниацина в 1000 г продукта (мг)
ПРОДУКТЫ	C	NUMBER (4)	Аскорбинки в 1000 г продукта (мг)
РЕЦЕПТЫ	ИД	NUMBER (9)	Идентификатор рецепта
РЕЦЕПТЫ	КОД_БЛЮДА	NUMBER (2)	Код блюда
РЕЦЕПТЫ	РЕЦЕПТ	VARCHAR2 (560)	Технология приготовления блюда
РЕЦЕПТЫ	ВАРИАНТ	NUMBER (2)	Номер варианта рецепта
СОСТАВ	КОД_БЛЮДА	NUMBER (2)	Код блюда
СОСТАВ	КОД_ПРОДУКТА	NUMBER (2)	Код продукта
СОСТАВ	ВЕС	NUMBER (3)	Масса продукта в блюде
ТРАПЕЗЫ	КОД_ТРАПЕЗЫ	NUMBER (1)	Номер трапезы
ТРАПЕЗЫ	ТРАПЕЗА	VARCHAR2 (7)	Название трапезы

Глава 15

Оптимизация SQL-запросов

15.1. Введение

Как сделать запросы SQL более эффективными, прежде всего с точки зрения времени их выполнения?

Оптимизация SQL — это огромная и очень важная тема, специфичная для каждой СУБД. Мы рассмотрим общие вопросы оптимизации и укажем, на что надо обратить внимание.

Каждая СУБД имеет в своем составе оптимизатор запросов — наиболее хитроумный, сложный и интересный компонент. Сложность оптимизации SQL-запросов состоит в том, что язык SQL декларативен. В формулировках SQL-запросов указывается, какими свойствами должны обладать данные, которые желательно получить, но ничего не говорится о том, как система должна реально выполнить запрос. Проблема в том, чтобы по декларативной формулировке запроса найти (или построить) программу, которую принято называть *планом выполнения запроса*. Более точно, основная трудность состоит в том, что нужно:

- уметь построить все возможные программы, результаты которых соответствуют указанным свойствам;
- выбрать из множества этих программ такую программу, выполнение которой было бы наиболее эффективным.

Обе части проблемы нетривиальны. Прежде всего, необходимо обнаружить все корректные планы выполнения запроса или, по крайней мере, не упустить какой-либо план, который является наиболее эффективным. Далее, для облегчения решения второй части проблемы требуется предельно сократить пространство корректных планов, оставив только те планы, которые претендуют на максимальную эффективность. Обе эти задачи являются не полностью формализуемыми, поскольку отсутствуют точные математические критерии

выбора. Обычно решение задач опирается на эвристические алгоритмы; обсудим некоторые из них.

Предположим теперь, что первая часть проблемы каким-то образом решена. Теперь требуется решить вторую — и более ответственную — часть проблемы: найти в пространстве планов выполнения запроса единственный план, в соответствии с которым запрос будет реально выполнен (нередко эту часть проблемы называют проблемой физической оптимизации). Здесь уже требуются формальные критерии отбора. Таким критерием может быть оценочная стоимость выполнения запроса по данному плану. Основным компонентом оцениваемой стоимости может быть число обменов с устройствами внешней памяти, которые потребуются при выполнении плана запроса. Именно этот подход продолжает использоваться в подавляющем большинстве SQL-ориентированных СУБД.

Технически не очень трудно обеспечить полный набор планов выполнения для любой заданной формулировки SQL-запроса. Но ситуация существенно усложняется тем, что для любого нетривиального SQL-запроса существует несколько (и даже много) семантически эквивалентных формулировок. Если не учитывать альтернативные формулировки заданного запроса, можно упустить эффективные планы выполнения. Если учитывать все возможные формулировки, пространство корректных планов может оказаться слишком большим, чтобы можно было эффективно решить вторую часть проблемы оптимизации. Эти соображения привели к возникновению направления, которое принято называть *логической оптимизацией запросов*.

Мы не будем углубляться в детали этой проблемы. Отметим только, что СУБД позволяет познакомиться с полученным ею планом доступа и вносить в него изменения, если вы знаете лучший способ его выполнения. Однако ни один оптимизатор не сможет найти хорошего плана, если вы не будете знать и по возможности использовать важнейшие способы увеличения производительности системы. (Для того чтобы выиграть в лотерею, необходимо иметь хотя бы один лотерейный билет!)

- Использование индексов. Индексы — это объекты баз данных, основная цель которых состоит в повышении производительности запросов (см. *разд. 15.2*).
- Задание выражений SQL с учетом производительности. Многие детали отличаются в разных реализациях. В *разд. 15.4* вводятся общие принципы и предлагаются некоторые вариации, повышающие производительность.
- Настройка физических параметров СУБД. Под физическими параметрами понимаются такие, как способ разделения пространства хранения данных, размер кеша, выделенного под операторы (при условии, что ваша система

поддерживает кеширование операторов). Все эти вопросы зависят исключительно от конкретной реализации, поэтому они не освещаются в этой главе. Однако мы упоминаем о них, так как, возможно, будет полезно справиться о них в документации продукта.

- Минимизация уровня локализации блокировки или использование оптимистической блокировки (см. *разд. 9.3.2*).

15.2. Использование индексов

15.2.1. Что такое индексы

Основным способом повышения производительности операций SQL, не зависящим от платформы, является использование индексов. Индекс SQL похож на полный алфавитный указатель в книге, т. е. на список всех терминов (обычно за исключением тривиальных), упоминающихся в книге со ссылками на все места, где они встречаются. *Индекс SQL* — это список всех значений в группе из одного или нескольких столбцов, упорядоченный в некотором приемлемом для данного типа данных смысле (например, в порядке возрастания для чисел или в алфавитном порядке для символьных строк). Каждое значение имеет указатель на строку в таблице, где это значение встречается.

Стандарты, как правило, не касаются вопросов производительности. Поэтому синтаксис индексов и их функциональность различны для конкретных продуктов. Однако имеются некоторые общие черты.

Индексы работают неявно. Они в основном используются в запросах. При выполнении запроса СУБД решает, какие индексы надо применить и надо ли вообще. Если вы создаете индексы, то следовательно, определяете возможность их использования. Можно также использовать подсказки, называемые *директивами* (directives), которые приказывают оптимизатору изменить способ выполнения, включая использование или неиспользование конкретных индексов при условии, что это приведет к хорошим результатам.

Индексы можно классифицировать несколькими способами, но самая фундаментальная классификация — это *уникальные* (unique) и *неуникальные* (non-unique) индексы. В уникальном индексе может быть только одна ссылка для каждого значения, а в неуникальном индексе — произвольное число. Уникальный индекс похож на ограничение целостности `UNIQUE`, которое запрещает повторение значения в столбце или в группе столбцов.

15.2.2. Создание индексов

Индекс в качестве объекта схемы создается так же, как и другие объекты — с помощью предложения CREATE:

```
CREATE [UNIQUE] INDEX имя_индекса ON имя_таблицы {имя_столбца [,...]);
```

Ключевые слова:

```
CREATE [UNIQUE] INDEX имя_индекса
```

Создается новый индекс с именем *имя_индекса* в контексте текущей базы данных и схемы. Поскольку индексы связаны с конкретными таблицами, *имя_индекса* должно быть уникальным только в пределах таблицы, от которой индекс зависит. Ключевое слово UNIQUE определяет индекс как ограничение типа UNIQUE в данной таблице. Оно запрещает повторяющиеся значения в проиндексированном столбце или столбцах таблицы.

```
ON имя_таблицы
```

Объявляет, с какой таблицей связывается индекс. Индекс зависит от таблицы. Если таблица удаляется, то же происходит и с индексом.

```
(имя_столбца [,...])
```

Определяется один или несколько столбцов таблицы, которые будут индексироваться. Наличие проиндексированного столбца или столбцов позволяют оптимизатору запросов базы данных значительно увеличить скорость манипуляций с данными, например инструкций SELECT или DELETE. Все крупные производители СУБД поддерживают *составные индексы* (composite index), иначе называемые *сцепленными индексами* (concatenated index). Эти индексы используются в тех случаях, когда лучше всего проводить поиск по двум столбцам как по единому целому, например по фамилии и имени.

Общие правила. Индексы создаются по таблицам для ускорения операций манипулирования данными, которые проводятся с таблицами, например предложений WHERE и JOIN. Индексы способны ускорить и другие операции, например:

- определение значений MIN() или MAX() по индексированному столбцу;
- сортировку и группировку столбцов таблицы;
- поиск типа IS NULL или IS NOT NULL;
- извлечение данных, когда нужны только проиндексированные данные.

После создания таблицы вы можете создавать индексы по столбцам таблицы. Хорошей практикой является создание индексов по столбцам, которые часто указываются во фразах WHERE и JOIN.

15.2.3. Необходимость использования индексов

Использование индексов имеет некоторые недостатки. На них расходуются ресурсы.

- При всяком изменении или удалении содержимого индексированного столбца, а также при добавлении новой строки индекс необходимо обновлять. Это замедляет операции обновления. Однако обычно индексы не замедляют обновление на столько же, на сколько они ускоряют запросы.
- Индекс сам по себе занимает место — вдобавок к тому, что занимает таблица. Индекс не обязательно должен храниться вместе с таблицей, если только он не в той же схеме.
- При извлечении из таблицы очень большого числа строк использование индекса — только потеря времени. Это несколько замедляет исполнение.

Решите, стоит ли индекс того места, которое он занимает, и превысит ли выигрыш во времени исполнения запросов замедление обновлений. Чтобы лучше понять ситуацию, рассмотрите следующие вопросы:

1. Как часто будет выполняться запрос к таблице и как часто она будет обновляться?
2. Насколько вероятно то, что обычные запросы и обновления будут выполняться слишком медленно?
3. Как часто столбцы, которые вы собираетесь индексировать, будут использоваться в предикатах, особенно в предикатах запросов?
4. Как часто столбцы, которые вы собираетесь индексировать, будут использоваться в предикатах *соединения* (join)? Соединения в основном выполняются медленнее, чем другие виды запросов, и выигрыш во времени получается больше.
5. Что для вас более важно: время исполнения или место на диске? Индексы жертвуют одним ради другого. Обычно время более ценно, так как индексы радикально уменьшают время исполнения, а места занимают не так уж много.
6. Имеются ли предпосылки для создания специальных видов индексов, таких как индексы массивов битов или кластеры? Эти виды индексов и области их применения подробно обсуждаются в [1 и 3].

Наилучший способ решить, нужны ли индексы — это проанализировать код приложения и понять, чего оно требует. При использовании интерактивного или динамического SQL нужно выбрать наилучшие из возможных предположений. Однако полезной оказывается возможность создания и удаления индексов по ходу дела, так что можно постепенно настраивать производительность приложения желаемым способом.

ЧАСТЬ VI

СОЗДАНИЕ ПРИЛОЖЕНИЙ НА SQL

Глава 16

Программирование на SQL

16.1. Введение

Язык SQL можно использовать для доступа к базам данных в двух *режимах*. При интерактивной работе и в прикладных программах. По большей части сам язык одинаков в обоих вариантах. Эта двойственность SQL имеет несколько преимуществ:

- прикладные программисты могут относительно легко научиться писать программы, которые в ходе своей работы обращаются к базам данных;
- все возможности, доступные в интерактивном языке запросов, автоматически доступны и в прикладных программах;
- предложения SQL, предназначенные для использования в программах, вначале могут быть проверены в интерактивном режиме, а затем вставлены в исходный текст программы;
- программы могут работать с базами данных на уровне таблиц и результатов запросов.

Так как непроцедурный язык SQL первоначально создавался исключительно для осуществления доступа к базам данных, то в нем отсутствовали даже самые элементарные возможности процедурных языков программирования [2]. В SQL нельзя объявлять переменные, в нем отсутствуют инструкция перехода `GOTO`, инструкция `IF` для проверки условий, инструкции `FOR`, `DO` и `WHILE` для организации циклов и т. д. Чтобы создать полноценную программу, предназначенную для работы с базой данных, необходимо либо написать ее:

- на обычном языке программировании, таком как COBOL, PL/1, PASCAL, C, JAVA или PHP, и по мере надобности "встраивать" в нее инструкции SQL;
- на каком-либо из расширений языка SQL, разработанных различными производителями СУБД, и, к сожалению, пока не стандартизированных.

Рассмотрим существующие возможности создания прикладных программ, использующих язык SQL:

- встраивание в код некоторого языка программирования SQL-предложений (*статический SQL*);
- формирование в процессе выполнения программы на некотором языке программирования кода SQL-предложений и дальнейшего их выполнения (*динамический SQL*);
- использование *интерфейса программирования приложений* (Application Programming Interface, API), позволяющего реализовывать работу с базой данных через предоставляемый набор функций. API может быть целевым, предоставленным производителем коммерческой СУБД для работы именно с этой базой данных, или межплатформенным, реализующим унифицированные средства доступа к СУБД различных производителей (например, ODBC — Open DataBase Connectivity);
- использование *процедурных расширений SQL*, вводимых производителями СУБД. Практически в каждой СУБД применяется свой процедурный язык: SQLpl в DB2, PL/SQL в Oracle, PL/pgSQL в PostgreSQL, Trunsact-SQL в SQL Server и т. п. Эти диалекты включают кроме SQL средства условной обработки (например, `IF…THEN`), управляющие операторы (например, циклы `WHILE`), переменные, средства обработки ошибок, позволяющие создавать хранимые процедуры, функции и триггеры, использующиеся большим количеством приложений, повышая эффективность их функционирования, обеспечивая высокую степень защиты последних и унифицируя способы обращения к данным из приложений.

16.2. Статический SQL

При таком подходе к созданию прикладных программ, предложения SQL встраиваются непосредственно в исходный текст программы, создаваемой на "полноценном" языке программирования, который поддерживает SQL. Для пересылки информации из базы данных в программу используются специальные предложения встроенного SQL (рис. 16.1). Исходный текст программы, включающий в себя предложения встроенного SQL, перед компиляцией подается на вход специального препроцессора SQL, который с помощью ряда других программных модулей преобразует этот исходный текст в исполняемую программу.

Везде, где во встроенных предложениях SQL могут стоять константы, вместо них можно использовать переменные из базовой прикладной программы.

Эти переменные называются базовыми переменными. С помощью этих, входных, переменных в базу данных можно передавать значения, вводимые пользователем.

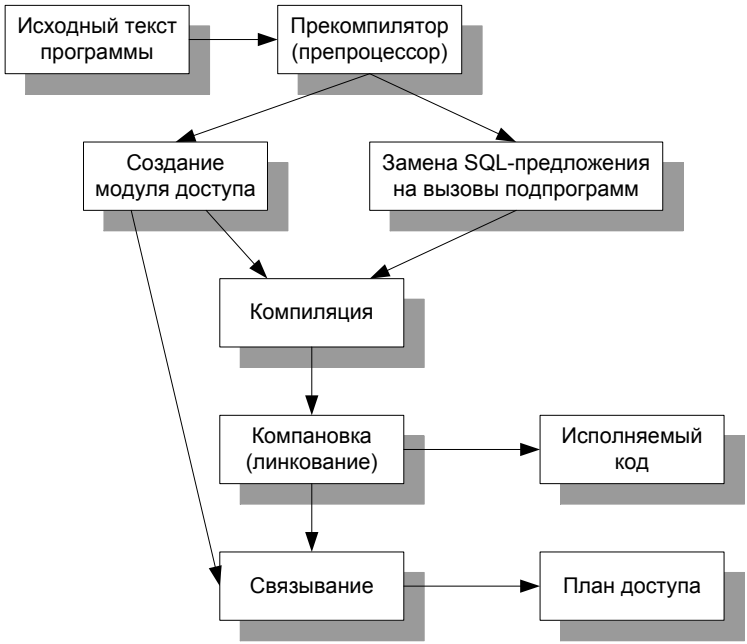


Рис. 16.1. Процесс выполнения программы, содержащей предложения встроенного SQL

Базовые переменные применяются также и для получения результатов запросов. Значения этих, выходных, переменных могут затем обрабатываться прикладной программой.

16.3. Динамический SQL

Статический SQL пригоден для написании обычных программ обработки данных, в которых заранее определена и жестко зафиксирована схема доступа к базе данных. В каждом встроенном предложении SQL программист заранее указывает, на какие таблицы и столбцы он будет ссылаться. Входные базовые переменные придают статическому SQL некоторую гибкость, но не могут коренным образом изменить его статическую природу.

Однако существует достаточно большой класс приложений, в которых невозможно заранее определить схему доступа к базе данных. Например, программа создания запросов или программа, генерирующая отчеты, должна иметь возможность во время выполнения решать, какие предложения SQL она будет использовать для доступа к базе данных. Программа для работы с электронными таблицами, установленная на персональном компьютере и имеющая доступ к серверной базе данных, также должна иметь возможность сформировать запрос к этой базе данных "на ходу". Перечисленные программы, а также другие клиентские приложения общего назначения невозможно написать, используя инструкции статического SQL. Для создания этих программ необходима усовершенствованная разновидность встроенного SQL, которая называется *динамический SQL*.

Общая концепция, лежащая в основе динамического SQL, проста — встроенная инструкция SQL не записывается в исходный текст программы. Вместо этого программа формирует текст инструкции во время выполнения в одной из своих областей данных, а затем передает сформированную инструкцию в СУБД для динамического выполнения. Хотя детали реализации являются довольно сложными, весь динамический SQL построен на этом простом принципе, о котором не следует забывать.

Динамический SQL менее эффективен (в смысле производительности), чем статический SQL. По этой причине всегда, когда это возможно, используется статический SQL.

Важность динамического SQL возросла с появлением трехуровневых Internet-архитектур, в которых управляющее программное обеспечение расположено на одной системе (прикладной, или промежуточный, уровень), а СУБД — на другой (информационный, или серверный, уровень). В большинстве таких систем программная логика довольно непостоянна, динамична по своей природе. Она должна адаптироваться к меняющимся условиям бизнеса, к появлению новых деловых правил. Регулярно изменяющаяся программная среда плохо сочетается со статическим SQL, в котором между программой и содержимым базы данных существует жесткая связь.

Статический и динамический SQL представляют собой классический пример компромисса между эффективностью и гибкостью, что выражается в следующем.

- *Простота.* Статический SQL относительно прост; даже самый сложный его элемент — наборы записей — можно легко освоить, вспомнив концепцию файлового ввода/вывода. Динамический SQL довольно сложен, в нем осуществляется динамическое формирование предложений, используются структуры данных переменной длины и выполняется распределение памяти.

- *Производительность.* Во время компиляции программы, использующей статический SQL, создастся план выполнения всех встроенных инструкций; инструкции динамического SQL компилируются непосредственно на этапе выполнения. В результате производительность статического SQL, как правило, намного выше, чем динамического.
- *Гибкость.* Динамический SQL дает программе возможность решать на этапе выполнения, какие конкретно инструкции SQL она будет выполнять. Статический SQL требует, чтобы все инструкции SQL были написаны заранее, на этапе создания программы; тем самым он ограничивает гибкость программы.

Пример использования динамического SQL будет рассмотрен в *главе 18*.

16.4. Интерфейс программирования приложений

Наконец, еще об одном способе создания приложений к базам данных — это интерфейс прикладного программирования (API) баз данных, который используется для передачи предложений SQL в СУБД и для получения результатов их обработки из СУБД. Упомянем здесь только три распространенных API, которые предоставляют единый интерфейс для работы с разными платформами баз данных.

- *ODBC (Open DataBase Connectivity)* — это программный интерфейс доступа к базам данных, разработанный фирмой X/Open. Позволяет единообразно оперировать с разными источниками данных, отвлекаясь от особенностей взаимодействия в каждом конкретном случае. Поставщики различных СУБД создают драйверы, реализующие конкретное наполнение стандартных функций из ODBC API с учетом особенностей их продукта. Приложения используют эти функции, реализованные в драйверах, для унифицированного доступа к различным источникам данных, используя соответствующие источникам данных драйверы.
- *ADO.NET (ActiveX Data Objects)* — это высокоуровневый интерфейс прикладного программирования баз данных от компании Microsoft, работающий на платформе .NET. Он представляет собой коллекцию интерфейсов .NET, доступ к которым осуществляется с помощью любого языка с поддержкой .NET. Главное преимущество ADO.NET — это простота исполь-

зования, переносимость в пределах платформы .NET, интеграция с XML и доступ к источникам данных, отличным от реляционных баз данных.

- *JDBC (Java Database Connectivity)* разработан компанией Sun Microsystems в первую очередь как API баз данных для языка Java, JDBC является наиболее популярным интерфейсом прикладного программирования баз данных на языке Java [3]. Он предоставляет возможность перенесения кода с одной операционной системы на другую, предлагает приемлемую производительность для большинства областей применения, и он достаточно хорошо документирован. Кроме того, драйверы для большинства баз данных распространяются, как правило, бесплатно.

Глава 17

Процедурные расширения SQL

17.1. Введение

Так как описание всех процедурных расширений языка SQL (SQLpl в DB2, PL/SQL в Oracle, PL/pgSQL в PostgreSQL, Trunsact-SQL в SQL Server и др.) не входило в задачу данной книги, то познакомим читателя с одним из них — PL/SQL. Попытаемся дать здесь краткое его описание, достаточное, с нашей точки зрения, для создания простых приложений на этом языке. Для получения дополнительных сведений о PL/SQL предлагаем обратиться к книге Скотта Урмана [10] и материалам *приложения А*. В *приложении А* помещена электронная версия книги Урмана (Oracle 9i. Программирование на языке PL/SQL. Скотт Урман. — М.: Издательство "ЛОРИ", 2004, 548 с.) и документация к Oracle Database 10g Release 2, в которой есть раздел: "PL/SQL User's Guide and Reference".

17.2. Основы PL/SQL

PL/SQL (Procedural Language / Structured Query Language) — процедурный язык программирования, разработанный в корпорации Oracle. Является процедурным расширением языка SQL. Базируется на языке Ада.

PL/SQL — это развитый язык программирования, используемый для доступа к базам данных Oracle из различных сред. PL/SQL интегрирован с сервером базы данных, поэтому программы PL/SQL обрабатываются быстро и эффективно. Этот язык доступен и в некоторых клиентских инструментальных средствах Oracle. Он дает возможность использовать переменные, операторы, массивы, курсоры и исключения.

Стандартный SQL является декларативным языком программирования. Это накладывает на язык определенные ограничения, такие как, например,

отсутствие прямой поддержки циклов. PL/SQL же, как полноценный процедурный язык программирования, позволяет разработчикам обрабатывать реляционную базу данных Oracle, используя (более привычный) императивный стиль программирования. Операторы SQL могут быть легко вызваны непосредственно из кода PL/SQL-процедуры, функции или триггера.

PL/SQL — блочно-структурированный язык. Это означает, что основные единицы программ PL/SQL (анонимные блоки, процедуры и функции) являются логическими блоками, которые могут содержать любое число вложенных в них блоков. Структура блока имеет следующий вид:

```
DECLARE
/* Раздел объявлений — здесь перечисляются переменные, типы, курсоры и
локальные подпрограммы PL/SQL. */
BEGIN
/* Выполняемый раздел — процедурные и SQL-операторы. Это
основной раздел блока и единственный обязательный. */
EXCEPTION
/* Раздел обработки исключительных ситуаций — здесь находятся операторы
обработки ошибок. */
END;
```

Обязательным является только выполняемый раздел (`BEGIN`). Он должен содержать по меньшей мере один выполняемый оператор (например, `NULL;`).

17.2.1. Анонимный блок PL/SQL

В качестве примера структуры блока PL/SQL, рассмотрим анонимный блок (неименованную программу), в котором выполняется проверка правильности ввода (изменения) фамилии, имени или отчества и, если необходимо, их корректировка. Фамилия (имя) должна состоять только из русских букв и символов тире и пробел. Первая буква фамилии (имени или отчества) должна быть заглавной, а остальные — строчными. В двойных (тройных) фамилиях и именах каждая часть должна начинаться с заглавной буквы (например, Смирнов-Сокольский, Жан-Жак Руссо, Гай Юлий Цезарь).

```
DECLARE
/* декларативная часть анонимного блока описание локальных переменных */
  tekst  VARCHAR2(20) := 'Жан - жак  руссо'; /* Проверяемый текст
                                      (фамилия, имя или отчество)*/
  result VARCHAR2(20); /* Результат (проверенный или исправленный входной
                          текст или "0" при неисправимой ошибке)*/
  kol    INTEGER; /* Количество символов в проверяемом тексте */
```

```
BEGIN /* исполняемая часть (тело) анонимного блока*/
  result := LOWER(RTRIM(LTRIM(tekst)));
     /* первый вариант результата, полученный после удаления из входного
        текста начальных (LTRIM) и конечных (RTRIM) пробелов, а также
        преобразования всех букв текста в строчные (LOWER) */
  kol := LENGTH(result);
     /* определение числа символов, оставшихся в тексте после удаления
        начальных и конечных пробелов */
  IF kol > 0 THEN
     /* если текст содержит символы, то его дальнейшее преобразование и
        проверка на наличие запрещенных символов */
    result := REPLACE(result,'  ',' '); -- замена двух пробелов на один
    result := REPLACE(result,'- ','-'); -- замена тире и пробела на тире
    result := REPLACE(result,' -','-'); -- замена пробела и тире на тире
    kol := LENGTH(result); -- определение числа оставшихся символов
    FOR i IN 1..kol LOOP /* перебор всех символов текста */
     IF INSTR('- абвгдеёжзийклмнопрстуфхцчшщъыьэюя',SUBSTR(result,i,1))=0
        THEN
          /* с помощью функции INSTR определяется позиция первого
             включения i-го символа текста (вырезанного с помощью функции
             SUBSTR) в набор '-абвгдеёжзийклмнопрстуфхцчшщъыьэюя', и если
             эта позиция равна 0 (i-го символа нет в наборе), то
             производится: */
        result := '0';  -- установка нулевого результата и
        EXIT; -- выход из цикла
     END IF;
    END LOOP;
  ELSE
    result := '0';  -- установка нулевого результата при отсутствии
                    -- в тексте символов, отличных от пробелов
  END IF;
  IF result <> '0' THEN -- если текст содержит символы, то:
    result := INITCAP(result);
              /* преобразование первых букв слов текста в заглавные */
  END IF;
  DBMS_OUTPUT.PUT_LINE(result);
            /* использование встроенного пакета DBMS_OUTPUT для вывода на
               экран результата преобразования; если ранее не выполнялась
```

```
                установка SET SERVEROUTPUT ON, разрешающая вывод на экран
                информации, заданной в DBMS_OUTPUT, то ее надо выполнить
                до выполнения этого блока */
END;
/       -- наклонная черта, указывающая на окончание текста блока PL/SQL
```

Этот анонимный блок можно ввести с помощью текстового редактора в файл (например, с именем an_fio.sql) или вводить строчка за строчкой прямо в SQL*Plus. В последнем случае ввод в первой строке одного из зарезервированных слов `DECLARE` или `BEGIN` переводит SQL*Plus в режим построчного ввода строк анонимного блока.

После ввода наклонной черты блок выполняется и на экран выдается результат его работы:

```
     Жан-Жак Руссо
     Процедура PL/SQL успешно завершена.
SQL> _
```

Если текст анонимного блока был сохранен в файле an_fio.sql, то такой сценарий SQL можно выполнить с помощью команды `@an_fio.sql` или `@an_fio` (расширение sql, указывающее, что данный файл является файлом запроса, можно опустить):

```
SQL> @an_fio
     Жан-Жак Руссо
     Процедура PL/SQL успешно завершена.
SQL> _
```

Синтаксис команд PL/SQL и выражений практически не отличается от тех, которые описаны в *главе 4*.

17.3. Переменные, константы, записи PL/SQL

В программах PL/SQL могут использоваться *переменные* и *константы*, описываемые в разделе `DECLARE` с помощью конструкции вида:

Имя_переменной [CONSTANT] *тип_данных* [NOT NULL] [{ := | DEFAULT } *выражение*]

Если в объявлении переменной указано `CONSTANT`, то она должна быть инициализирована, и ее начальное значение не может быть изменено. Константная переменная рассматривается в блоке в качестве переменной "только для

чтения". Константы часто используются для хранения тех значений, которые известны к моменту создания блока. Например:

```
rojdenie    DATE;
kol_vo      NUMBER(9) := 0;
priznak     VARCHAR2(6) NOT NULL := 'академ';
pi          CONSTANT REAL := 3.14159;
area        REAL := pi*radius**2;
valid_id    BOOLEAN;
valid_id    VARCHAR2(5);  -- недопустимое вторичное описание  valid_id
i, j, k     NUMBER(9);    -- нельзя описывать список, надо:
                          -- i NUMBER(9); j  NUMBER(9); k  NUMBER(9);
credit      NUMBER(9,2);
debit       credit%TYPE;  -- тип данных, аналогичный типу данных
                          -- "credit"
```

Записи PL/SQL — это совокупность полей, каждое из которых должно иметь уникальное имя (в пределах записи). Эти поля могут принадлежать различным типам данных.

Если создаваемая запись (`bludo`) соответствует описанию столбцов какой-либо базовой таблицы (например, `Блюда`), то ее объявление можно осуществить в разделе `DECLARE` с помощью атрибута `%ROWTYPE`:

```
bludo Блюда%ROWTYPE;
```

В противном случае для объявления записи необходимо сначала определить ее тип. Для описания типа записи используется синтаксис:

```
TYPE имя_типа_записи IS RECORD
  ( имя_столбца1 {тип_данных | имя_переменной%TYPE |
имя_таблицы.имя_столбца%TYPE | имя_таблицы%ROWTYPE} [NOT NULL],
    имя_столбца2 {тип_данных | имя_переменной%TYPE |
имя_таблицы.имя_столбца%TYPE | имя_таблицы%ROWTYPE} [NOT NULL], ...);
```

где *имя_типа_записи* — спецификатор типа, используемый в последующих объявлениях записей PL/SQL, и *тип_данных* — тип данных. С помощью атрибута `%TYPE` можно установить *тип_данных*, соответствующий типу данных какой-либо переменной (*имя_переменной*) или столбца (*имя_таблицы.имя_столбца*). Атрибут `%ROWTYPE` позволяет определить поле как запись, соответствующую описанию столбцов какой-либо базовой таблицы.

Например, если необходимо создать запись `rec_menu`, содержащую поля `трапеза`, `вид`, `блюдо` и `дата` из таблиц `меню`, `блюда`, `виды_блюд`, `трапезы`, то для нее можно определить тип:

```
TYPE rec_menu_type IS RECORD
( trapeza трапезы.трапеза%TYPE,
```

```
  vid виды_блюд.вид%TYPE,
  bludo блюда.блюдо%TYPE,
  date_menu меню.дата%TYPE );
```

Теперь можно определить запись:

```
rec_menu rec_menu_type;
```

Для ее создания можно использовать следующий анонимный блок:

```
DECLARE
TYPE rec_menu_type IS RECORD
(trapeza трапезы.трапеза%TYPE,
 vid виды_блюд.вид%TYPE,
 bludo блюда.блюдо%TYPE,
 date_menu меню.дата%TYPE
);
rec_menu rec_menu_type;
BEGIN
NULL; -- пустой оператор (см. разд. 17.4.4)
END;
/
```

При объявлении типа записи можно присвоить ее полям некоторые значения. Если же для поля вводится ограничение `NOT NULL` (для предотвращения назначения пустых значений), то этому полю надо обязательно присвоить значение. Например:

```
TYPE BludoRecTyp IS RECORD (код_блюда NUMBER(2) NOT NULL := 11,
блюдо VARCHAR2(16), код_вида NUMBER(1), основа VARCHAR2(6),
выход Блюда.выход%TYPE, труд NUMBER(3));
```

или

```
TYPE BludoRecTyp IS RECORD (Блюда%ROWTYPE);
```

Объявление создаваемой записи (например, *имя_plsql_записи*) производится в разделе `DECLARE` и имеет вид:

имя_plsql_записи имя_типа_записи;

Ссылки на отдельные поля записи осуществляются так:

имя_plsql_записи.имя_поля;

Для присвоения значения конкретному полю записи используется синтаксис:

имя_plsql_записи.имя_поля := *выражение*;

Примеры использования записей в программах PL/SQL приведены в *разд. 17.7*.

17.4. Команды управления ходом выполнения программы

17.4.1. Команды условного перехода (*IF*...)

Существует три модификации оператора условного перехода:

- IF-THEN (если-то):

```
IF условие THEN
  последовательность_команд;
END IF;
```

- IF-THEN-ELSE (если-то-иначе):

```
IF условие THEN
  1_последовательность_команд;
ELSE
  2_последовательность_команд;
END IF
```

- IF-THEN-ELSIF (если-то-иначе-если):

```
IF условие1 THEN
  1_последовательность_команд;
ELSIF условие2 THEN
  2_последовательность_команд;
ELSIF условиеN THEN
  N_последовательность_команд;
[ ELSE
  N+1_последовательность_команд;]
END IF
```

Во всех модификациях, если *условие* или *условие1* истинно, то выполняется *последовательность_команд* или *1_последовательность_команд* и управление передается на первый оператор после END IF. Если же оно ложно, то:

- в модификации IF-THEN управление передается на первый оператор после END IF;
- в модификации IF-THEN-ELSE выполняется *2_последовательность_команд* и управление передается на первый оператор после END IF;
- в модификации IF-THEN-ELSIF проверяется *условие2*; если оно истинно, то выполняется *2_последовательность_команд* и управление передается на первый оператор после END IF; если *условие1* и *условие2* ложны,

а *условие3* истинно, то выполняется *3_последовательность_команд* и управление передается на первый оператор после `END IF`; наконец, если условия 1, 2, ..., N ложны, то выполняется N+1 последовательность команд и управление передается на первый оператор после `END IF`.

Все это справедливо, если внутри последовательности команд нет операторов, осуществляющих переход за пределы этой последовательности.

17.4.2. Метки и оператор безусловного перехода (*GOTO*)

В любом месте программы может быть поставлена метка, имеющая синтаксис:

<<имя_метки>>

Оператор `GOTO` позволяет осуществить безусловный переход к метке, имя которой должно быть уникальным внутри программы или блока PL/SQL. Например, управление передается вниз к помеченному оператору:

```
BEGIN
 ...
    GOTO insert_row;
 ...
    <<insert_row>>
    INSERT INTO Блюдо VALUES ...
END;
```

В следующем примере управление передается вверх к помеченной последовательности операторов:

```
BEGIN
    ...
    <<update_row>>
    BEGIN
        UPDATE Блюдо SET ...
        ...
    END;
    ...
    GOTO update_row;
    ...
END;
```

Следует отметить, что использование `GOTO` (особенно в тех случаях, когда метка предшествует оператору `GOTO`) может привести к сложным, нераспознаваемым кодам ошибок, которые трудно обрабатывать. Поэтому реже

используйте GOTO, тем более что этот оператор нельзя использовать для выполнения перехода:

- в IF-блок, LOOP-блок или в другой блок, не включающий текущий;
- из одного предложения IF-оператора к другому;
- из внешнего блока во внутренний блок;
- из обработчика особых ситуаций в текущий блок.

17.4.3. Операторы цикла (*LOOP*, *WHILE...LOOP* и *FOR...LOOP*)

Циклы служат для повторяемого выполнения последовательности команд. В PL/SQL используются три модификации операторов цикла: LOOP, WHILE...LOOP и FOR...LOOP.

Цикл LOOP имеет следующий синтаксис:

```
LOOP
 последовательность_команд;
END LOOP;
```

и приводит к бесконечному повторению *последовательности_команд*, если внутри нее нет команд EXIT (выход из цикла), RAISE (вызов обработчика исключительных ситуаций) или GOTO (безусловный переход). Например,

```
LOOP
 последовательность_команд;
  IF условие THEN
    EXIT;
  END IF;
END LOOP;
```

приведет к выходу из цикла после выполнения *последовательности_команд*, как только *условие* станет истинным.

Цикл WHILE предназначен для повторения *последовательности_команд*, пока *условие* остается истинным:

```
WHILE условие LOOP
  последовательность_команд;
END LOOP;
```

Цикл FOR является наиболее распространенной модификацией цикла и имеет следующий синтаксис:

```
FOR индекс IN [REVERSE] нижняя_граница..верхняя_граница LOOP
  последовательность_команд;
END LOOP;
```

Здесь *индекс* (счетчик циклов) изменяется от нижней до верхней границы с шагом 1, а при использовании REVERSE — от верхней до нижней границы с шагом –1. Например,

```
FOR i IN 1..3 LOOP             -- для i = 1, 2, 3
  последовательность_команд;   -- цикл выполняется 3 раза
END LOOP;

FOR i IN REVERSE 1..3 LOOP     -- для i = 3, 2, 1
  последовательность_команд;   -- цикл выполняется 3 раза
END LOOP;
```

Если нижняя граница равна верхней, последовательность выполняется один раз. Если нижняя граница больше верхней, последовательность не выполняется, и управление переходит к следующему за циклом оператору.

Пределы диапазона цикла могут быть литералами, переменными или выражениями, но должны быть целыми числами. Например, допустимы следующие диапазоны:

```
j IN -5..5
k IN REVERSE first..last
step IN 0..TRUNC(high/low) * 2
code IN ASCII('A')..ASCII('J')
```

Объявлять индекс не нужно — он объявлен неявно как локальная переменная типа integer.

PL/SQL позволяет определять диапазон цикла динамически во время выполнения. Например:

```
SELECT COUNT(код_блюда) INTO bluda_count FROM Блюда;
  FOR i IN 1..bluda_count LOOP
      ...
  END LOOP;
```

Значение bluda_count не известно при компиляции; предложение SELECT определяет это значение во время выполнения (синтаксис SELECT…INTO рассмотрен в *разд. 17.5.1*).

Индекс может использоваться в выражениях внутри цикла, но он не может изменяться.

Индекс определен только внутри цикла, и на него нельзя ссылаться снаружи цикла. После выполнения цикла индекс не определен.

Подобно PL/SQL блокам, циклы могут быть помечены. Метка устанавливается перед оператором LOOP, и ее имя может быть указано после соответствующего END LOOP:

```
<<имя_метки>>
LOOP
  последовательность_команд;
END LOOP [имя_метки];
```

Помеченные циклы используются для улучшения чтения программы (разборчивости).

С любой формой утверждения EXIT можно завершать не только текущий цикл, но и любой внешний цикл. Для этого маркируйте внешний цикл, который надо завершить, и используйте метку в утверждении EXIT следующим образом:

```
<<outer>>
LOOP
    ...
    LOOP
        ...
        EXIT outer WHEN ...  -- завершаются оба цикла
    END LOOP;
    ...
END LOOP outer;
```

Если требуется преждевременно выйти из вложенного цикла FOR, маркируйте цикл и используйте метку в утверждении EXIT. Например:

```
<<outer>>
FOR i IN 1..5 LOOP
    ...
    FOR j IN 1..10 LOOP
        FETCH s1 INTO ShRec;
        EXIT outer WHEN s1%NOTFOUND;  -- завершаются оба цикла
        ...
    END LOOP;
END LOOP outer;
-- управление передается сюда
```

17.4.4. Операторы *EXIT*, *EXIT-WHEN* и *NULL*

EXIT используется для завершения цикла, когда дальнейшая обработка нежелательна или невозможна. Внутри цикла можно помещать один или большее количество операторов EXIT. Имеются две формы EXIT: EXIT и EXIT-WHEN.

По оператору EXIT цикл завершается немедленно и управление переходит к следующему за END LOOP оператору.

По оператору EXIT-WHEN цикл завершится только в том случае, когда становится истинным условие в предложении WHEN.

Оператор EXIT-WHEN позволяет завершать цикл преждевременно. Например, следующий цикл обычно выполняется десять раз, но как только не находится значение s1, цикл завершается независимо от того, сколько раз цикл выполнился.

```
FOR j IN 1..10 LOOP
  FETCH s1 INTO ShRec;
  EXIT WHEN s1%NOTFOUND;  -- выход при отсутствии возвращаемой строки
    ...
END LOOP;
```

NULL — пустой оператор; он передает управление к следующему за ним оператору. Однако к нему может передаваться управление и его наличие часто улучшает читаемость программы. Он также полезен для создания фиктивных подпрограмм для резервирования областей определения функций и процедур при отладке программ.

17.5. SQL-предложения в PL/SQL

Из всех SQL-предложений в программах PL/SQL можно применять лишь предложения DML (см. *разд. 4.4.1*) и управления транзакциями. Предложения DDL использовать нельзя. Предложение EXPLAIN PLAN (объяснить план) — хотя оно и относится к категории DML — применять также не разрешается. Чтобы пояснить смысл этих ограничений, рассмотрим принципы создания программ PL/SQL.

В любом языке программирования привязка переменных может быть либо ранней, либо поздней. *Привязка* (binding) переменной — это процесс указания области памяти, соответствующей идентификатору программы.

В PL/SQL в процесс привязки входит также проверка базы данных на наличие полномочий, позволяющих обращаться к объектам схем. В том языке, где используется *ранняя привязка* (early binding), этот процесс осуществляется на этапе компиляции программы, а в языке, где применяется *поздняя привязка* (late binding), она откладывается до этапа выполнения программы. Ранняя привязка означает, что компиляция программы будет занимать большее время (так как при этом нужно привязывать переменные), однако выполняться такая программа будет быстрее, потому что привязка уже завершена. Поздняя

привязка сокращает время компиляции, но увеличивает время выполнения программы.

При разработке PL/SQL было принято решение об использовании ранней привязки, чтобы к моменту выполнения блока объекты базы данных были уже проверены, и блок выполнялся максимально быстро. Это вполне оправданно, поскольку блоки PL/SQL можно хранить в базе данных как процедуры, функции, пакеты (модули) и триггеры. Такие объекты хранятся в скомпилированном виде, т. е. при необходимости их можно загрузить из базы данных и выполнить (см. *главу 18*). Именно поэтому нельзя использовать операторы DDL. Оператор DDL модифицирует объект базы данных, следовательно, полномочия на объект должны быть подтверждены вновь. Процесс подтверждения полномочий требует привязки идентификаторов, а это уже было сделано во время компиляции.

Тем не менее, существует способ, обеспечивающий выполнение в PL/SQL всех допустимых предложений SQL, включая DDL. Это динамический SQL (см. *разд. 17.8*). Он позволяет создавать оператор SQL динамически, во время выполнения программы, а затем проводить его синтаксический анализ и выполнение. Такой оператор до момента выполнения программы фактически еще не создан, поэтому от компилятора PL/SQL не требуется привязывать идентификаторы этого оператора, что дает возможность скомпилировать блок.

17.5.1. *SELECT...INTO*

В тех случаях, когда программе необходимо иметь значения столбцов из одной строки таблицы, можно воспользоваться предложением `SELECT...INTO`, формат которого имеет вид:

```
SELECT [{ALL | DISTINCT}] отбираемый_элемент [AS псевдоним] [,...]
INTO имя_переменной [,...] | имя_записи
FROM [ONLY | OUTER]
   { имя_таблицы [[AS] псевдоним] |
    имя_представления [[AS] псевдоним] } [,...]
   [ [тип_соединения] JOIN условие_соединения ]
[WHERE условие_поиска [ {AND | OR | NOT} условие_поиска [...]]
[GROUP BY группировка_по_выражению { группировка_по_столбцам |
     ROLLUP группировка_по_столбцам | CUBE группировка_по_столбцам |
     GROUPING SETS ( список_наборов_группировок ) | ( ) |
      набор_группировок, список__наборов_ группировок }
   [HAVING условие_поиска] ];
```

Практически это обычный SELECT (см. *разд. 5.1*), выполняющий присвоение выбираемых значений столбцов переменным, перечисленным во фразе INTO. Однако такое присвоение происходит только в том случае, если фразы WHERE или GROUP BY обеспечивают возвращение по запросу *лишь одной строки* и переменные заранее описаны в декларативной части блока PL/SQL.

17.5.2. *INSERT*, *UPDATE* и *DELETE*

Эти предложения отличаются от аналогичных предложений интерактивного SQL, в основном, лишь тем, что в их *скалярных_выражениях* могут использоваться переменные PL/SQL.

Кроме того, в WHERE-фразах предложений UPDATE и DELETE может быть указана конструкция CURRENT OF *имя_курсора* (см. *разд. 17.7.2*). Также необходимо обеспечить, чтобы эти предложения производили изменения только по одной строке.

17.6. Обработка ошибок

Нельзя создать приложение, которое будет безошибочно работать в любых ситуациях: возможны аппаратные сбои, невыявленные ошибки приложения и ошибки из-за некорректных действий пользователей приложения (клиентов). Если при этом программная ошибка произошла в блоке PL/SQL, вложенном в другой блок, а тот, в свою очередь, вложен в третий блок и т. д., то она может дойти до клиентского приложения. Чтобы устранить возможную отмену большого объема ранее выполненных операций и трафик из-за возвращаемых клиенту ошибок, чтобы посылать клиенту точные сообщения о причине ошибки и способе ее устранения (если она все же дошла до клиента), разработчики приложения должны предусматривать возможные программные ошибки и создавать процедуры, адекватно реагирующие на них.

В PL/SQL предусмотрены механизмы перехвата и обработки ошибок, возникающих при выполнении программы. Эти механизмы называются *исключительными ситуациями*. Когда программа обнаруживает заданное условие ошибки, то вызывается соответствующая исключительная ситуация. Обработка исключительных ситуаций в программе производится в разделе EXCEPTION (см. *разд. 17.2*).

При обнаружении исключительной ситуации, обработка основного тела программы останавливается, и управление передается соответствующему обработчику исключительной ситуации, который определяет дальнейшие действия.

В PL/SQL используются следующие типы исключительных ситуаций:

- встроенные исключительные ситуации;
- исключительные ситуации, определяемые пользователем;
- обработчик OTHERS.

Примеры использования механизмов перехвата и обработки ошибок приведены в *разд. 18.4*.

17.6.1. Встроенные исключительные ситуации

Oracle включает двадцать две встроенные исключительные ситуации, соответствующих типовым ошибкам, приведенным в табл. 17.1.

Таблица 17.1. *Встроенные исключительные ситуации*

Исключительная ситуация	Ошибка Oracle	Описание
CURSOR_ALREADY_OPEN	ORA-06511	Попытка открытия уже открытого курсора
DUP_VAL_ON_INDEX	ORA-00001	Попытка вставить дубликат значения для уникального индекса
INVALID_CURSOR	ORA-01001	Попытка выполнения запрещенной операции с курсором (например, закрытие неоткрытого курсора)
INVALID_NUMBER	ORA-01722	Отказ преобразования строки символов в число
LOGIN_DENIED	ORA-01017	Неправильное имя пользователя/пароль
NO_DATA_FOUND	ORA-01403	Предложение SELECT...INTO возвращает ноль строк
NOT_LOGGED_ON	ORA-01012	Нет подключения к Oracle
PROGRAM_ERROR	ORA-06501	Внутренняя ошибка PL/SQL
STORAGE_ERROR	ORA-06500	Пакет PL/SQL вышел из пределов памяти или если память разрушена
TIMEOUT_ON_RESOURCE	ORA-00051	Истекло время ожидания ресурса Oracle

Таблица 17.1 *(окончание)*

Исключительная ситуация	Ошибка Oracle	Описание
TOO_MANY_ROWS	ORA-01422	Предложение SELECT...INTO возвращает более одной строки
TRANSACTION_BACKED_OUT	ORA-00061	Удаленный сервер отменил транзакцию
ZERO_DIVIDE	ORA-01476	Попытка деления на ноль
PROGRAM_ERROR	ORA-06501	Внутренняя ошибка PL/SQL
VALUE_ERROR	ORA-06502	Ошибка усечения, арифметическая ошибка или ошибка преобразования
ROWTYPE_MISMATCH	ORA-06504	Базовая курсорная переменная и курсорная переменная PL/SQL имеют несовместимые типы строк
ACCESSJNTO_NUll	ORA-06530	Попытка присвоить значение атрибуту NULL-объекта
COLLECTION IS NULL	ORA-06531	Попытка применить к таблице или изменяемому массиву PL/SQL, содержащему NULL, метод, отличный от EXISTS
SUBSCRIPT_OUTSIDE_LIMIT	ORA-0653	Ссылка на индекс вложенной таблицы или изменяемого массива, лежащий вне объявленного диапазона (например, –1)
SUBSCRIPT_BEYOND_COUNT	ORA-06533	Ссылка на индекс таблицы или изменяемого массива, больший, чем число элементов данной сборной конструкции
CASE_NOT_FOUND	ORA-06592	Не найдено соответствующее предложение WHEN в операторе CASE
SELF IS NULL	ORA-30625	Попытка вызвать метод экземпляра NULL-объекта

Если в раздел EXCEPTION программы (блока) включена фраза

```
WHEN имя_исключения THEN текст_обработчика_исключения;
```

с именем какого-либо встроенного исключения и возникла соответствующая ошибка, то вместо прекращения исполнения программы и выдачи типового сообщения об ошибке, будет исполняться созданный пользователем текст обработчика исключения. Такой обработчик может, например, выяснить ситуацию, при которой произошло деление на ноль, и выдать правдоподобный результат операции деления или прервать исполнение программы и дать сообщение об изменении каких-либо данных. В последнем случае это может быть не типовое сообщение "Вы пытаетесь делить на ноль", а любое подготовленное пользователем сообщение.

Для выдачи сообщения об ошибке, обеспечения возврата в среду, из которой вызывалась текущая программа (блок) и отмены всех действий, выполненных в текущей транзакции, целесообразно использовать процедуру

```
RAISE_APPLICATION_ERROR (номер_ошибки, сообщение);
```

где *номер_ошибки* — отрицательное целое число в диапазоне от 20 000 до 20 999 и *сообщение* — символьная строка длиной до 2048 символов.

17.6.2. Исключительные ситуации, определяемые пользователем

Кроме встроенных могут быть использованы собственные исключительные ситуации, имена которых необходимо описать в разделе DECLARE блока PL/SQL (например, err_nachalo EXCEPTION). В разделе EXCEPTION блока должен быть описан соответствующий обработчик исключительной ситуации, например:

```
WHEN err_nachalo THEN RAISE_APPLICATION_ERROR(-20013,
      'Дата начала должна быть больше '||to_char(nach));
```

Определяемые пользователем ошибки обычно проверяются в основной программе с помощью операторов условия (IF-THEN). Для передачи управления обработчику пользовательской исключительной ситуации в случае обнаружения ошибки используется предложение:

```
RAISE имя_пользовательского_исключения
```

Например:

```
IF :new.nachalo <> kon + 1 THEN
  RAISE err_nachalo;
END IF;
```

17.6.3. Обработчик *OTHERS*

Если исключительная ситуация явно не обрабатывается в блоке и для ее перехвата не используется обработчик OTHERS, то PL/SQL отменяет выполняемые блоком транзакции и возвращает необработанную исключительную ситуацию обратно в вызывающую среду.

Обработчик особых ситуаций OTHERS описывается последним в блоке для перехвата всех исключительных ситуаций. Он может иметь вид:

```
WHEN OTHERS THEN RAISE_APPLICATION_ERROR (-20099,'Какая-то другая
ошибка');
```

17.7. Курсоры

17.7.1. Связь объектов PL/SQL с таблицами базы данных

Чтобы программа PL/SQL могла работать с информацией, содержащейся в базах данных, необходимо организовать обмен между значениями столбцов таблиц баз данных и переменными PL/SQL.

Известно, что для выбора информации из таблиц используется SQL-предложение SELECT. При его выполнении Oracle создает специальную рабочую область, содержащую:

- информацию о самом SELECT,
- данные, которые требуются для его выполнения (например, результаты подзапросов),
- окончательный результат выполнения SELECT.

PL/SQL имеет несколько механизмов доступа к этой рабочей области. Одним из них является курсор, с помощью которого можно присвоить имя этой рабочей области и манипулировать содержащейся в ней информацией, последовательно выбирая строки результата и пересылая значения столбцов текущей строки в переменные PL/SQL. Существуют и другие механизмы, не требующие создания явного курсора.

17.7.2. Явный курсор

Курсор — это средство языка SQL, позволяющее с помощью команд OPEN, FETCH и CLOSE получить построчный доступ к результату запроса к базе данных.

Будем также называть курсором и сам набор строк, полученный в результате выполнения запроса.

Для использования курсора его надо сначала объявить, т. е. дать ему имя и указать (с помощью предложения SELECT), какие столбцы и строки базовых таблиц должны быть помещены в набор строк, названный этим именем.

Команда OPEN инициализирует получение указанного набора и установку перед первой его строкой указателя текущей строки. Команда FETCH служит для:

- выборки из текущей строки курсора значений указанных столбцов с пересылкой их в переменные PL/SQL;
- установки указателя текущей строки на следующую запись.

Выполнением FETCH в цикле можно последовательно выбрать информацию из всех строк курсора.

Наконец, команда CLOSE позволяет закрыть (удалить из памяти) набор строк (при этом описание курсора сохраняется и его можно снова открыть командой OPEN).

Существует модификация ("Курсор в цикле FOR"), позволяющая организовать последовательный выбор строк объявленного курсора без явного использования команд OPEN, FETCH и CLOSE (см. *разд. 17.7.2*).

Объявление курсора

Перед работой с курсором его следует объявить в разделе DECLARE или другом допустимом разделе, используя синтаксис:

```
CURSOR имя_курсора [ (параметр, параметр, ... ] ) ] IS SELECT ...
```

где *имя_курсора* — имя курсора, формируемое по правилам, описанным, например, в *разд. 4.4.2*.

SELECT **...** — предложение SELECT, определяющее строки курсора.

параметр имеет следующий синтаксис:

```
имя_переменной [IN] имя_типа [ { := | DEFAULT } значение ]
```

а *имя_типа* — любой тип данных PL/SQL без указания ограничений (например, длины символьных значений).

Формальные параметры курсора используются только для передачи значений в WHERE-фразу предложения SELECT с целью отбора нужных строк запроса. Передача таких значений производится во время открытия курсора командой OPEN. Если значения формальных параметров отсутствуют в команде OPEN и не заданы по умолчанию (:=*значение* или DEFAULT *значение*), то выдается

ошибка. При наличии параметров в описании курсора и команде `OPEN`, используются параметры из команды `OPEN`.

Пример курсора приведен в листинге 17.1.

Листинг 17.1. Пример явного курсора

```
SET SERVEROUTPUT ON; -- разрешение вывода на экран информации, заданной в
                     -- DBMS_OUTPUT
DECLARE
  trap VARCHAR2(7); -- входной параметр курсора (Имя трапезы)
  mesto INTEGER;    -- входной параметр курсора (Место за столом)
  date_m DATE;      -- входной параметр курсора (Дата меню)
  -- Описание курсора с входными параметрами
  CURSOR menu_mesto (trap VARCHAR2 := 'Завтрак', mesto INTEGER := 20,
                     date_m DATE := '15.05.1989') IS
  -- Запрос, определяющий строки курсра
  SELECT трапеза,вид,блюдо,дата FROM меню,блюда,виды_блюд,трапезы,выбор
  WHERE меню.код_блюда = блюда.код_блюда
  AND блюда.код_вида = виды_блюд.код_вида
  AND меню.код_трапезы = трапезы.код_трапезы
  AND меню.строка = выбор.строка
  AND трапезы.трапеза = trap
  AND выбор.место = mesto
  AND меню.дата = date_m;
  -- Описание типа записи
  TYPE rec_menu_type IS RECORD
    (trapeza трапезы.трапеза%TYPE,
     vid виды_блюд.вид%TYPE,
     bludo блюда.блюдо%TYPE,
     date_menu меню.дата%TYPE
    );
  -- Описание записи, использующейся при выборе строк
  rec_menu rec_menu_type;
BEGIN
  OPEN menu_mesto; -- Открытие курсора
    LOOP --Начало цикла по выбору строк курсора
      FETCH menu_mesto INTO rec_menu;  -- Выбор строк курсора в запись
         EXIT WHEN menu_mesto%NOTFOUND; -- Выход при отсутствии
                                        -- возвращаемой строки
```

```
      -- Вывод на экран полей записи
      DBMS_OUTPUT.PUT_LINE(rec_menu.trapeza||' '||rpad(rec_menu.vid,10)||
      rpad(rec_menu.bludo,20)||rec_menu.date_menu);
    END LOOP;  -- Конец цикла
  CLOSE menu_mesto; -- Закрытие курсора
END;
/
```

Открытие курсора (*OPEN*)

Команда `OPEN` имеет следующий синтаксис:

`OPEN` *имя_курсора* `[ (`*значение* `[,`*значение*`] ...) ];`

где список значений `(`*значение* `[,`*значение*`] ...)` используется для передачи параметров курсора и должен по числу и типу данных совпадать с описанием этих параметров.

Команда выполняет объявленное в курсоре из листинга 17.1 предложение `SELECT...`, используя (если есть параметры) передаваемые из `OPEN` значения или значения, указанные при объявлении курсора, создавая набор строк и устанавливая указатель текущей строки перед первой из них. Так, по команде

```
OPEN menu_mesto;
```

будет создан набор, приведенный в листинге 17.2.

Листинг 17.2

```
Завтрак Закуска   Салат витаминный    15.05.89
Завтрак Горячее   Пудинг рисовый      15.05.89
Завтрак Напиток   Молочный напиток    15.05.89
```

где использовались значения параметров, заданные при описании.

По команде

```
OPEN menu_mesto ('Обед', 20, '15.05.1989');
```

будет создан другой набор — листинг 17.3.

Листинг 17.3

```
Обед Закуска   Салат летний        15.05.89
Обед Суп       Суп харчо           15.05.89
Обед Горячее   Сырники             15.05.89
Обед Десерт    Яблоки печеные      15.05.89
```

И, наконец, по команде

```
OPEN menu_mesto ('Ужин', 17, '15.05.1989');
```

выдается набор из листинга 17.4.

Листинг 17.4

```
Ужин Закуска   Мясо с гарниром     15.05.89
Ужин Горячее   Драчена             15.05.89
Ужин Напиток   Молочный напиток    15.05.89
```

Выборка строк из курсора (*FETCH*)

Команда FETCH, используемая для продвижения на один шаг указателя текущей строки курсора и пересылки ее значений в переменные или запись, имеет следующий синтаксис:

```
FETCH имя_курсора INTO { имя_переменной1[,имя_переменной2]...} |
имя_записи ;
```

Для каждого значения столбца, возвращенного запросом, в списке INTO должна иметься переменная или поле записи соответствующего типа данных. Такие переменные или записи должны быть заранее описаны в декларативной части блока PL/SQL.

Закрытие курсора (*CLOSE*)

Команда CLOSE используется для освобождения всех ресурсов, которые поддерживались открытым курсором (при этом описание курсора сохраняется и его можно снова открыть командой OPEN). Синтаксис команды CLOSE имеет вид:

```
CLOSE имя_курсора;
```

Использование курсора в цикле *FOR*

В большинстве ситуаций, которые требуют явного курсора, текст программы может быть упрощен при использовании "курсора в цикле FOR", заменяющего команды OPEN, FETCH и CLOSE. Курсор в цикле FOR:

- неявно объявляет индекс цикла записью, поля которой соответствуют столбцам (псевдонимам) предложения SELECT... из описания курсора;
- передает параметры курсора (если они есть) и открывает курсор;

- выбирает в цикле строки из полученного набора в индекс цикла (поля записи);
- закрывает курсор после обработки всех строк набора или по досрочному выходу из него с помощью команд `EXIT` или `GOTO`.

Синтаксис курсора в цикле `FOR` имеет вид:

```
FOR имя_индекса_цикла IN имя_курсора [ (значение [,значение]...) ] LOOP
  тело_цикла
END LOOP;
```

где *имя_индекса_цикла* — индекс цикла, в котором при первом прохождении цикла хранится первая строка набора, при втором прохождении цикла вторая строка и т. д.

(*значение* [,*значение*]...) — список значений, используемый для передачи параметров курсора (он заменяет в данном случае список из команды `OPEN`).

тело_цикла содержит нужные строки повторяющейся части программы, в которых используются переменные с именами *имя_индекса_цикла.имя_столбца*, а *имя_столбца* — имя столбца из перечня столбцов предложения `SELECT` в описании курсора.

Вот как сокращается текст предыдущего анонимного блока при использовании цикла `FOR`:

```
SET SERVEROUTPUT ON;
DECLARE
  trap VARCHAR2(7); -- входной параметр курсора (Имя трапезы)
  mesto INTEGER;    -- входной параметр курсора (Место за столом)
  date_m DATE;      -- входной параметр курсора (Дата меню)
  -- Описание курсора с входными параметрами
  CURSOR menu_mesto (trap VARCHAR2 := 'Завтрак', mesto INTEGER := 20,
                     date_m DATE := '15.05.1989') IS
  -- Запрос, определяющий строки курсра
  SELECT трапеза,вид,блюдо,дата FROM меню,блюда,виды_блюд,трапезы,выбор
  WHERE меню.код_блюда = блюда.код_блюда
  AND блюда.код_вида = виды_блюд.код_вида
  AND меню.код_трапезы = трапезы.код_трапезы
  AND меню.строка = выбор.строка
  AND трапезы.трапеза = trap
  AND выбор.место = mesto
  AND меню.дата = date_m;
BEGIN
  FOR rec_menu IN menu_mesto LOOP
```

```
      -- Вывод на экран полей записи
      DBMS_OUTPUT.PUT_LINE(rec_menu.трапеза||' '||rpad(rec_menu.вид,10)||
      rpad(rec_menu.блюдо,20)||rec_menu.дата);
    END LOOP;
END;
/
```

В итоге будет получен результат, приведенный в листинге 17.2, а при использовании параметров во фразе `FOR`:

```
FOR rec_menu IN menu_mesto ('Ужин', 17, '15.05.1989') LOOP,
```

получим листинг 17.4.

Атрибуты явного курсора

Для анализа состояния курсора используются специальные переменные, имена которых составляются из имени курсора и суффиксов `%FOUND`, `%NOTFOUND`, `%ROWCOUNT` и `%ISOPEN`, называемых атрибутами курсора. Если курсор назван *имя_курсора*, то эти переменные имеют имена: *имя_курсора*`%NOTFOUND`, *имя_курсора*`%FOUND`, *имя_курсора*`%ROWCOUNT` и *имя_курсора*`%ISOPEN`.

Значения таких переменных анализируются при выполнении программы с помощью различных операторов управления (`IF-THEN`, `EXIT WHEN` и т. п.). Эти операторы изменяют (при необходимости) ход выполнения программы. Следует отметить, что ссылка на специальные переменные до открытия курсора приводит к появлению сообщения `INVALID_CURSOR`.

Переменная с атрибутом `%ISOPEN` позволяет определить, открыт ли курсор. Если он открыт, то эта переменная возвращает `TRUE`, иначе — `FALSE`. Например:

```
IF NOT menu_mesto %ISOPEN THEN  -- курсор не открыт ?
  OPEN menu_mesto;              -- открыть курсор !
IF END;
FETCH ...
```

Переменные с `%NOTFOUND` и `%FOUND` атрибутами показывают состояние текущей позиции курсора (перед первой выборкой строки курсора обе переменных имеют значение `NULL`). Переменная с `%NOTFOUND` принимает значение `FALSE` тогда, когда выборка возвратила строку (при этом переменная с `%FOUND` принимает значение `TRUE`). Если же в результате выборки строка не возвращается, то переменные с `%NOTFOUND` и `%FOUND` принимают значения `TRUE` и `FALSE` соответственно. Пример использования `%NOTFOUND` был рассмотрен в листинге 17.1.

Переменная с атрибутом `%ROWCOUNT` содержит количество строк, выбранных из курсора на текущий момент (при открытии курсора эта переменная содер-

жит ноль). В следующем примере переменная menu_mesto%ROWCOUNT ограничивает выборку из курсора menu_mesto двумя строками:

```
LOOP
  FETCH menu_mesto INTO rec_menu;
  IF menu_mesto%ROWCOUNT <=2 THEN
    DBMS_OUTPUT.PUT_LINE(rec_menu.trapeza||' '||rpad(rec_menu.vid,10)||
                         rpad(rec_menu.bludo,20)||rec_menu.date_menu);
  ELSE
    EXIT;
  END IF;
END LOOP;
```

Изменение или удаление текущей строки курсора

Существуют два предложения, позволяющие изменить или удалить ту строку таблицы базы данных, на которую позиционирована текущая строка курсора:

```
UPDATE [имя_схемы.]имя_таблицы | имя представления} [псевдоним]
  SET { (имя_столбца [,имя_столбца] ...) = (подзапрос)
      |  имя_столбца = { expr | (subquery) } }
   [, { (имя_столбца [,имя_столбца] ...) = (подзапрос)
      |  имя_столбца = { expr | (подзапрос) } } ] ...
  WHERE CURRENT OF имя_курсора;

DELETE [FROM] [имя_схемы.]имя_таблицы | имя представления} [псевдоним]
  WHERE CURRENT OF имя_курсора;
```

Для этого необходимо, чтобы при объявлении курсора предложение SELECT... содержало фразу:

```
FOR UPDATE OF [[имя_схемы.]имя_таблицы | имя представления] имя_столбца
   [, [[имя_схемы.]имя_таблицы | имя представления] имя_столбца ] ... ;
```

в которой следует привести список обновляемых столбцов.

17.7.3. Неявный курсор (SQL-курсор)

Для всех команд языка SQL, не связанных с объявлением курсора (явного курсора), PL/SQL открывает курсор (неявный курсор), на который можно ссылаться по курсорному имени SQL%. При работе с таким курсором нельзя использовать команды OPEN, FETCH и CLOSE, но можно использовать атрибуты курсора, чтобы получить информацию о его текущем состоянии.

Атрибуты неявного курсора (SQL-курсора)

Для анализа результата выполнения предложений SELECT...INTO, INSERT, UPDATE и DELETE используются три переменные: SQL%NOTFOUND, SQL%FOUND и SQL%ROWCOUNT (Oracle закрывает SQL-курсор сразу после выполнения SQL-предложения, что делает бессмысленным использование переменной SQL%ISOPEN, так как ее значение всегда равно FALSE).

Перед выполнением предложений SELECT...INTO, INSERT, UPDATE и DELETE переменные SQL%NOTFOUND и SQL%FOUND имеют значение NULL. Переменная SQL%NOTFOUND принимает значение TRUE, если INSERT, UPDATE и DELETE не произвели изменений таблиц базы данных или SELECT...INTO не возвратил строк (при этом переменная SQL%FOUND принимает значение FALSE). В противном случае переменная SQL%NOTFOUND принимает значение FALSE, а переменная SQL%FOUND — TRUE.

17.8. Динамический SQL в PL/SQL

PL/SQL использует раннее связывание для выполнения предложений SQL. Следствием этого является то, что только предложения DML могут непосредственно включаться в блоки PL/SQL. Однако можно решить эту проблему с помощью динамического SQL. Динамический SQL разбирается и исполняется во время выполнения, а не синтаксического разбора блока PL/SQL.

Существуют два способа выполнения динамического SQL в PL/SQL: использование встроенного динамического SQL или встроенного модуля (пакета) DBMS_SQL.

Встроенный динамический SQL, появившийся в Oracle 8, является составной частью самого языка. Вследствие этого он значительно проще в применении и быстрее, чем пакет DBMS_SQL.

Базовым оператором, используемым в предложениях DML, DDL и блоках PL/SQL, является предложение EXECUTE IMMEDIATE (выполнить немедленно). В нем производится подготовка и немедленное выполнение заданного в текстовой форме оператора SQL, который не должен содержать формальных параметров и комментариев.

Синтаксис EXECUTE IMMEDIATE выглядит следующим образом:

```
EXECUTE IMMEDIATE строковое_выражение
[INTO { имя_переменной [,имя_переменной]... | имя_записи }]
[USING [IN | OUT | IN OUT] имя_параметра
    [, [IN | OUT | IN OUT] имя_параметра]...];
```

где *строковое_выражение* — SQL-предложение или PL/SQL-блок.

имя_переменной — переменная, в которой сохраняется значение выбранного столбца.

имя_записи — запись, которая должна иметь тип, объявленный пользователем или %ROWTYPE, в ней будет содержаться выбранная строка.

имя_параметра — параметр, значение которого передается в динамическое SQL-предложение или PL/SQL-блок.

Каждый параметр необходимо включить в оператор USING. Тип параметра по умолчанию IN, если он не указан явно. Во время выполнения программы каждый параметр оператора USING замещает соответствующий шаблон SQL-предложения или PL/SQL-блока. Поэтому каждому шаблону должен соответствовать параметр оператора USING. В операторе USING возможно использование числовых, символьных и строковых значений. Значения типа Boolean (TRUE, FALSE, NULL) не допускаются.

Приведем примеры использования динамического SQL.

Пример 17.1. Пояснение особенности синтаксиса.

```
SET SERVEROUTPUT ON; -- для вывода на экран информации по DBMS_OUTPUT
DECLARE
   sql_stmt VARCHAR2(100);
   plsql_block VARCHAR2(200);
   kod_trapezy NUMBER(1) := 4;
   trapeza     VARCHAR2(7) := 'Полдник';
   trap_rec трапезы%ROWTYPE;
BEGIN
-- Вставить строку в таблицу Трапезы, используя параметры
   sql_stmt := 'INSERT INTO трапезы VALUES (:1, :2)';
   EXECUTE IMMEDIATE sql_stmt USING kod_trapezy, trapeza;
-- Удалить строку из таблицы Трапезы, используя параметр
   EXECUTE IMMEDIATE 'DELETE FROM трапезы WHERE код_трапезы = :n' USING
      kod_trapezy;
-- Напечатать указанную строку таблицы Трапезы
   sql_stmt := 'SELECT * FROM трапезы WHERE код_трапезы = :kt';
   EXECUTE IMMEDIATE sql_stmt INTO trap_rec USING 3;
   DBMS_OUTPUT.PUT_LINE(trap_rec.код_трапезы||' '||trap_rec.трапеза);
-- Напечатать содержимое таблицы с помощью анонимного блока PL/SQL.
-- Здесь целый блок (включая точку с запятой) помещается в одну строку.
   plsql_block :=
    'BEGIN
```

```
      FOR trap_rec IN (SELECT * FROM трапезы) LOOP
        DBMS_OUTPUT.PUT_LINE
                    (trap_rec.код_трапезы||'' ''||trap_rec.трапеза);
      END LOOP;
    END;';
  EXECUTE IMMEDIATE plsql_block;
-- Создать таблицу temp
  EXECUTE IMMEDIATE 'CREATE TABLE temp (t1 number(2), t2 varchar2(2))';
-- Удалить таблицу temp
  EXECUTE IMMEDIATE 'DROP TABLE temp';
END;
/
```

Пример 17.2. В *разд. 8.3.2* рассматривался пример создания таблицы `К_меню`, содержащей калорийность и стоимость всех блюд, которые можно приготовить из имеющихся продуктов. Эта таблица будет использоваться шеф-поваром для составления меню на следующий день. Применим для создания таблицы `К_меню` и заполнения ее данными анонимный блок:

```
BEGIN
-- Создание временной таблицы К_меню
EXECUTE IMMEDIATE
'CREATE TABLE К_меню ' ||
 '( Код_вида NUMBER(2),'||
 '  Блюдо VARCHAR2(16),'||
 '  Калор_блюда NUMBER(4),'||
 '  Стоим_блюда NUMBER(4,2) )';
--
-- Загрузка актуальных данных в созданную таблицу
EXECUTE IMMEDIATE 'INSERT INTO К_меню'||
' SELECT Блюда.Код_вида, Блюдо,'||
    'ROUND(SUM(((Белки+Углев)*4.1+Жиры*9.3) * Вес/1000)) Колор_блюда, '||
    'ROUND((SUM(Стоимость/К_во*Вес/1000) + MIN(Труд/100))*10,2)
           Стоим_блюда '||
'FROM   Блюда, Виды_блюд, Состав, Продукты, Наличие '||
'WHERE  Блюда.Код_блюда     = Состав. Код_блюда '||
'AND    Состав.Код_продукта = Продукты.Код_продукта '||
'AND    Состав.Код_продукта = Наличие.Код_продукта '||
'AND    Блюда.Код_вида      = Виды_блюд.Код_вида '||
'AND    Блюда.Код_блюда NOT IN '||
```

```
'          (SELECT   Код_блюда '||
'           FROM  Состав '||
'           WHERE Код_продукта IN '||
'             (SELECT Код_продукта '||
'              FROM   Наличие '||
'              WHERE  К_во = 0)) '||
'GROUP BY Блюда.Код_вида, Блюдо '||
'ORDER BY Блюда.Код_вида, Колор_блюда';
END;
/
--
-- Анонимный блок для удаления таблицы К_меню после ее использования:
BEGIN
EXECUTE IMMEDIATE 'DROP TABLE К_меню';
END;
/
```

Пример 17.3. Используя сведения из представлений словаря данных, получить информацию обо всех таблицах текущей схемы пользователя, в которой бы содержалось имя таблицы, число столбцов и строк каждой из них.

В примере 14.2 был приведен запрос, позволяющий получить сведения о таблицах текущей схемы с указанием числа столбцов в каждой из них:

```
SELECT table_name, COUNT(column_name) Столбцов
  FROM USER_TAB_COLUMNS
  WHERE table_name NOT IN (SELECT view_name FROM USER_VIEWS)
  GROUP BY table_name;
```

Вроде бы достаточно вставить во фразу `SELECT` подзапрос, подсчитывающий число строк в таблице `table_name`:

```
SELECT table_name, COUNT(column_name) Столбцов,
(SELECT COUNT(*) FROM table_name) Строк  -- Подсчет количества строк
  FROM USER_TAB_COLUMNS
  WHERE table_name NOT IN (SELECT view_name FROM USER_VIEWS)
  GROUP BY table_name;
```

Однако в этом случае будет получено сообщение об ошибке:

```
ORA-00942: таблица или представление пользователя не существует
```

Действительно, на момент выполнения синтаксического разбора этого запроса, SQL не знает таблицы `table_name`. Имена таблиц текущей схемы будут определяться по мере вывода строк запроса.

Например, запрос:

```
SELECT table_name, COUNT(column_name) Столбцов,
(SELECT COUNT(*) FROM Блюда) Строк
  FROM USER_TAB_COLUMNS
  WHERE table_name NOT IN (SELECT view_name FROM USER_VIEWS)
  GROUP BY table_name;
```

где вместо

```
(SELECT COUNT(*) FROM table_name)
```

вставлен подзапрос с конкретной таблицей `Блюда`

```
(SELECT COUNT(*) FROM Блюда)
```

дает результат, содержащий список таблиц, числа из столбцов и количества строк таблицы `Блюда` для каждой из этих таблиц:

```
TABLE_NAME       СТОЛБЦОВ      СТРОК
-------------- ---------- ----------
БЛЮДА                   6         38
ВИДЫ_БЛЮД               2         38
ВЫБОР                   2         38
МЕНЮ                    4         38
ПОСТАВКИ                5         38
ПОСТАВЩИКИ              6         38
ПРОДУКТЫ               11         38
РЕЦЕПТЫ                 4         38
СОСТАВ                  3         38
ТРАПЕЗЫ                 2         38
```

Поэтому для решения поставленной задачи, придется создать PL/SQL-блок, в котором при помощи курсора будут последовательно выводиться строки запроса

```
SELECT table_name, COUNT(column_name) Столбцов
  FROM USER_TAB_COLUMNS
  WHERE table_name NOT IN (SELECT view_name FROM USER_VIEWS)
  GROUP BY table_name;
```

и каждая из них будет обрабатываться динамическим SQL для определения числа строк той таблицы, сведения о которой выводятся в текущей строке курсора.

```
SET SERVEROUTPUT ON
DECLARE
  /* курсор для выбора информации о таблицах схемы */
  CURSOR rec_tab IS SELECT table_name, COUNT(column_name) Столбцов
  FROM USER_TAB_COLUMNS
```

```
  WHERE table_name NOT IN (SELECT view_name FROM USER_VIEWS)
  GROUP BY table_name;
  STROK NUMBER(6); /* Переменная для подсчета количества строк. */
BEGIN
  -- Вывод на экран "Шапки" таблицы. Здесь функция RPAD дополняет
  -- выводимое значение пробелами до заданной длины.
  DBMS_OUTPUT.PUT_LINE
     (RPAD('Имя таблицы',25)||RPAD('Столбцов',10)||RPAD('Строк',10));
  DBMS_OUTPUT.PUT_LINE('----------------------------------------');
  FOR ind_tab IN rec_tab LOOP
  EXECUTE IMMEDIATE 'SELECT COUNT(*) FROM '||ind_tab.table_name
                                            INTO STROK;
    -- Вывод на экран полей записи
    DBMS_OUTPUT.PUT_LINE
    (rpad(ind_tab.table_name,30)||' '||rpad(ind_tab.Столбцов,6)||STROK);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('----------------------------------------');
END;
/
```

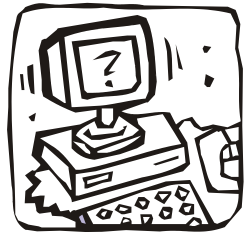
При запуске этого блока в схеме COOK, будет получен следующий результат:

```
Имя таблицы              Столбцов  Строк
----------------------------------------
БЛЮДА                          6     38
ВИДЫ_БЛЮД                      2     5
ВЫБОР                          2     320
МЕНЮ                           4     21
ПОСТАВКИ                       5     25
ПОСТАВЩИКИ                     6     8
ПРОДУКТЫ                       11    20
РЕЦЕПТЫ                        4     34
СОСТАВ                         3     148
ТРАПЕЗЫ                        2     3
----------------------------------------
PL/SQL procedure successfully completed
```

Запуск блока в схеме UCHEB даст такой результат:

```
Имя таблицы              Столбцов  Строк
----------------------------------------
Н_ВЕДОМОСТИ                    16    48944
Н_ВЕДОМОСТИ_НОМЕРА             11    0
Н_ВИДЫ_ОБУЧЕНИЯ                8     6
```

```
Н_ВИДЫ_РАБОТ                    9     45
Н_ГРУППЫ_ПЛАНОВ                 6     5040
Н_ДИСЦИПЛИНЫ                    7     1756
Н_ИЗМ_ЛЮДИ                      13    4
Н_ИЗМ_ОТДЕЛЫ                    14    0
Н_КВАЛИФИКАЦИИ                  6     22
Н_КОМПОНЕНТЫ                    7     7
Н_ЛЮДИ                          15    2067
Н_НАПР_СПЕЦ                     7     277
Н_НАПРАВЛЕНИЯ_СПЕЦИАЛ           12    377
Н_ОБУЧЕНИЯ                      7     903
Н_ОТДЕЛЫ                        15    227
Н_ОЦЕНКИ                        3     12
Н_ПЛАНЫ                         23    3915
Н_СВОЙСТВА_ВР                   7     9
Н_СВОЙСТВА_ОТДЕЛОВ              7     14
Н_СЕССИЯ                        16    12498
Н_СОДЕРЖАНИЯ_ЭЛЕМЕНТОВ_СТРОК    9     203871
Н_СТРОКИ_ПЛАНОВ                 11    46912
Н_ТИПЫ_ВЕДОМОСТЕЙ               6     3
Н_ТИПЫ_ПЛАНОВ                   7     5
Н_ТИПЫ_СТАНДАРТОВ               6     8
Н_УЧЕБНЫЕ_ГОДА                  3     13
Н_УЧЕНИКИ                       19    4267
Н_ФОРМЫ_ОБУЧЕНИЯ                12    4
Н_ХАРАКТЕРИСТИКИ_ВИДОВ_РАБОТ    6     41
Н_ХАРАКТЕРИСТИКИ_ОТДЕЛОВ        8     164
Н_ЦИКЛЫ_ДИСЦИПЛИН               7     14
Н_ЭКЗ_ЛИСТЫ_НОМЕРА              13    0
Н_ЭЛЕМЕНТЫ_СТРОК                9     59079
-----------------------------------------
PL/SQL procedure successfully completed
```



Глава 18

Хранимые процедуры

18.1. Введение

Напомним, что PL/SQL — язык Oracle четвертого поколения, объединяющий структурированные элементы процедурного языка программирования с языком SQL, разработанный специально для организации вычислений в среде клиент-сервер. Он позволяет передать на сервер программный блок PL/SQL, содержащий логику приложения, как оператор SQL, одним запросом. Используя PL/SQL, можно значительно уменьшить объем обработки в клиентской части приложения и нагрузку на сеть. Например, может понадобиться выполнить различные наборы операторов SQL в зависимости от результата некоторого запроса. Запрос, последующие операторы SQL и операторы условного управления могут быть включены в один блок PL/SQL и пересланы серверу за одно обращение к сети.

При этом вся логика приложений делится на клиентскую и серверную части. Серверная часть может быть реализована в виде функций, хранимых процедур, триггеров и пакетов.

Функции — часть логики приложения, ориентированной на выполнение конкретного комплекса операций на сервере, результат которых возвращается в виде значения функции. Откомпилированные функции и их исходные тексты содержатся в базе данных.

Хранимые процедуры — часть логики приложения, особенно нуждающаяся в доступе к базе данных, может храниться там, где она обрабатывается (на сервере). Хранимые процедуры не возвращают значения результата, обеспечивают удобный и эффективный механизм безопасности. Откомпилированные хранимые процедуры и их исходные тексты содержатся в базе данных.

Пакеты — часть логики приложения: функций и процедур, предназначенных для решения задач в рамках одной подсистемы автоматизированной информационной системы.

Триггеры базы данных. Можно использовать триггеры, чтобы организовать сложный контроль целостности, выполнять протоколирование (аудит) и другие функции безопасности, реализовать в приложениях выдачу предупреждений и мониторинг.

Декларативная целостность. Ограничения активизируются сервером всякий раз, когда записи вставляются, обновляются или удаляются. В дополнение к ограничениям ссылочной целостности, которые проверяют соответствие первичного и внешнего ключей, можно также накладывать ограничения на значения, содержащиеся в столбцах таблицы. Поддержка целостности на сервере уменьшает размер кода клиентской части, необходимого для проверки допустимости данных, и увеличивает устойчивость бизнес-модели, определенной в базе данных.

18.2. Хранимые процедуры

Недостатком выполнения сценариев анонимных процедур (см. *разд. 17.2.1*) в распределенной вычислительной среде является то, что необходимо хранить актуальную версию этих сценариев на большом числе компьютеров. Программное обеспечение намного проще хранить в базе данных (в одном месте), чтобы к нему мог обратиться каждый. Для этого используются хранимые процедуры (функции и пакеты).

18.2.1. Создание описания процедуры

Предложение для создания (`CREATE`) или замены (`CREATE OR REPLACE`) независимой (не входящей в состав пакета) процедуры имеет вид:

```
CREATE [OR REPLACE] PROCEDURE [схема.]имя_процедуры
[(параметр1 [IN | OUT | IN OUT] [NOCOPY] тип_данных [,...] ) ]
{IS | AS}
{блок_кода | LANGUAGE
{JAVA NAME имя_внешней_программы | C [NAME имя_внешней_программы]
LIBRARY имя_библиотеки [AGENT IN (аргумент [,...]) [WITH CONTEXT]]
[PARAMETERS (параметр [,...]) ] };
```

Создается новая пользовательская процедура. Здесь:

`OR REPLACE`

предназначено для замены существующей процедуры без предварительного ее удаления и последующего переназначения всех связанных с ней объектов.

`[IN | OUT | IN OUT]`

Определяет, является ли параметр для процедуры входным, выходным или и тем и другим одновременно.

`NOCOPY`

Этот параметр увеличивает производительность, если аргумент типа `OUT` или `IN OUT` очень велик (например, относится к типу `VARRAY` или `RECORD`).

`IS | AS`

Ключевые слова `IS | AS` в Oracle идентичны. Любое из них начинает *`блок_кода`*.

`блок_кода`

Тело (текст) процедуры на языке PL/SQL.

`LANGUAGE`

В качестве альтернативы (для хранимых процедур на Java и C) можно использовать предложение `LANGUAGE`, в котором определяется реализация внешней программы на Java или C. Параметры и семантика объявлений специфичны для Java и C, а не для SQL.

18.2.2. Удаление описания процедуры

Для удаления описания процедуры используется предложение:

`DROP PROCEDURE` *`[схема.]имя_процедуры`*

При удалении хранимой процедуры неработоспособными становятся все зависимые объекты. Зависимые объекты будут возвращать ошибку при обращении к ним до тех пор, пока не будет создана хранимая процедура. Если хранимая процедура воссоздана и осуществляется обращение к зависимому объекту, зависимый объект будет перекомпилирован.

18.2.3. Перекомпиляция процедуры

Для перекомпиляции процедуры используется предложение:

`ALTER PROCEDURE` *`[схема.]имя_процедуры`* `COMPILE;`

18.2.4. Пример создания процедуры

Создадим хранимую процедуру, выполняющую те же действия, что и анонимный блок (см. *разд. 17.2.1*). Текст ее тела отличается от текста тела анонимного блока лишь отсутствием обращения к утилите `DBMS_OUTPUT.PUT_LINE`, используемой для вывода результата:

```
CREATE OR REPLACE PROCEDURE
   p_fio                    -- имя процедуры
   (tekst VARCHAR2,         -- входной параметр (опущен умалчиваемый IN)
    result OUT VARCHAR2)    -- параметр с OUT позволяет возвращать
      -- обновленное значение подпрограмме, вызывающей данную процедуру
    IS
-- описание локальных переменных, используемых в теле процедуры
  kol    INTEGER; /* Количество символов в проверяемом тексте */
BEGIN /* исполняемая часть (тело) процедуры*/
  result := LOWER(RTRIM(LTRIM(tekst)));
     /* первый вариант результата, полученный после удаления из входного
        текста начальных (LTRIM) и конечных (RTRIM) пробелов, а также
        преобразования всех букв текста в строчные (LOWER) */
  kol := LENGTH(result);
     /* определение числа символов, оставшихся в тексте после удаления
        начальных и конечных пробелов */
  IF kol > 0 THEN
     /* если текст содержит символы, то его дальнейшее преобразование и
        проверка на наличие запрещенных символов */
    result := REPLACE(result,'  ',' '); -- замена двух пробелов на один
    result := REPLACE(result,'- ','-'); -- замена тире и пробела на тире
    result := REPLACE(result,' -','-'); -- замена пробела и тире на тире
    kol := LENGTH(result); -- определение числа оставшихся символов
    FOR i IN 1..kol LOOP /* перебор всех символов текста */
     IF INSTR('- абвгдеёжзийклмнопрстуфхцчшщъыьэюя',SUBSTR(result,i,1))=0
        THEN
          /* с помощью функции INSTR определяется позиция первого
             включения i-го символа текста (вырезанного с помощью функции
             SUBSTR) в набор '-абвгдеёжзийклмнопрстуфхцчшщъыьэюя', и если
             эта позиция равна 0 (i-го символа нет в наборе), то
             производится: */
        result := '0';  -- установка нулевого результата и
        EXIT; -- выход из цикла
```

```
     END IF;
    END LOOP;
  ELSE
    result := '0';  -- установка нулевого результата при отсутствии
                    -- в тексте символов, отличных от пробелов
  END IF;
  IF result <> '0' THEN -- если текст содержит символы, то:
    result := INITCAP(result);
              /* преобразование первых букв слов текста в заглавные */
  END IF;
END p_fio;
/
```

Так же как и текст анонимного блока, текст команды по созданию процедуры можно ввести с помощью текстового редактора в файл (например, с именем p_fio.sql) и выполнить в SQL*Plus команду `@p_fio.sql` или вводить строчка за строчкой прямо в SQL*Plus.

После создания процедуры `p_fio` ее можно вызвать из другой процедуры или анонимного блока:

```
DECLARE
  vchod  VARCHAR2(20) := 'Жан - жак  руссо';
  rezult VARCHAR2(20);
BEGIN
  p_fio (vchod, rezult);
  dbms_output.put_line(rezult);
  /* использование встроенного пакета DBMS_OUTPUT для вывода на экран
     результата преобразования; если ранее не выполнялась установка
     SET SERVEROUTPUT ON, разрешающая вывод на экран информации, заданной
     в DBMS_OUTPUT, то ее надо выполнить до выполнения блока */
END;
/
```

При обращении к процедуре в качестве фактических параметров можно использовать любые выражения и, естественно, их составляющие, например, константы:

```
p_fio ('Жан - жак  руссо', rezult);
```

или

```
p_fio ('Жан'||' - жак'||'  руссо', rezult);
```

В том и другом случае на экран будет выведен текст:

```
Жан-Жак Руссо
```

18.3. Функции

Функция PL/SQL отличается от процедуры тем, что возвращаемое значение расположено в ее имени так же, как и в стандартных функциях (например, SIN, ABS, SUBSTR).

18.3.1. Создание описания функции

Предложение для создания функции имеет следующий синтаксис:

```
CREATE [OR REPLACE] FUNCTION [схема.]имя_функции
[(параметр1 [IN | OUT | IN OUT] [NOCOPY] тип_данных [,...] ) ]
RETURN тип_данных
{IS | AS}
{блок_кода | LANGUAGE
{JAVA NAME имя_внешней_программы | C [NAME имя_внешней_программы]
LIBRARY имя_библиотеки [AGENT IN (аргумент [,...]) [WITH CONTEXT]]
[PARAMETERS (параметр [,...]) ] };
```

Создается новая пользовательская функция. Предложение OR REPLACE предназначено для замены существующей функции без предварительного ее удаления и последующего переназначения всех связанных с ней объектов.

[IN | OUT | IN OUT]

Определяет, является ли параметр для функции входным, выходным или и тем и другим одновременно.

NOCOPY

Этот параметр увеличивает производительность, если аргумент типа OUT или IN OUT очень велик (например, относится к типу VARRAY или RECORD).

RETURN *тип_данных*

Используется для описания типа данных значения, которое будет размещено в имени функции.

IS | AS

Ключевые слова IS | AS в Oracle идентичны. Любое из них начинает *блок_кода*.

блок_кода

Тело (текст) функции на языке PL/SQL.

LANGUAGE

В качестве альтернативы (для функций на Java и C) можно использовать предложение LANGUAGE, в котором определяется реализация внешней про-

граммы на Java или C. Параметры и семантика объявлений специфичны для Java и C, а не для SQL.

18.3.2. Удаление описания функции

Для удаления описания функции используется команда:

```
DROP FUNCTION [схема.]имя_функции;
```

18.3.3. Перекомпиляция функции

Для перекомпиляции функции используется команда:

```
ALTER FUNCTION [схема.]имя_функции COMPILE ;
```

18.3.4. Пример создания функции

Создадим функцию, выполняющую те же действия, что и процедура (см. *разд. 18.2*), текст ее тела отличается от текста тела процедуры наличием описания возвращаемого значения (`RETURN VARCHAR2`) и предложением `RETURN result`, завершающим выполнение функции и присваивающим значение ее имени:

```
CREATE OR REPLACE FUNCTION
    fio                 -- имя функции
    (tekst VARCHAR2)    -- параметр (текст фамилии, имени или отчества)
                        -- и тип данных этого текста
    RETURN VARCHAR2     -- тип возвращаемого значения
    IS
-- описание локальных переменных, используемых в теле функции
  result  VARCHAR2(20);
  kol     INTEGER; /* Количество символов в проверяемом тексте */
BEGIN -- начало тела функции
  result := LOWER(RTRIM(LTRIM(tekst)));
     /* первый вариант результата, полученный после удаления из входного
        текста начальных (LTRIM) и конечных (RTRIM) пробелов, а также
        преобразования всех букв текста в строчные (LOWER) */
  kol := LENGTH(result);
     /* определение числа символов, оставшихся в тексте после удаления
        начальных и конечных пробелов */
```

```
  IF kol > 0 THEN
     /* если текст содержит символы, то его дальнейшее преобразование и
        проверка на наличие запрещенных символов */
    result := REPLACE(result,'  ',' '); -- замена двух пробелов на один
    result := REPLACE(result,'- ','-'); -- замена тире и пробела на тире
    result := REPLACE(result,' -','-'); -- замена пробела и тире на тире
    kol := LENGTH(result); -- определение числа оставшихся символов
    FOR i IN 1..kol LOOP /* перебор всех символов текста */
     IF INSTR('- абвгдеёжзийклмнопрстуфхцчшщъыьэюя',SUBSTR(result,i,1))=0
        THEN
          /* с помощью функции INSTR определяется позиция первого
             включения i-го символа текста (вырезанного с помощью функции
             SUBSTR) в набор '-абвгдеёжзийклмнопрстуфхцчшщъыьэюя', и если
             эта позиция равна 0 (i-го символа нет в наборе), то
             производится: */
        result := '0';  -- установка нулевого результата и
        EXIT; -- выход из цикла
     END IF;
    END LOOP;
  ELSE
    result := '0';  -- установка нулевого результата при отсутствии
                    -- в тексте символов, отличных от пробелов
  END IF;
  IF result <> '0' THEN -- если текст содержит символы, то:
    result := INITCAP(result);
              /* преобразование первых букв слов текста в заглавные */
  END IF;
    RETURN result;
END fio;
/
```

Текст команды по созданию функции вводится в Oracle так же, как и текст процедуры. Вызов функции PL/SQL можно осуществить в тех же местах, где и вызов стандартной функции. Например, в списке фразы `SELECT`:

```
SELECT fio('Жан - жак  руссо') Фамилия FROM DUAL;
```

Результат ее выполнения имеет вид:

```
ФАМИЛИЯ
--------------------
Жан-Жак Руссо
```

Здесь использовалась специальная однострочная таблица DUAL, которая создается Oracle для каждой схемы и обычно используется для вывода значения каких-либо выражений.

18.4. Триггеры

Триггер — это сочетание хранимой в базе данных процедуры и события, которое заставляет ее выполняться. Такими событиями могут быть: ввод новой строки таблицы, изменение значений одного или нескольких ее столбцов и (или) удаление строки таблицы. При любом из этих событий автоматически запускаются один или несколько заранее созданных триггеров, которые производят проверку запрограммированных в них условий, и если они не выполняются, отменяют ввод, изменение или удаление, посылая об этом заранее подготовленное сообщение пользователю.

Триггеры похожи на процедуры и функции тем, что также являются именованными блоками PL/SQL и имеют раздел объявлений, выполняемый раздел и раздел обработки исключительных ситуаций. Подобно процедурам и функциям, триггеры хранятся как автономные объекты в базе данных.

Триггеры позволяют:

- реализовывать сложные ограничения целостности данных, которые невозможно реализовать через ограничения, устанавливаемые при создании таблицы;
- контролировать информацию, хранимую в таблице, посредством регистрации вносимых изменений и пользователей, производящих эти изменения;
- автоматически оповещать другие программы о том, что необходимо делать в случае изменения информации, содержащейся в таблице;
- публиковать информацию о различных событиях.

Триггеры также делятся на три основных типа.

- *Триггеры DML* активизируются предложениями ввода, обновления и удаления информации (INSERT, UPDATE, DELETE) до или после выполнения предложения, на уровне строки или таблицы.
- *Триггеры замещения* (instead of) можно создавать только для представлений (либо объектных, либо реляционных). В отличие от триггеров DML, которые выполняются в дополнение к предложениям DML, триггеры замещения выполняются вместо предложений DML, вызывающих их срабатывание. Триггеры замещения должны быть строковыми триггерами.

- *Системные триггеры* активизируется не на предложение DML, выполняемое над таблицей, а на системное событие, например, на запуск или останов базы данных. Системные триггеры срабатывают и на предложения DDL, такие как создание таблицы.

Триггеры также подразделяются на *строковые* (строчные или уровня строки таблицы) и *табличные* (операторные или уровня всей таблицы). Строковый триггер запускается один раз для каждой строки, обрабатываемой активизирующим предложением. Табличный триггер выполняется только один раз, независимо от того, сколько строк содержит запускающее его предложение. Выбор того или иного типа триггера зависит от его логики.

Внутри строкового триггера можно обращаться к данным обрабатываемой строки. Для этого служат два идентификатора корреляции — `:NEW` и `:OLD`. *Идентификатор корреляции* (correlation identifier) — это переменная привязки PL/SQL особого рода. Двоеточие перед идентификатором указывает на то, что это переменные привязки (подобны базовым переменным, используемым во встроенном PL/SQL), а не обычные переменные PL/SQL.

18.4.1. Создание описания триггера

Предложение для создания (`CREATE`) или замены (`CREATE OR REPLACE`) триггера имеет вид:

```
CREATE [OR REPLACE] TRIGGER имя_триггера {BEFORE | AFTER | INSTEAD OF}
{ {[событие_объекта] [событие_базы_даннах] [...] ON {DATABASE |
схема.SCHEMA}} |
{[DELETE] [OR] [INSERT] [0R] [UPDATE [OF столбец [,...] ]] [,...]}
ON {имя_таблицы | [NESTED TABLE имя_столбца OF] имя_представления}
[REFERENCING {[OLD [AS] имя_для_старых] [NEW [AS] имя_для_новых]}
[FOR EACH ROW] }
[WHEN (условия)]
блок_кода;
```

Здесь:

`CREATE [OR REPLACE] TRIGGER` *имя_триггера*

Создается новый с именем *имя_триггера* или пересоздается (`OR REPLACE`) существующий триггер, которому присваиваются новые определения.

`{BEFORE | AFTER | INSTEAD OF}`

Указывается, что логика триггера запускается до (`BEFORE`) или после (`AFTER`) выполнения предложения по манипуляции данными. Триггеры с фразой `BEFORE` запускаются до выполнения операций вставки, обновления или уда-

ления, и это позволяет даже вообще обходить операции по манипуляции. Триггеры с предложением AFTER запускаются после завершения операции по манипуляции, и они полезны для операций, совершаемых постфактум (например, при пересчете промежуточных сумм). Условие активизации INSTEAD OF (вместо) позволяет *изменять представления* путем изменения базовых таблиц, входящих в его состав.

событие_объекта

В качестве дополнения к стандартным событиям, связанным с модификацией данных, Oracle позволяет запускать триггеры в зависимости от событий объектов. Операции с такими событиями могут использоваться с ключевыми словами BEFORE и AFTER. Событие объекта запускает триггер при возникновении данного события, при этом используются ключевые слова: ALTER, ANALYZE, ASSOCIATE STATISTICS, AUDIT, COMMENT, DDL, DISASSOCIATE STATISTICS, DROP, GRANT, NOAUDIT, RENAME, REVOKE, TRUNCATE.

событие_базы_данных

Помимо стандартных событий, связанных с модификацией данных, Oracle позволяет запускать триггеры на основе событий базы данных. Такие события можно использовать с ключевыми словами BEFORE и AFTER. Список доступных ключевых слов для предложения *событие_базы_данных*: LOGON, LOGOFF, SERVERERROR, SHUTDOWN, STARTUP, SUSPEND.

ON {DATABASE | *схема*.SCHEMA}

Объявляется, что триггер запускается, если любой пользователь базы данных вызвал событие, запускающее триггер, при наличии предложения ON DATABASE. Затем триггер запускает события, возникающие в любом месте базы данных. При указании предложения ON *схема*.SCHEMA триггер запускается, если пользователь, вызвавший запускающее событие, входил в систему под схемой с именем *схема*. Затем триггер запускают события, возникающие в любом месте текущей схемы.

{[DELETE] [OR] [INSERT] [0R] [UPDATE [OF *столбец* [,...]]] [,...]}

Определяется предложение по манипуляции данными, которое запускает триггер: DELETE, INSERT, UPDATE или любые их сочетания, объединяемые предикатом OR. С помощью фразы UPDATE OF *столбец* [,.,.] можно указать, какие столбцы будут запускать триггер обновления. Если обновление будет касаться столбцов, не входящих в список, триггер запускаться не будет.

ON {*имя_таблицы* | [NESTED TABLE *имя_столбца* OF] *имя_представления*}

Указывается таблица с именем *имя_таблицы*, от которой зависит триггер или указывается, что триггер запускается только в том случае, когда операция по манипулированию данными применяется к столбцу (столбцам) представления

с именем *имя_представления*. Предложение ON NESTED TABLE совместимо только триггерами INSTEAD OF.

REFERENCING {[OLD [AS] *имя_для_старых*] [NEW [AS] *имя_для_новых*]}

Используется для изменения заданных по умолчанию префиксов NEW и OLD, которые указываются перед именами столбцов, новые или старые значения которых должны использоваться в тексте тела строчного триггера. При выполнении предложения INSERT во вводимой строке существуют только новые значения. При выполнении предложения DELETE — только старые значения, а при выполнении UPDATE — и те и другие. REFERENCING используется тогда, когда имя столбца должно уточняться именем таблицы, которую зачем-то назвали NEW или OLD, и, следовательно, возникает неоднозначность в описании столбца.

[FOR EACH ROW]

Этот параметр устанавливается для триггера *строчного типа* (который должен запускаться в процессе ввода (удаления, изменения) каждой строки таблицы; при отсутствии FOR EACH ROW создается триггер *табличного типа*, запускаемый только один раз, независимо от того, сколько строк должно быть введено (удалено или изменено) при выполнении заданного SQL-предложения.

[WHEN (*условие*)]

Используется для отбора тех вводимых, удаляемых или изменяемых строк, которые должны вызвать запуск строчного триггера (по умолчанию строчный триггер запускается при вводе, удалении или изменении каждой строки соответствующей таблицы).

блок_кода

Тело (текст) триггера на языке PL/SQL.

Последовательность выполнения нескольких триггеров одной таблицы

Активация триггеров DML происходит при выполнении предложений INSERT, UPDATE или DELETE. Для каждого из этих предложений может быть создано четыре триггера: табличный BEFORE, табличный AFTER, строковый BEFORE и строковый AFTER. При наличии всех этих триггеров порядок их активации таков:

1. Табличный триггер BEFORE.
2. Строковый триггер BEFORE.
3. Строковый триггер AFTER.
4. Табличный триггер AFTER.

Предикаты *INSERTING*, *UPDATING* и *DELETING*

В одном описании можно объединить несколько триггеров, если они относятся к одному уровню (строка или предложение) и принадлежат к одной таблице. Если триггеры объединяются в одно описание, то для разделения логики на отдельные сегменты можно использовать в блоке кода PL/SQL три *логические функции (предикаты)*, определяющие тип выполняемой операции.

- `INSERTING` — принимает значение `TRUE`, если активизирующий оператор `INSERT`; `FALSE` в противном случае.
- `UPDATING` — принимает значение `TRUE`, если активизирующий оператор `UPDATE`; `FALSE` в противном случае.
- `DELETING` — принимает значение `TRUE`, если активизирующий оператор `DELETE`; `FALSE` в противном случае.

18.4.2. Изменение описания триггера

Синтаксис инструкции `ALTER TRIGGER`, которая позволяет переименовывать, включать или отключать триггер без его удаления и повторного создания, имеет вид:

```
ALTER TRIGGER имя_триггера
{ {ENABLE | DISABLE} | RENAME TO новое_имя |
COMPILE [директивы_компилятора] [DEBUG] [REUSE SETTINGS]}
```

Здесь:

`ENABLE`

Включает ранее деактивированный триггер. В качестве альтернативы можно использовать предложение `ALTER TABLE имя_таблицы ENABLE ALL TRIGGERS`.

`DISABLE`

Отключает работающий триггер. В качестве альтернативы можно использовать предложение `ALTER TABLE имя_таблицы DISABLE ALL TRIGGERS`.

`RENAME TO новое_имя`

Триггер переименовывается в *новое_имя*, но его состояние остается неизменным.

`COMPILE [DEBUG] [REUSE SETTINGS]`

Триггер компилируется независимо от того, является он допустимым или нет, а также компилируются все объекты, от которых он зависит. Если любой из объектов неработоспособен, триггер будет неработоспособным. Если все объекты работоспособны, в том числе *блок_кода* триггера, то триггер стано-

вится работоспособным. У предложения COMPILE есть несколько дополнительных предложений.

DEBUG

Компилятор PL/SQL будет генерировать и записывать дополнительную информацию, которую сможет использовать отладчик PL/SQL.

REUSE SETTINGS

Система Oracle сохранит все настройки компилятора, что поможет сэкономить много времени при компиляции.

директивы_компилятора

Указывается специальное значение, используемое компилятором PL/SQL в формате *директива='значение'*. Существуют следующие директивы: PLSQL_OPTIMIZE_LEVEL, PLSQL_CODE_TYPE, PLSQL_DEBUG, PLSQL_WARNINGS и PLSQL_NLSLENGTHSEMANTICS. Для каждой из них в инструкции можно указать значение один раз. Директива действует только в рамках компилируемого модуля.

Примеры создания триггеров были рассмотрены в *разд. 12.3* (листинги 12.1—12.7). В *разд. 18.4.4* будут рассмотрены более сложные примеры.

18.4.3. Удаление описания триггера

```
DROP TRIGGER имя_триггера;
```

18.4.4. Использование триггера

Приводимые далее примеры созданы для объектов базы данных "UCHEB", подробно рассматриваемой в *части VII*.

Пример 18.1. Создадим триггер для формирования номера нового человека в момент ввода данных о нем в таблицу Н_ЛЮДИ, а также для исправления некорректных значений его фамилии и имени с помощью рассмотренной ранее функции fio (см. *разд. 18.3.4*). Этот триггер запускается не только при выполнении вставки новых строк (INSERT) в таблицу Н_ЛЮДИ, но и при изменении ее строк (UPDATE). Для того чтобы он создавал номер человека только при вставке строк, используется логическая функция INSERTING.

```
CREATE OR REPLACE TRIGGER люди_biur
BEFORE INSERT OR UPDATE ON н_люди
FOR EACH ROW
DECLARE
```

```
  err_fam     EXCEPTION;
  err_im      EXCEPTION;
BEGIN
  :new.Фамилия := fio(:new.Фамилия); -- фамилия замещается результатом
                                     -- работы функции fio
  :new.Имя := fio(:new.Имя); -- имя замещается результатом работы
                             -- функции fio
  IF :new.Фамилия = '0' THEN RAISE err_fam;
  END IF; -- выход по ошибке при неправильном написании фамилии
  IF :new.Имя     = '0' THEN RAISE err_im;
  END IF;  -- выход по ошибке при неправильном написании имени
IF INSERTING THEN
  -- Формирование нового номера человека
   SELECT н_люди_посл.NEXTVAL INTO :new.ид FROM dual;
END IF;
EXCEPTION     -- начало обработчика исключений основной программы
  WHEN err_fam THEN RAISE_APPLICATION_ERROR(-20040,
    'Фамилия должна состоять только из букв русского алфавита, '||
    'пробела, дефиса и начинаться с заглавной буквы !');
  WHEN err_im THEN RAISE_APPLICATION_ERROR(-20041,
    'Имя должно состоять только из букв русского алфавита, '||
    'пробела, дефиса и начинаться с заглавной буквы !');
  WHEN OTHERS THEN RAISE_APPLICATION_ERROR(-20999,
    'Какая-то другая ошибка');
END люди_biur;
/
```

Пример 18.2. Создадим триггер для формирования номера строки при ее вводе в таблицу `Н_УЧЕНИКИ` и для проверки правильности ввода значений некоторых ее столбцов.

```
CREATE OR REPLACE TRIGGER учен_BIR
  BEFORE INSERT ON н_ученики
  FOR EACH ROW
DECLARE
plan_gr    EXCEPTION; -- План должен быть связан с группой
nach_kon   EXCEPTION; -- Начало должно быть меньше или равно концу
null_val   EXCEPTION; -- какое-то из обязательных значений не заполнено
test       NUMBER;
BEGIN
  -- Если не введены какие-то обязательные значения, выводим сообщение
```

```
  IF :new.члвк_ид IS NULL
    OR :new.план_ид IS NULL OR :new.группа IS NULL
    OR :new.начало IS NULL OR :new.конец IS NULL THEN
    RAISE null_val;
  END IF;
  -- Проверяем, и если такой строки нет, то вставляем ее в н_обучения
  SELECT COUNT(*) INTO test FROM н_обучения
  WHERE члвк_ид = :NEW.члвк_ид AND вид_обуч_ид = :NEW.вид_обуч_ид;
  IF test = 0 THEN
    INSERT INTO н_обучения(члвк_ид,вид_обуч_ид) VALUES
(:NEW.члвк_ид,:NEW.вид_обуч_ид);
    COMMIT;
  END IF;
  -- генерируем ид
  SELECT н_учен_посл.NEXTVAL INTO :new.ид FROM dual;
  SELECT COUNT(*) INTO test FROM н_группы_планов WHERE группа =
:new.группа AND план_ид = :new.план_ид;
  -- Если план не связан с группой
  IF test = 0 THEN
    RAISE plan_gr;
  END IF;
  IF :new.начало > :new.конец THEN
    RAISE nach_kon;
  END IF;
  :new.конец_по_приказу := :new.конец;
EXCEPTION
  WHEN plan_gr THEN RAISE_APPLICATION_ERROR(-20040,'Номер группы не
сопоставлен с номером плана !');
  WHEN nach_kon THEN RAISE_APPLICATION_ERROR(-20041,'Конец не может быть
меньше начала !');
  WHEN null_val THEN RAISE_APPLICATION_ERROR(-20042,'Не заполнены
обязательные поля (информация о человеке, группа/план, даты) !');
  WHEN OTHERS THEN RAISE_APPLICATION_ERROR(-20999,
    'Какая-то другая ошибка');
END учен_bir;
/
```

Пример 18.3. Создадим триггер для изменения ряда значений таблицы Н_УЧЕНИКИ.

```
CREATE OR REPLACE TRIGGER учен_bur
  BEFORE UPDATE OF члвк_ид, начало, конец, план_ид, группа
                ON н_ученики
  FOR EACH ROW
```

```
DECLARE
plan_gr    EXCEPTION; -- План должен быть связан с группой
nach_kon   EXCEPTION; -- Начало должно быть меньше или равно концу
null_val   EXCEPTION; -- какое-то из обязательных значений не заполнено
test       NUMBER;
BEGIN
  -- Если не введены какие-то обязательные значения, выводим сообщение
  IF :new.члвк_ид IS NULL
    OR :new.план_ид IS NULL OR :new.группа IS NULL
    OR :new.начало IS NULL OR :new.конец IS NULL THEN
    RAISE null_val;
  END IF;
  --
  SELECT COUNT(*) INTO test FROM н_группы_планов WHERE группа =
:new.группа AND план_ид = :new.план_ид;
  -- Если план не связан с группой
  IF test = 0 THEN
    RAISE plan_gr;
  END IF;
  IF :new.начало > :new.конец THEN
      RAISE nach_kon;
  END IF;
  -- Если обновляется конец, то мы старый его вариант сохраняем
  -- в поле конец_по_приказу
  IF :old.конец <> :new.конец AND :new.конец <> :new.конец_по_приказу
THEN
    :new.конец_по_приказу := :old.конец;
   END IF;
EXCEPTION
  WHEN plan_gr THEN RAISE_APPLICATION_ERROR(-20040,'Номер группы не
сопоставлен с номером плана ! (ид '||:NEW.ид||')');
  WHEN nach_kon THEN  RAISE_APPLICATION_ERROR(-20041,'Конец не может быть
меньше начала !');
  WHEN null_val THEN  RAISE_APPLICATION_ERROR(-20042,'Не заполнены обяза-
тельные поля (информация о человеке, группа/план, даты) !');
  WHEN OTHERS THEN RAISE_APPLICATION_ERROR(-20999,'Какая-то другая
ошибка');
END учен_bur;
/
```

18.4.5. Изменяющиеся (мутирующие) таблицы

Из тела строкового триггера нельзя обращаться к таблицам, которые в момент обращения модифицируются предложениями INSERT, DELETE или UPDATE, так как это может привести к неверным результатам. Такие таблицы получили названия изменяющихся (мутирующих) таблиц.

Правда, если используется предложение:

```
INSERT INTO имя_таблицы  ... VALUES (скалярное_выражение [,...] );
```

для вставки одного значения (или по одному значению) в таблицу *имя_таблицы*, то для строковых триггеров BEFORE и AFTER эта таблица не является изменяющейся. И это единственная ситуация, когда строковый триггер может считывать и модифицировать информацию активизирующей таблицы.

Однако при использовании предложения:

```
INSERT INTO имя_таблицы  ... предложение_SELECT;
```

таблица *имя_таблицы* всегда является изменяющейся, несмотря на то, сколько строк возвращается в подзапросе.

Рассмотрим достаточно объемный пример, в котором появится мутирующая таблица и ошибки, возникающие при ее появлении, и можно будет подробно обсудить, как избежать этих ошибок.

1. Создадим сначала в схеме COOK таблицу Должности, в которой будут храниться сведения о должностях и должностных окладах сотрудников пансионата:

```
CREATE TABLE ДОЛЖНОСТИ
(
  ИД           NUMBER(9) PRIMARY KEY,
  ДОЛЖНОСТЬ    VARCHAR2(25) NOT NULL,
  ОКЛАД        NUMBER(15,2) NOT NULL,
  НАЧАЛО       DATE NOT NULL,
  КОНЕЦ        DATE NOT NULL,
  СОСТОЯНИЕ    VARCHAR2(10) DEFAULT 'Проект' NOT NULL
);

COMMENT ON COLUMN ДОЛЖНОСТИ.ИД
  IS 'Уникальный идентификатор';
COMMENT ON COLUMN ДОЛЖНОСТИ.ДОЛЖНОСТЬ
  IS 'Наименование должности';
```

```
COMMENT ON COLUMN ДОЛЖНОСТИ.ОКЛАД
  IS 'Значение оплаты в рублях по должности';
COMMENT ON COLUMN ДОЛЖНОСТИ.НАЧАЛО
  IS 'Начало периода действия оплаты';
COMMENT ON COLUMN ДОЛЖНОСТИ.КОНЕЦ
  IS 'Актуальный конец периода оплаты';
COMMENT ON COLUMN ДОЛЖНОСТИ.СОСТОЯНИЕ
  IS 'Состояние (Проект или Утвержден)';


ALTER TABLE ДОЛЖНОСТИ
  ADD CONSTRAINT "Только Проект или Утвержден"
CHECK (СОСТОЯНИЕ IN ('Проект', 'Утвержден'));
```

2. Заполним эту таблицу данными:

```
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (1, 'Зав_производством', 170, '05.11.1987', '14.11.1987',
'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (2, 'Директор', 250, '05.01.1987', '04.11.1987', 'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (3, 'Шеф_повар', 150, '05.11.1987', '09.09.9999', 'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (4, 'Повар_1_категории', 120, '05.11.1987', '09.09.9999',
'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (5, 'Повар_2_категории', 100, '05.11.1987', '09.09.9999',
'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (6, 'Посудомойка', 80, '05.11.1987', '09.09.9999',
'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (7, 'Уборщица', 80, '05.11.1987', '09.09.9999', 'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (8, 'Директор', 270, '05.11.1987', '09.09.9999', 'Утвержден');
insert into ДОЛЖНОСТИ (ИД, ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО, КОНЕЦ, СОСТОЯНИЕ)
values (9, 'Зав_производством', 180, '15.11.1987', '09.09.9999',
'Утвержден');
commit;
```

3. Выполним запрос:

```
select * from должности t ORDER BY должность,начало;
```

4. Результат запроса:

```
ИД ДОЛЖНОСТЬ          ОКЛАД НАЧАЛО      КОНЕЦ      СОСТОЯНИЕ
-- ---------------- ------ ---------- ---------- ---------
 2 Директор         250,00 05.01.1987 04.11.1987 Утвержден
 8 Директор         270,00 05.11.1987 09.09.9999 Утвержден
 1 Зав_производством 170,00 05.11.1987 14.11.1987 Утвержден
 9 Зав_производством 180,00 15.11.1987 09.09.9999 Утвержден
 4 Повар_1_категории 120,00 05.11.1987 09.09.9999 Утвержден
 5 Повар_2_категории 100,00 05.11.1987 09.09.9999 Утвержден
 6 Посудомойка       80,00 05.11.1987 09.09.9999 Утвержден
 7 Уборщица          80,00 05.11.1987 09.09.9999 Утвержден
 3 Шеф_повар        150,00 05.11.1987 09.09.9999 Утвержден
```

Перед выполнением остальных действий создадим последовательность с начальным значением 10:

```
CREATE SEQUENCE ДОЛЖН_ПОСЛ
START WITH 10
INCREMENT BY 1;
```

А теперь рассмотрим "правила игры":

1. Значение должностного оклада устанавливается с определенной даты (НАЧАЛО), а так как не известно, до какой даты оно будет актуальным, то в столбец КОНЕЦ устанавливается запредельная дата: 9 сентября 9999 года.
2. При изменении должностного оклада, с какой-либо даты необходимо ввести в таблицу ДОЛЖНОСТИ новую строку с:
   - наименованием этой должности;
   - новым значением оклада;
   - датой, с которой должен выплачиваться этот оклад (эта дата должна быть больше даты начала предшествующего должностного оклада);
   - датой окончания действия этого оклада, равной 09.09.9999;
   - идентификатором этой строки (ИД), полученным с использованием созданной ранее последовательности ДОЛЖН_ПОСЛ;
   - состоянием 'Проект'.

При этом в предложении INSERT достаточно указать только три первых пункта, последние пункты должны выполниться триггером и значением по умолчанию, указанным в описании таблицы ДОЛЖНОСТИ:

```
СОСТОЯНИЕ    VARCHAR2(10) DEFAULT 'Проект' NOT NULL

CREATE OR REPLACE TRIGGER должн_bir
  BEFORE INSERT ON должности
```

```
  FOR EACH ROW
DECLARE
null_val   EXCEPTION; -- какое-то из обязательных значений не заполнено
nach_nach  EXCEPTION; -- новое начало должно быть больше существующего
test       NUMBER;
ID         NUMBER;
nach       DATE;
BEGIN
--  Если не введены какие-то обязательные значения, выводим сообщение
  IF :new.должность IS NULL
    OR :new.оклад IS NULL
    OR :new.начало IS NULL THEN
    RAISE null_val;
  END IF;
  -- Находим строку с той же должностью и датой конца = '09.09.9999'
  SELECT COUNT(*) INTO test FROM должности
  WHERE должность = :NEW.должность AND конец = '09.09.9999';
     IF test <> 0 THEN
       SELECT ид, начало INTO ID, nach FROM должности
       WHERE должность = :NEW.должность AND конец = '09.09.9999'
         AND состояние = 'Утвержден';
       -- если существующее начало больше нового, выводим сообщение
       IF nach > :NEW.начало THEN
          RAISE nach_nach;
       END IF;
     END IF;
  -- Установка даты конца нового оклада
  :new.конец := '09.09.9999';
  -- генерация ид
  SELECT должн_посл.NEXTVAL INTO :new.ид FROM dual;
EXCEPTION
  WHEN null_val THEN RAISE_APPLICATION_ERROR(-20042,'Не заполнены
                                                обязательные поля !');
  WHEN nach_nach THEN RAISE_APPLICATION_ERROR(-20045,'Вводимое начало
                                              меньше существующего !');
  WHEN OTHERS THEN RAISE_APPLICATION_ERROR(-20999, 'Какая-то другая
                                                          ошибка');
END должн_bir;
/
```

Проверим работу триггера, введя новый должностной оклад:

```
INSERT INTO ДОЛЖНОСТИ (ДОЛЖНОСТЬ, ОКЛАД, НАЧАЛО)
VALUES ('Зав_производством', 181, '15.11.1988');
```

и получив новую строку в таблице:

```
ИД ДОЛЖНОСТЬ          ОКЛАД НАЧАЛО      КОНЕЦ       СОСТОЯНИЕ
-- ---------------- ------ ----------- ----------- ---------
10 Зав_производством 181,00 15.11.1988  09.09.9999  Проект
```

Строка с состоянием 'Проект' должна быть в дальнейшем либо утверждена, либо удалена.

Рассмотрим сначала предложение для удаления введенной строки (строка с ИД=10)

```
DELETE должности WHERE ид=10;
```

и триггер, который должен проверить допустимость удаления:

```
CREATE OR REPLACE TRIGGER должн_bdr
  BEFORE DELETE ON должности
  FOR EACH ROW
DECLARE
  sost_val   EXCEPTION;
BEGIN
--  Если состояние строки "Утвержден", то удалять нельзя
  IF :OLD.состояние = 'Утвержден' THEN
    RAISE sost_val;
  END IF;
EXCEPTION
  WHEN sost_val THEN RAISE_APPLICATION_ERROR(-20043,'Нельзя удалять
  утвержденный должностной оклад!');
  WHEN OTHERS THEN RAISE_APPLICATION_ERROR(-20999,'Какая-то другая
  ошибка');
END должн_bdr;
/
```

Если же требуется утвердить новый должностной оклад, то нужно изменить его состояние на 'Утвержден'. При этом значение даты окончания действия предыдущего оклада по этой должности должно быть заменено датой начала нового оклада, уменьшенной на один день. Изменение даты конца предыдущего должностного оклада выполняем с помощью триггера:

```
CREATE OR REPLACE TRIGGER должн_bur
  BEFORE UPDATE OF состояние ON должности
  FOR EACH ROW
DECLARE
sost_val   EXCEPTION;
test       NUMBER;
ID         NUMBER;
BEGIN
  -- Если состояние строки "Утвержден", то изменять нельзя
  IF :OLD.состояние = 'Утвержден' THEN
    RAISE sost_val;
  END IF;
  -- Ищем строку с той же должностью и датой конца, равной '09.09.9999'
  SELECT COUNT(*) INTO test FROM должности
```

```
  WHERE должность = :OLD.должность AND конец = '09.09.9999';
   IF test <> 0 THEN
     -- Определяем номер найденной строки
     SELECT ид INTO ID FROM должности WHERE должность = :OLD.должность
           AND конец = '09.09.9999' AND состояние = 'Утвержден';
      -- Изменяем дату конца найденной строки
      UPDATE должности SET конец = :OLD.начало-1 WHERE ид = ID;
   END IF;
EXCEPTION
  WHEN sost_val THEN RAISE_APPLICATION_ERROR(-20044,'Нельзя изменять ут-
вержденный должностной оклад!');
END должн_bur;
/
```

Теперь можно утвердить должностной оклад:

```
UPDATE должности SET состояние = 'Утвержден' WHERE ид = 10;
```

Результат выполнения этого предложения дал такой результат:

```
ORA-04091: таблица СООК.ДОЛЖНОСТИ изменяется, триггер/функция может не
заметить это
ORA-06512: на "СООК.ДОЛЖН_BUR", line 11
ORA-04088: ошибка во время выполнения триггера 'СООК.ДОЛЖН_BUR'
```

Вот мы и столкнулись с ошибкой изменяющейся (мутирующей) таблицы (ORA-04091), т. е. попыткой изменить таблицу, с которой связан строковый триггер `ДОЛЖН_BUR`.

А нельзя ли строковый триггер заменить табличным, который не подвержен подобным ошибкам? Тем более что для этого достаточно закомментировать в `ДОЛЖН_BUR` строку `FOR EACH ROW`.

Однако при таком изменении триггера и попытки утвердить должностной оклад, возникает новая ошибка:

```
ORA-04082: ссылки на NEW или OLD недопустимы в триггерах уровня таблицы.
```

Проблему можно разрешить с помощью создания вместо `ДОЛЖН_BUR` двух триггеров — строкового `ДОЛЖН_BSUR (BEFORE)` и табличного `ДОЛЖН_ATUR (AFTER)`. В строковом триггере `ДОЛЖН_BSUR` будут определяться значения `:OLD.ид`, `:OLD.начало`, `:OLD.должность` и `:OLD.состояние`, но сама таблица `ДОЛЖНОСТИ` запрашиваться и изменяться не будет. Запрос и изменение будут выполняться в табличном триггере `ДОЛЖН_ATUR`, где и будут использоваться значения, определенные в строковом триггере.

А где же сохранять и как передать найденные значения `:OLD.ид`, `:OLD.начало`, `:OLD.должность` и `:OLD.состояние`? Oracle рекомендует использовать для этого пакет. Поэтому дальнейшее рассмотрение проблемы изменяющихся таблиц перенесено в *разд. 18.5.5*.

18.5. Пакеты (модули)

18.5.1. Модули

Модуль, или *пакет* (package) — еще одно средство, пришедшее в PL/SQL из языка программирования Ada. Модуль — это конструкция PL/SQL, которая позволяет хранить связанные объекты в одном месте. Модуль состоит из двух частей: описания и тела. Они хранятся по отдельности в словаре данных. В отличие от процедур и функций, которые могут содержаться локально в блоке или храниться в базе данных, модули могут быть только хранимыми и никогда локальными. Помимо того, что модули позволяют группировать связанные объекты, они полезны еще и тем, что ограничений, налагаемых зависимостями, в них меньше, чем в хранимых подпрограммах. Кроме того, они имеют ряд свойств, улучшающих функционирование системы.

В сущности, модуль представляет собой именованный раздел объявлений. Все, что может входить в состав раздела объявлений блока, может входить и в модуль: процедуры, функции, курсоры, типы и переменные. Размещение их в модуле полезно тем, что это позволяет обращаться к ним из других блоков PL/SQL, поэтому в модулях можно описывать глобальные переменные PL/SQL (внутри одного сеанса работы с базой данных).

18.5.2. Создание описания пакета

Описание спецификации (заголовка) пакета

Спецификация (заголовок) пакета (package specification) содержит информацию о составе модуля, однако в описание не входит текст процедур. Синтаксис спецификации пакета имеет вид:

```
CREATE [OR REPLACE] PACKAGE [схема.]имя_пакета
   [ AUTHID {CURRENT_USER | DEFINER} ]
   { IS | AS } pl/sql_спецификация_пакета;
```

Здесь:

`AUTHID CURRENT_USER`

Служит для того, чтобы указать, что пакет выполняется с привилегиями `CURRENT_USER` и что внешние названия в предложениях языка соответствуют схеме `CURRENT_USER`. В противном случае все будет соответствовать схеме, в которой расположен пакет.

`AUTHID DEFINER`

Служит для того, чтобы указать, что пакет выполняется с привилегиями владельца схемы, в которой расположен пакет, и что внешние названия в предложениях языка соответствуют этой же схеме.

`IS | AS`

Ключевые слова `IS | AS` идентичны и начинают *pl/sql_спецификация_пакета*.

pl/sql_спецификация_пакета

Может включать: *определение_типа*, *описание_процедуры*, *описание_функции*, *объявление_переменной*, *объявление_исключителъной_ситуации*, *объявление_курсора*, *объявление_прагмы*.

Элементы пакета (описания процедур и функций, переменные и т. д.) аналогичны тому, что указывается в разделе объявлений анонимного блока. Для заголовка пакета действуют те же синтаксические правила, что и для раздела объявлений, за исключением объявлений процедур и функций. Перечислим эти правила.

- Элементы пакета могут указываться в любом порядке. Однако, как и в разделе объявлений, объект должен быть объявлен до того, как на него будут произведены ссылки. Например, если частью предложения `WHERE` курсора является некоторая переменная, она должна быть объявлена до объявления курсора.
- Совсем не обязательно, чтобы присутствовали элементы всех видов. К примеру, модуль может состоять только из описаний процедур и функций и не содержать объявлений исключительных ситуаций или типов.
- Объявления всех процедур и функций должны быть предварительными. В *предварительном объявлении* (forward declaration) описываются подпрограмма и ее аргументы (если есть), но не включается программный текст. В этом отличие пакета от раздела объявлений блока, где могут находиться как предварительные объявления, так и текст процедур и функций. Программный текст процедур и функций пакета содержится в теле этого пакета.

Описание тела пакета

Тело пакета (package body) — это объект словаря данных, хранящийся отдельно от заголовка пакета. Тело пакета нельзя скомпилировать, если ранее не был успешно скомпилирован заголовок. В теле содержится текст подпрограмм, предварительно объявленных в заголовке пакета. В нем могут находиться также дополнительные объявления, глобальные для тела пакета, но не видимые в его описании.

Синтаксис тела пакета имеет вид:

```
CREATE [OR REPLACE] PACKAGE BODY [схема.]имя_пакета
   { IS | AS } pl/sql_тело_пакета;
```

В теле пакета содержится программный текст (*pl/sql_тело_пакета*) для предварительных объявлений, сделанных в заголовке пакета, и могут также находиться дополнительные переменные, курсоры, типы и подпрограммы. На объекты в заголовке, которые не были объявлены предварительно (например, исключение), можно ссылаться в теле пакета непосредственно.

Тело пакета не является обязательной его частью. Если в заголовке не указаны какие-либо процедуры или функции (а только переменные, курсоры, типы и т. д.), тело можно не создавать. Такой способ полезен для объявления глобальных переменных, поскольку все объекты пакета видимы вне его пределов (область действия и область видимости элементов пакета обсуждаются в следующем разделе).

Любое предварительное объявление в заголовке пакета должно быть раскрыто в его теле. Описание процедуры или функции должно быть таким же и включать в свой состав имя подпрограммы, имена ее параметров и вид каждого параметра.

Область действия пакетов и их объектов

Любой объект, объявленный в заголовке пакета, находится в области действия и видим вне этого пакета. При обращении к объекту нужно указать имя пакета. Например, для вызова процедуры `p_upd1`, размещенной в пакете `pk_долж`, надо предварить имя процедуры именем пакета:

```
pk_долж.p_upd1
```

Внутри тела пакета на объекты, представленные в его заголовке, можно ссылаться без указания имени пакета.

Если процедура объявлена в теле пакета локальной (т. е. она отсутствует в описании заголовка пакета), то область ее действия — только тело пакета. Ее можно вызывать из других процедур тела пакета, но вне тела она невидима.

18.5.3. Изменение описания пакета

Для изменения описания пакета используется предложение:

```
ALTER PACKAGE [схема.]имя_пакета
  COMPILE [DEBUG] [PACKAGE | SPECIFICATION | BODY]
```

```
  [имя_параметра = значение_параметра
    [имя_параметра = значение_параметра] ... ]
  [REUSE SETTINGS];
```

Здесь:

`COMPILE`

Ключевое слово перед тем, как указываются объекты перекомпиляции и (или) отладка.

`PACKAGE`

Определяет, что нужно перекомпилировать спецификацию и тело пакета, а также всех зависимых объектов.

`SPECIFICATION`

Определяет, что нужно перекомпилировать спецификацию. При этом будут лишены законной силы любые местные объекты, которые зависят от спецификации, типа процедур, которые называют процедуры или функции в пакете. Тело пакета также зависит от его спецификации. Если впоследствии делается ссылка на один из этих зависимых объектов (без предварительного их перекомпилирования), то СУБД сама их перекомпилирует.

`BODY`

Определяет, что нужно перекомпилировать тело пакета. Перекомпилирование тела пакета не лишает законной силы объекты, которые зависят от спецификации пакета.

`DEBUG`

Дает указание компилятору генерировать и хранить код для использования отладчиком. Определение этого пункта имеет тот же самый эффект как установка `PLSQL_DEBUG=TRUE` в *`compiler_parameters_clause`*.

`compiler_parameters_clause`

Используется для определения текущих значений, указанных в перечне параметров компилятора PL/SQL.

`REUSE SETTINGS`

Не позволяет изменять существующие параметры настройки при перекомпиляции.

18.5.4. Удаление пакета

Для удаления всего пакета или только его тела (`BODY`) используется синтаксис:

```
DROP PACKAGE [BODY] [схема.]имя_пакета;
```

18.5.5. Примеры пакетов

Продолжим рассмотрение примера *разд. 18.4.5*.

1. Создадим строковый триггер ДОЛЖН_BSUR, в котором вызывается процедура P_UPD1 пакета PK_ДОЛЖ (см. далее). В нее передаются в качестве входных параметров значения некоторых столбцов обновляемой строки.

```
CREATE OR REPLACE TRIGGER должн_bsur
  BEFORE UPDATE OF состояние ON должности
  FOR EACH ROW
BEGIN
  pk_долж.p_upd1(:OLD.ид, :OLD.начало,:OLD.должность,:OLD.состояние);
END должн_bsur;
/
```

2. Аналогично создается табличный триггер ДОЛЖН_ATUR, в котором вызывается процедура P_UPD2 того же пакета.

```
CREATE OR REPLACE TRIGGER должн_atur
  AFTER UPDATE OF состояние ON должности
BEGIN
  pk_долж.p_upd2;
END должн_atur;
/
```

3. Теперь создаем спецификацию (заголовок) пакета PK_ДОЛЖ, в которой описаны две процедуры, вызываемые в приведенных ранее триггерах:

```
CREATE OR REPLACE PACKAGE pk_долж AS
  -- Описание структуры пакета процедур для триггеров таблицы ДОЛЖНОСТИ
PROCEDURE p_upd1
  (
   o_ИД               ДОЛЖНОСТИ.ИД%TYPE,
   o_НАЧАЛО           ДОЛЖНОСТИ.НАЧАЛО%TYPE,
   o_ДОЛЖНОСТЬ        ДОЛЖНОСТИ.ДОЛЖНОСТЬ%TYPE,
   o_СОСТОЯНИЕ        ДОЛЖНОСТИ.СОСТОЯНИЕ%TYPE
  );
PROCEDURE p_upd2;
END pk_долж;
```

4. Далее создаем тело этого пакета:

```
CREATE OR REPLACE PACKAGE BODY pk_долж AS
  -- Тело пакета процедур для триггеров таблицы ДОЛЖНОСТИ.
  -- Описание переменных, предназначенных для хранения некоторых
  -- значений строки
```

```
v_ид                должности.ид%TYPE;
v_НАЧАЛО            ДОЛЖНОСТИ.НАЧАЛО%TYPE;
v_ДОЛЖНОСТЬ         ДОЛЖНОСТИ.ДОЛЖНОСТЬ%TYPE;
v_состояние         должности.состояние%TYPE;
-- Процедура, запускаемая триггером должн_bsur (BEFORE UPDATE ON
-- ДОЛЖНОСТИ FOR EACH ROW). Этот строковый триггер стоит на первом месте
-- в последовательности исполнения триггеров, созданных для UPDATE)
--
-- Получение значений столбцов обновляемой строки и перепись их
-- в глобальные переменные
PROCEDURE p_upd1(
   o_ид             должности.ид%TYPE,
   o_НАЧАЛО         ДОЛЖНОСТИ.НАЧАЛО%TYPE,
   o_ДОЛЖНОСТЬ      ДОЛЖНОСТИ.ДОЛЖНОСТЬ%TYPE,
   o_состояние      должности.состояние%TYPE
  ) IS
BEGIN
  v_ид        := o_ид;
  v_НАЧАЛО    := o_НАЧАЛО;
  v_ДОЛЖНОСТЬ := o_ДОЛЖНОСТЬ;
  v_состояние := o_состояние;
END p_upd1;
--
-- Процедура, запускаемая триггером должн_atur (AFTER UPDATE ON
-- ДОЛЖНОСТИ). Этот табличный триггер стоит на втором месте
-- в последовательности исполнения триггеров, созданных для UPDATE)
--
-- Анализ обновляемой строки и, если необходимо, принятие решения
-- об отклонении обновления с выдачей сообщения о причине.
PROCEDURE p_upd2 IS
  -- Описание локальных переменных и исключений
  ID        NUMBER;
  test      NUMBER;    -- Рабочая переменная
  sost_val  EXCEPTION; -- Нельзя изменять утвержденный должностной оклад
BEGIN
  -- Если состояние строки "Утвержден", то изменять нельзя
  IF v_состояние = 'Утвержден' THEN
    RAISE sost_val;
```

```
  END IF;
  -- Ищем строку с той же должностью и датой конца, равной '09.09.9999'
  SELECT COUNT(*) INTO test FROM должности
  WHERE должность = v_ДОЛЖНОСТЬ AND конец = '09.09.9999'
  AND состояние = 'Утвержден' AND ид <> v_ид;
   IF test <> 0 THEN
      -- Определяем номер найденной строки
      SELECT ид INTO ID FROM должности WHERE должность = v_ДОЛЖНОСТЬ
           AND конец = '09.09.9999' AND состояние = 'Утвержден'
           AND ид <> v_ид;
      -- Изменяем дату конца найденной строки
      UPDATE должности SET конец = v_НАЧАЛО-1 WHERE ид = ID;
   END IF;
EXCEPTION
  WHEN sost_val THEN RAISE_APPLICATION_ERROR
       (-20044,'Нельзя изменять утвержденный должностной оклад!');
END p_upd2;
END pk_долж;
/
```

5. Снова попробуем утвердить должностной оклад:

```
UPDATE должности SET состояние = 'Утвержден' WHERE ид = 10;
```

6. Результат выполнения этого предложения:

```
1 row updated
```

7. Проверка результата обновления, выполненная с помощью предложения

```
select * from должности t ORDER BY должность,начало;
```

```
ИД ДОЛЖНОСТЬ           ОКЛАД НАЧАЛО      КОНЕЦ       СОСТОЯНИЕ
-- ----------------- ------ ---------- ---------- ----------
 2 Директор          250,00 05.01.1987  04.11.1987  Утвержден
 8 Директор          270,00 05.11.1987  09.09.9999  Утвержден
 1 Зав_производством 170,00 05.11.1987  14.11.1987  Утвержден
 9 Зав_производством 180,00 15.11.1987  14.12.1988  Утвержден
10 Зав_производством 185,00 15.12.1988  09.09.9999  Утвержден
 4 Повар_1_категории 120,00 05.11.1987  09.09.9999  Утвержден
 5 Повар_2_категории 100,00 05.11.1987  09.09.9999  Утвержден
 6 Посудомойка        80,00 05.11.1987  09.09.9999  Утвержден
 7 Уборщица           80,00 05.11.1987  09.09.9999  Утвержден
 3 Шеф_повар         150,00 05.11.1987  09.09.9999  Утвержден
```

18.6. Встроенные пакеты PL/SQL

PL/SQL предлагает целый ряд дополнительных функций, которые доступны в поставляемых им пакетах. Эти пакеты принадлежат пользователю SYS, однако для них созданы общие синонимы, поэтому пакеты можно вызывать, не используя префикс SYS перед именем пакета. Для вызова процедур и функций этих пакетов пользователь, который не является SYS, должен иметь полномочие EXECUTE на конкретный пакет. Дополнительную информацию можно найти в документации Oracle и литературе [10].

Далее кратко описываются некоторые из почти тысячи встроенных пакетов (модулей), доступных для использования в PL/SQL.

DBMS_ALERT

Применяется для обмена сообщениями между сеансами, соединенными с одной и той же базой данных. Оповещения (alerts) синхронны, т. е. они посылаются при завершении транзакций. Оповещения отправляет процедура SIGNAL, а получают процедуры WAITONE и WAITANY. Чтобы получить оповещение, сеанс должен быть зарегистрирован с помощью процедуры REGISTER. Информацию об оповещениях можно найти в таблице словаря данных dbms_alert_info.

DBMS_APPLICATION_INFO

Применяется для регистрации информации (info) о конкретной программе в системной таблице v$session. С помощью этой информации можно определить, какие приложения (applications) функционируют в данный момент и какие действия они выполняют. В частности, DBMS_APPLICATION_INFO можно использовать для считывания и модификации столбцов module, action и client_info таблицы v$session текущего сеанса.

В столбце module хранится имя приложения. Например, SQL*Plus помещает в этот столбец значение 'SQL*Plus'. В столбце action содержится фрагмент приложения, выполняющийся в данный момент. Например, можно задать определенное значение action перед вводом фрагмента программы Pro*C. В столбец client_info можно вносить произвольные строки символов.

DBMS_APPLICATION_INFO может применяться также (с помощью процедуры SET_SESSION_LONGOPS) для обновления таблицы v$session_longop, используемой для указания статуса долговременных операций.

DBMS_DDL

Предоставляет PL/SQL-эквиваленты ряда полезных команд DDL, которые нельзя использовать в PL/SQL непосредственно. Для выполнения таких команд можно применять динамический SQL, но в модуле DBMS_DDL они имеют

другой синтаксис. Среди процедур в DBMS_DLL имеются ALTER_COMPILE, служащая для компиляции пакетов, триггеров, процедур и функций, и ANALYZE_OBJECT, применяемая для анализа таблиц, кластеров и индексов.

DBMS_OUTPUT

DBMS_OUTPUT (в комбинации с SQL*Plus или SQLDEVELOPER) предоставляет ограниченные возможности вывода для PL/SQL.

Это удобно при тестировании и отладке кода PL/SQL. DBMS_OUTPUT не предназначен для написания отчетов — для этого лучше подходит утилита Oracle Reports.

DBMS_SQL

DBMS_SQL реализует динамический PL/SQL. С его помощью можно создавать операторы SQL и блоки PL/SQL во время выполнения и выполнять их. Кроме того, DBMS_SQL можно использовать для выполнения в PL/SQL операторов DDL. Oracle предлагает внутренний динамический SQL (см. *разд. 17.8*), более быстрый и простой, чем DBMS_SQL, использующийся в ранних версиях Oracle.

DBMS_STANDARD и STANDARD

Совместно DBMS_STANDARD и STANDARD реализуют все встроенные функции PL/SQL. В отличие от других модулей, имена входящих в них подпрограмм не нужно предварять именем модуля.

DBMS_STATS

Позволяет просматривать и модифицировать статистические показатели оптимизатора, хранящиеся в словаре данных или в определенной пользователем таблице. Оптимизатор по стоимости выполнения будет использовать статистику только из словаря данных.

ЧАСТЬ VII

ПРИМЕР СОЗДАНИЯ БАЗЫ ДАННЫХ "UCHEB"

Глава 19

Описание предметной области "Учебные планы"

19.1. О Государственных образовательных стандартах (ГОС)

Во всех учебных заведениях обучение производится по утвержденным учебным планам и программам, составленным в соответствии с Государственными стандартами. На рис. 19.1 приведены выдержки из ГОС направления подготовки дипломированного специалиста 654600 — Информатика и вычислительная техника, в рамках которого ведется подготовка по специальности 220100 — Вычислительные машины, комплексы, системы и сети, составленного учебно-методическим объединением (УМО) вузов по образованию в области машиностроения и приборостроения. Созданы и другие УМО (например, Учебно-методическое объединение по оптическому и приборостроительному образованию, председателем которого является ректор СПбГУ ИТМО В. Н. Васильев).

Учебно-методические объединения создают не только ГОС, но и примерную образовательную программу, включающую примерные учебные планы, а также примерные программы учебных дисциплин и производственных практик. Эти документы являются основой для разработки основных образовательных программ вуза — учебных планов (базовых учебных планов) и программ.

На рис. 19.2 для иллюстрации приведены выдержки из примерного учебного плана по направлению подготовки дипломированного специалиста 654600 — Информатика и вычислительная техника.

ГОС'ы и примерные планы по любому направлению подготовки дипломированного специалиста и (или) специальности, направлениям подготовки бакалавров и магистров можно найти в Интернете по адресу: **http://db.informika.ru/spe/archiv.htm**.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДАЮ
Заместитель Министра
образования Российской
Федерации
_____В. Д. Шадриков
"27" 03 2000г.
Регистрационный номер 224 тех/дс

ГОСУДАРСТВЕННЫЙ ОБРАЗОВАТЕЛЬНЫЙ СТАНДАРТ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

Направление подготовки дипломированного специалиста
654600 – Информатика и вычислительная техника

Квалификация - *инженер*

Вводится с момента утверждения

Москва 2000 г.

---

1. Общая характеристика направления подготовки дипломированного специалиста "Информатика и вычислительная техника"

1.1. Направление подготовки дипломированного специалиста утверждено приказом Министерства образования Российской Федерации от 02.03.2000 г. № 686.
1.2. Перечень образовательных программ (специальностей), реализуемых в рамках данного направления подготовки дипломированного специалиста:

220100 – Вычислительные машины, комплексы, системы и сети;
220200 – Автоматизированные системы обработки информации и управления;
220300 – Системы автоматизированного проектирования;
220400 – Программное обеспечение вычислительной техники и автоматизированных систем.

1.3. Квалификация выпускника - *инженер*.
...

2. Требования к уровню подготовки абитуриента

...

3. Общие требования к основной образовательной программе по направлению подготовки дипломированного специалиста "Информатика и вычислительная техника"

3.1. Основная образовательная программа подготовки инженера разрабатывается на основании настоящего государственного образовательного стандарта дипломированного специалиста и включает в себя перечень учебных дисциплин, программы учебных, научных и производственных практик.
...
3.4. Основная образовательная программа подготовки инженера должна предусматривать изучение студентом следующих циклов дисциплин:

- цикл ГСЭ - Общие гуманитарные и социально-экономические дисциплины;
- цикл ЕН - Общие математические и естественнонаучные дисциплины;
- цикл ОПД - Общепрофессиональные дисциплины;
- цикл СД - Специальные дисциплины, включая дисциплины специализации;
- ФТД - Факультативные дисциплины.

Рис. 19.1, *а*. Выдержки из ГОС 654600 — Информатика и вычислительная техника

4. Требования к обязательному минимуму содержания основной образовательной программы подготовки дипломированного специалиста по направлению "Информатика и вычислительная техника"

Индекс	Наименование дисциплин и их основные разделы	Всего часов
1	2	3
ГСЭ	Общие гуманитарные и социально-экономические дисциплины	**1800**
ГСЭ.Ф.00. ГСЭ.Ф.01.	Федеральный компонент **Иностранный язык** Специфика артикуляции звуков, интонации, акцентуации и ритма …	1260 340
…		
ГСЭ.Р.00.	Национально-региональный (вузовский) компонент	270
ГСЭ.В.00.	Дисциплины и курсы по выбору студента, из числа устанавливаемых вузом	270
ЕН	**ОБЩИЕ МАТЕМАТИЧЕСКИЕ И ЕСТЕСТВЕННОНАУЧНЫЕ ДИСЦИПЛИНЫ**	**1912**
ЕН.Ф.00.	Федеральный компонент	1572
ЕН.Ф.01.	**Математика**	960
ЕН.Ф.01.01	*Алгебра и геометрия.* Основные алгебраические структуры, векторные пространства и линейные отображения. …	140
ЕН.Р.00	Национально-региональный (вузовский) компонент	170
ЕН.В.00	Дисциплины по выбору студента, устанавливаемые вузом	170
ОПД	**ОБЩЕПРОФЕССИОНАЛЬНЫЕ ДИСЦИПЛИНЫ**	**2200**
ОПД.Ф.00	Федеральный компонент	1860
ОПД.Ф.01	**Начертательная геометрия. Инженерная и компьютерная графика**	240
… ОПД.Ф.10	… **Базы данных** Назначение и основные компоненты системы баз данных; обзор современных систем управления базами данных (СУБД); …	140
ОПД.Р.00	Национально-региональный (вузовский) компонент	170
ОПД.В.00	Дисциплины по выбору студента, устанавливаемые вузом	170
СД	**СПЕЦИАЛЬНЫЕ ДИСЦИПЛИНЫ**	**1900**
СП.01	**Специальность "Вычислительные машины, комплексы, системы и сети"**	**1390**
ДС.00	Дисциплины специализации	510

Всего часов теоретического обучения **8748**

5. Сроки освоения основных образовательных программ по направлению подготовки дипломированных специалистов "Информатика и вычислительная техника"

5.1. Срок освоения основной образовательной программы подготовки инженера при очной форме обучения составляет …

…

5.3. Максимальный объем учебной нагрузки студента устанавливается 54 часа в неделю, включая все виды его аудиторной и внеаудиторной (самостоятельной) учебной работы.

5.4. Объем аудиторных занятий студента при очной форме обучения не должен превышать в среднем за период теоретического обучения 27 часов в неделю. При этом в указанный объем не входят обязательные практические занятия по физической культуре и занятия по факультативным дисциплинам.

…

СОСТАВИТЕЛИ:

Учебно-методическое объединение вузов
по образованию в области машиностроения и приборостроения
Председатель Совета УМО _____ И.Б. Федоров

Рис. 19.1, *б*. Выдержки из ГОС 654600 — Информатика и вычислительная техника

Министерство образования Российской Федерации

Утверждаю:
Начальник Управления
Шестаков Г.К.
"_27"__03__2000 г.

Регистрационный
номер 224 тех/дс_

ПРИМЕРНЫЙ УЧЕБНЫЙ ПЛАН
Направление подготовки
Дипломированного специалиста

654600
Информатика и вычислительная техника

инженер
квалификация

5 лет
нормативный срок обучения

N п\п	Наименование Дисциплин	ЧАСОВ			ПРИМЕРНОЕ распределение по семестрам (указать крестиком)										Форма итогового Контроля (экзамен\ зачет)
		Трудоемкость по Госстандарту	Из них												
			Аудиторные занятия	Самостоятельная работа	1	2	3	4	5	6	7	8	9	10	
ОБЩИЕ ГУМАНИТАРНЫЕ И СОЦИАЛЬНО-ЭКОНОМИЧЕСКИЕ ДИСЦИПЛИНЫ															
	Федеральный компонент	1260	850	410											
1	Иностранный язык	340	170	170	+	+	+	+							Э
2	Физическая культура	408	408		+	+	+	+	+	+	+	+			З
ОБЩИЕ МАТЕМАТИЧЕСКИЕ И ЕСТЕСТВЕННОНАУЧНЫЕ ДИСЦИПЛИНЫ															
	Федеральный компонент	1570	833	737											
1	Математика:	960	527	433											
1.1	Алгебра и геометрия	140	68	72	+										Э
ОБЩЕПРОФЕССИОНАЛЬНЫЕ ДИСЦИПЛИНЫ															
	Федеральный компонент	1860	952	908											
1.1	Инженерная графика	100	51	49	+										Э
1.2	Компьютерная графика	140	68	72					+						Э
10	Базы данных	140	68	72					+						Э
СПЕЦИАЛЬНЫЕ ДИСЦИПЛИНЫ															
Специальность 220100 Вычислительные машины, комплексы, системы и сети															

Примечание: Э - экзамен, З - зачет

Рис. 19.2. Выдержки из примерного учебного плана направления 654600

19.2. Основные образовательные программы (ООП)

19.2.1. Базовые учебные планы

Созданные учебно-методическими объединениями и научно-методическими советами ***примерные учебные планы*** и ***примерные программы*** являются основой для разработки основных образовательных программ вуза — базовых учебных планов, а также программ учебных дисциплин и производственных практик.

При составлении базовых учебных планов вуз:

- устанавливает последовательность (по семестрам) изучения дисциплин и прохождения практик;
- определяет наименования и объемы дисциплин специализаций, факультативов, дисциплин регионального (вузовского) компонента и дисциплин по выбору студентов;
- устанавливает объем и вид аудиторных занятий, а также формы итогового контроля по каждой дисциплине;
- добивается, чтобы суммарный объем часов в каждом семестре был равен произведению рабочей недели студента на количество недель в семестре (например, для очной формы обучения 54*17 = 918 часов);
- добивается, чтобы объем обязательных аудиторных занятий студента не превышал в среднем за период теоретического обучения определенной стандартом границы (например, 27 часов в неделю для студентов очной формы обучения).

Напомним, что стандарты отводят достаточно большие объемы часов для ***дисциплин по выбору студента*** по каждому из основных циклов дисциплин (ГСЭ, ЕН, ОПД). На эти часы в каждом цикле могут планироваться несколько дисциплин, расположенных в одном или нескольких семестрах. Для каждой из дисциплин по выбору студента должно существовать не менее двух альтернативных вариантов, предлагаемых одной или несколькими кафедрами. Так как такие дисциплины могут ежегодно обновляться, то их индексы (ГСЭ.В.00, ЕН.В.00 или ОПД.В.00) и названия будут уточняться в рабочих учебных планах. В базовых учебных планах для них отводится определенное место, производится распределение часов по видам занятий и устанавливаются формы итогового контроля.

Базовые учебные планы могут оформляться по-разному, но все они должны содержать в том или ином виде:

- заголовок, включающий:
  - названия вуза;
  - направления подготовки и (или) специальности и, может быть, специализации;
  - квалификацию (степень);
  - форму и срок обучения;
- график учебного процесса;
- строки учебного плана, содержащие:
  - индексы и названия дисциплин (практик, госаттестаций и др.);
  - объемы часов по видам занятий и распределение аудиторных часов по семестрам;
  - виды итогового контроля и шифр кафедры, которой поручено проведение занятий по дисциплине;
- итоговые сведения о числе учебных и аудиторных часов, экзаменов и зачетов в каждом из семестров.

На рис. 19.3 приведены фрагменты одного из представлений базового учебного плана, созданного с помощью Microsoft Excel, а на рис. 19.4 показан график учебного процесса к этому учебному плану.

Напомним также, что при разработке образовательной программы вуз (факультет) имеет право: "Изменять объем часов, отводимых на освоение учебного материала для циклов дисциплин — в пределах 5%, для дисциплин, входящих в цикл, — в пределах 10% без превышения максимального недельного объема нагрузки студентов", назначать названия и объемы дисциплин специализации и факультативов и т. п.

Вообще, при составлении учебных планов вуз пытается по возможности "усреднить" объемы гуманитарных, естественнонаучных и общепрофессиональных дисциплин близких направлений и специальностей с целью создания единых программ и лекционных потоков. В процессе согласования учебных планов разных направлений и специальностей составители должны достаточно часто переставлять дисциплины из одного семестра в другой, меняя их общие объемы и объемы аудиторной работы. При этом необходимо выполнять ограничения на максимальный объем учебной нагрузки студента и средний объем аудиторных занятий, число экзаменов, зачетов, курсовых и т. п. Все это достаточно просто выполняется в Excel, доступном любому разработчику плана (именно так выполнен учебный план, показанный на рис. 19.3).

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное агенство по образованию

Государственное образовательное учреждение высшего профессионального образования
Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

"УТВЕРЖДАЮ"

Ректор ИТМО
д.т.н., проф. ***Васильев В.Н.***

_____ _____ 200_ г.

инженер
квалификация
5 лет
срок обучения

УЧЕБНЫЙ ПЛАН
ПОДГОТОВКИ ДИПЛОМИРОВАННОГО СПЕЦИАЛИСТА ПО
СПЕЦИАЛЬНОСТИ: 230101.65 "ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ, КОМПЛЕКСЫ, СИСТЕМЫ И СЕТИ"
Очная форма обучения

Специализация 230101.65.11 - "Открытые информационно-вычислительные системы"

Общая продолжительность обучения - 10 семестров или 260 недель, в том числе:
теоретическое обучение - 161 недель или 8694 часов;
экзаменационные сессии - 27 недель;
практика - 10 недель;
итоговая государственная аттестация, включая выполнение ВКР - не менее 16 недель (864 часа);
каникулы - не менее 38 недель;
отпуск после окончания вуза - 8 недели.

Цифры в колонках 7,8,9 и т.д. (сверху вниз) по каждой дисциплине обозначают: первая - число часов лекций;
вторая - число часов лабораторных занятий; третья - число часов практических занятий (семинаров);
четвертая - число часов самостоятельной работы студента (СРС)

Индекс дисцип-лины	Название дисциплины	Форма итогового контроля		Объем работы студента (час)			Распределение по курсам и семестрам учебных часов в неделю										Код ка-федры
							КУРС										
							1		2		3		4		5		
		экзамен	зачет	всего	ауд. занятия	по видам занятий	СЕМЕСТР										
							1	2	3	4	5	6	7	8	9	10	
							КОЛИЧЕСТВО НЕДЕЛЬ										
							18	18	18	18	18	18	18	18	17	16	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19
	ГУМАНИТАРНЫЕ И СОЦИАЛЬНО-ЭКОНОМИЧЕСКИЕ ДИСЦИПЛИНЫ																
ГСЭ.Ф.01	Иностранный язык	4	1	340	180	0											ИЯ
			2			0											
			3			180	3	3	2	2							
						160	2,3	2,2	2,2	2,2							
...	...																
	ДИСЦИПЛИНЫ ЕСТЕСТВЕННО-НАУЧНОГО ЦИКЛА																
ЕН.Ф.01.01	Алгебра и геометрия	1		135	90	54	3										ВМ
						0	0										
						36	2										
						45	2,5										
ЕН.Ф.01.02	Математический анализ	2		342	216	108		3	3								ВМ
		3				0		0	0								
						108		3	3								
						126		3,1	3,9								
...	...																
ИТОГО:																	
Учебных часов теоретического обучения:				**8748**			54,0	54,0	54,0	54,0	54,0	54,0	54,0	54,0	54,0	54,0	**8748**
Аудиторных часов:					**3591**		24	24	22	22	20	20	18	18	16	0	**3591**
Экзаменов:							4	4	5	5	5	4	5	4	4	0	
Зачетов:							5	5	3	6	4	5	3	4	4	0	
Государственный экзамен																1	

Примечания:

1. Базовый учебный план предусматривает распределение по курсам, семестрам и формам итогового
...

Рис. 19.3. Фрагменты базового учебного плана специализации 230101

"Утверждаю"
Ректор СПбГУ ИТМО
__________ Васильев В.Н.
"__" ________ 200 г.

Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Квалификация специалиста	Срок обуч. (в год.)
	5 лет

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

План одобрен Ученым советом вуза
протокол №

УЧЕБНЫЙ ПЛАН

Специальность 230101.65

График учебного процесса

Курс	Сентябрь					Октябрь				Ноябрь				Декабрь				
	1-7 сен	8-14 сен	15-21 сен	22-28 сен	29 сен - 5 окт	6-12 окт	13-19 окт	20-26 окт	27 окт - 02 ноя	3-9 ноя	10-16 ноя	17-23 ноя	24 ноя - 30 ноя	1-7 дек	8-14 дек	15-21 дек	22-28 дек	29 дек - 4 янв
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1																		
2																		
3																		
4																		
5																		э

Курс	Январь				Февраль				Март					Апрель			
	5-11 янв	12-18 янв	19-25 янв	26 янв - 1 фев	2-8 фев	9-15 фев	16-22 фев	23 фев - 1 мар	2-8 мар	9-15 мар	16-22 мар	23-29 мар	30 мар - 5 апр	6-12 апр	13-19 апр	20-26 апр	27 апр - 3 май
	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	э	э	э	к	к												
2	э	э	э	к	к												
3	э	э	э	к	к												
4	э	э	э	к	к												
5	э	э	к	к	п	п	п	п	п	п	д	д	д	д	д	д	д

Курс	Май				Июнь					Июль				Август			
	4-10 май	11-17 май	18-24 май	25-31 май	1-7 июн	8-14 июн	15-21 июн	22-28 июн	29 июн - 5 июл	6-12 июл	13-19 июл	20-26 июл	27 июл - 2 авг	3-9 авг	10-16 авг	17-23 авг	24-30 авг
	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52
1							э	э	э	к	к	к	к	к	к	к	к
2							э	э	э	п	п	к	к	к	к	к	к
3							э	э	э	п	п	к	к	к	к	к	к
4							э	э	э	к	к	к	к	к	к	к	к
5	д	д	д	д	д	д	д	д	д	к	к	к	к	к	к	к	к

Курс	Неделъ: Теор. обучен	Экзаменационные сессии	Итоговая Аттестация	Практика	Дипл. раб. или проекты	Гос. Экзамены	Каникулы	Всего
1	36	6					10	52
2	36	6		2			8	52
3	36	6		2			8	52
4	36	6					10	52
5	17	3		6	36		10	52
Всего	161	27	0	10	16	0	46	260
ГОС	153	22		10 / 4 / 6			38	

Обозначения:
[ ] - Теоретическое обучение
[э] - Экзаменационная сессия
[п] - Практика
[нр] - Научно-исследовательская работа студентов
[К] - Каникулы
[Г] - Государственные экзамены
[вр] - Выпускная квалификационная работа

Рис. 19.4. График учебного процесса к базовому учебному плану специализации 230101

Следует отметить, что в вузах используются формы представления базовых учебных планов, имеющих различную степень детализации. Достаточно распространена показанная на рис. 19.5 форма, которая с небольшими модификациями существует более пятидесяти лет.

№ п/п	Название дисциплины	Распределение по семестрам				ЧАСОВ					Распределение по курсам и семестрам				
						Всего	из них				1 курс		2 курс		3 ку
		Экзаменов	Зачетов	Курсовых проектов	Курсовых работ		Лекии	Лабораторные занятия	Практические занятия	Самостоятельная работа	1 Семестр 17 Недель	2Семестр 17 Недель	3 Семестр 17 Недель	4 Семестр 17 Недель	5 Семестр 17 Недель
											Аудиторных часов в неделю				
1	2	3	4	5	6	7	6	9	10	11	12	13	14	15	16
1	Иностранный язык	4	1,2,3			340			170	170	3	3	2	2	
	…														
11	Математика	1,2, 3,4				700	136		187	377	5	5	5	4	
12	Информатика	1,2			1	201	51	51		99	3	3			
	…														
Итого						9180	950	1240	1210	5780	24	24	22	22	
…															

Рис. 19.5. Фрагменты базового учебного плана (второй вариант)

Основной недостаток этой формы заключается в том, что в ней отсутствует распределение часов по видам занятий в каждом из семестров (указано лишь распределение аудиторных часов). А как, например, распределен по семестрам 51 час лекций дисциплины "Информатика": они читаются только в первом семестре или как-то поделены между семестрами? Из-за отсутствия посеместрового распределения часов СРС нельзя обеспечить требования ГОС: "5.3.

Максимальный объем учебной нагрузки студента устанавливается 54 часа в неделю, включая все виды его аудиторной и внеаудиторной (самостоятельной) учебной работы".

19.2.2. Индивидуальные учебные планы и планы ускоренного обучения

В п. 6.1.2 ГОС сказано:

При реализации основной образовательной программы высшее учебное заведение имеет право:

> …
>
> — реализовывать основную образовательную программу подготовки инженера в сокращенные сроки для студентов высшего учебного заведения, имеющих среднее профессиональное образование соответствующего профиля или высшее профессиональное образование. Сокращение сроков проводится на основе аттестации имеющихся знаний, умений и навыков студентов, полученных на предыдущем этапе профессионального образования. При этом продолжительность сокращенных сроков обучения должна составлять не менее трех лет при очной форме обучения. Обучение по ускоренным программам допускается также для лиц, уровень образования или способности которых являются для этого достаточным основанием.

Такие студенты (например, студенты, окончившие профилирующий техникум) могут быть объединены в студенческие группы, для которых составляются и утверждаются ***учебные планы ускоренного обучения***. Особо одаренные студенты могут обучаться по ***индивидуальным учебным планам***, т. е. планам, составленным и утвержденным для каждого из них в отдельности. Естественно, что для получения диплома по выбранному направлению (специальности) в этих планах должны присутствовать все дисциплины федерального компонента (часть из которых может быть переаттестована). Дисциплины по выбору студента и дисциплины специализации могут выбираться из любых дисциплин, изучаемых в вузе (соседних вузах, организованных на предприятиях базовых кафедрах и т. п.), и утверждаться соответствующими Советами.

19.2.3. Рабочие учебные планы

Отсутствие:

- подробностей о распределении по семестрам и видам занятий аудиторных часов и часов СРС в некоторых формах базовых или индивидуальных учебных планов;

- ❒ названий кафедр, которым планируется проведение занятий по дисциплинам базовых или индивидуальных учебных планов;
- ❒ конкретных дисциплин, выбранных студентами на планируемый учебный год из "Дисциплин по выбору студента",

а также необходимость учета различных дополнительных параметров (например, используемых для расчета преподавательской нагрузки) и допустимых корректировок учебного процесса на планируемый учебный год приводит к необходимости использования еще одного типа документов — рабочих учебных планов.

В ***рабочих учебных планах*** приводится перечень и параметры дисциплин, которые должны изучаться студентами в каждом из семестров указанного учебного года, а также коды (шифры или названия) кафедр, которым поручается проведение занятий по этим дисциплинам. *Некоторые вузы включают в рабочие планы сведения о работах и видах контроля, которые должны выполняться студентами в процессе изучения каждой дисциплины (расчетно-графические работы, контрольные работы и т. п.), т. е.* ***сведения из рабочих программ дисциплин***. Эти и некоторые другие данные необходимы им для расчета преподавательской нагрузки. Однако при использовании более прогрессивных методик расчета нагрузки (например, разработанной в СПбГУ ИТМО) вполне пригодна минимально-возможная форма представления рабочего плана, приведенная на рис. 19.6.

В частном случае набор дисциплин рабочего учебного плана является копией аналогичного набора базового учебного плана в соответствующих семестрах. Однако реальная жизнь иногда требует допустимых корректировок этого набора. Например, при:

- ❒ изменении требований к выпускникам вуза у основных потребителей этих выпускников, а также желание и готовность вуза к немедленной корректировке учебного процесса для удовлетворения таких требований;
- ❒ незапланированных изменениях преподавательского состава (например, временная нетрудоспособность ведущего лектора, приводящая к необходимости сдвига дисциплины на следующий семестр);
- ❒ незапланированных изменениях материально-технической базы (например, временная невозможность использования помещения лаборатории или лабораторных установок).

Эти модификации фиксируются в документах "Рабочие учебные планы", ежегодно составляемых на факультетах по базовым учебным планам для каждого курса, специальности (специализации) по каждой форме обучения. (В некоторых вузах эти планы иногда привязывают не к специальностям, а к каким-либо студенческим потокам.)

19.2.4. Дисциплины по выбору студента и индивидуальные рабочие планы

Напомним, что стандарты отводят достаточно большие объемы часов для ***дисциплин по выбору студента*** по каждому из основных циклов дисциплин (ГСЭ, ЕН, ОПД). Приветствуется включение таких дисциплин и в перечень дисциплин специализации, входящих в цикл специальных дисциплин (СД). На эти часы в каждом из циклов могут планироваться несколько дисциплин, расположенных в одном или нескольких семестрах.

Процедура включения дисциплин по выбору студента в рабочие учебные планы может осуществляться, например, следующим образом. В середине текущего учебного года держатели учебных планов (учебные отделы, деканаты или выпускающие кафедры) опубликовывают "шаблоны" дисциплин по выбору студента по всем направлениям и специальностям на следующий учебный год. Пример такого шаблона (табл. 1) приведен на рис. 19.7.

Заинтересованные кафедры предлагают под эти шаблоны одну или несколько дисциплин с программами и (или) аннотациями, а также распределение часов занятий по видам и по семестрам. Эти дисциплины обсуждаются на Ученых советах факультетов и (или) вуза, где отбираются и утверждаются списки подходящих дисциплин, помещаемых в табл. 2 (рис. 19.8).

Затем утвержденные дисциплины предлагаются студентам, которые должны выбрать из них заданное учебным планом количество подобных дисциплин.

В этот период организуются встречи студентов с авторами предлагаемых дисциплин. В создаваемый рабочий учебный план включаются все дисциплины, выбранные не менее чем, например, восемью студентами (студентам, которые выбрали дисциплины, не попавшие в рабочий план, предлагается повторить выбор, но уже из сокращенного списка). В результате формируется приказ о закреплении за конкретными студентами выбранных ими дисциплин (рис. 19.9).

Кроме этих дисциплин ("по выбору") перечисленные студенты должны изучать общие для них дисциплины федеральной и региональной (вузовской) компоненты, а также дисциплины специализации и факультативы. Все эти дисциплины размещены в рабочих планах (см. например, рис. 19.6), определяющих объемы различных занятий и форм контроля по каждой из них. Назовем такой набор ***индивидуальным рабочим планом***. Он может включать или не включать набор работ и форм контроля, связанных с управлением самостоятельной деятельностью студентов в течение семестра.

16.05.2007 13:24:45 2007/2008 - учебный год

Факультет Компьютерных Технологий И Управления

Кафедра Вычислительной Техники

Рабочий план для специальности/специализации - 230101

Вычислительные машины, комплексы, системы и сети

Группы: 3100,3101,3102,3103 - форма обучения Очная

Семестр 5

Цикл	Ком	Номер	Дисциплина	Отдел	Лек	Лаб	Прак	Срс	Контр
ГСЭ	Ф	02	Физическая культура	ФВиВ	0	0	0	51	Зач
ГСЭ	Р	03	Менеджмент	МЕНЕДЖ	17	0	17	51	Зач
ГСЭ	В	02	Социология	ФИЛ	17	0	17	68	Зач
ОПД	Ф	04	Безопасность жизнедеятельности	ЛТиЭП	17	17	0	60	Зач
ОПД	Ф	08	Организация ЭВМ и систем	ВТ	51	17	0	77	Экз
ОПД	Ф	09	Операционные системы	ВТ	34	17	0	85	Экз
ОПД	Ф	10	Базы данных	ВТ	0	51	0	77	Экз КР
СД	Ф	02	Моделирование	ВТ	34	17	0	77	Экз КР
СД	Р	04	Объектно-ориентированное программирование	ИПМ	0	17	0	32	Экз

Учебных: 918 Аудиторных: 340

Семестр 6

Цикл	Ком	Номер	Дисциплина	Отдел	Лек	Лаб	Прак	Срс	Контр
ГСЭ	Ф	02	Физическая культура	ФВиВ	0	0	0	51	Зач
ГСЭ	Ф	03	Философия	ФИЛ	34	0	17	85	Экз
ГСЭ	В	03	Экономика предприятия и маркетинг	ПЭиМ	17	0	17	51	Зач
ЕН	В	02	Регулярные множества и выражения	ВТ	17	17	0	46	Зач
ОПД	Ф	11	Сети ЭВМ и средства телекоммуникаций	ВТ	51	17	0	85	Экз
ОПД	Р	01	Проектирование приложений к базам данных	ВТ	0	34	0	48	Экз КР
СД	Ф	05.01	Системное программное обеспечение (Язык ассемблера)	ИПМ	17	17	0	42	Зач
СД	Ф	05.02	Системное программное обеспечение (UNIX)	ВТ	17	34	0	34	Экз
СД	Р	02	Периферийные устройства	ВТ	17	17	0	46	Зач
ДН	Ф	01	Практика	ВТ	0	0	0	80	
ДН	Ф	01	Практика	МПБЭВА	0	0	0	28	
ФТД		01	Военная подготовка	ВМК	0	0	0	90	Зач

Учебных: 1026 Аудиторных: 340

Рис. 19.6. Пример рабочего учебного плана (первый вариант)

Перечень шаблонов "Дисциплин по выбору студента" для студентов специальности **Таблица 1**

19.07.00 "ОПТИКО-ЭЛЕКТРОННЫЕ ПРИБОРЫ И СИСТЕМЫ"

на 2003/2004 учебный год

Индекс дисциплины	Название дисциплины	Форма итогового контроля		Объем работы студента (час)			Распределение по курсам и семестрам учебных часов в неделю												Код кафедры
							КУРС												
							1		2		3		4		5		6		
							СЕМЕСТР												
		экзамен	зачет	всего	ауд. занятия	по видам занятий	1	2	3	4	5	6	7	8	9	10	11		
							КОЛИЧЕСТВО НЕДЕЛЬ												
							17	17	17	17	17	17	17	17	17	17	16		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18		19
ГСЭ.В.01 ГСЭ.В.02 ЕН.В.01 ...		5	4 4 5	102 170 102	51 85 0														

Рис. 19.7. Пример шаблона для организации процедуры выбора "Дисциплин по выбору студентов"

Перечень "Дисциплин по выбору студента", предлагаемых для студентов специальности **Таблица 2**

19.07.00 "ОПТИКО-ЭЛЕКТРОННЫЕ ПРИБОРЫ И СИСТЕМЫ"

на 2003/2004 учебный год

Индекс дисциплины	Название дисциплины	Форма итогового контроля		Объем работы студента (час)			Распределение по курсам и семестрам учебных часов в неделю												Код кафедры
							КУРС												
							1		2		3		4		5		6		
							СЕМЕСТР												
		экзамен	зачет	всего	ауд. занятия	по видам занятий	1	2	3	4	5	6	7	8	9	10	11		
							КОЛИЧЕСТВО НЕДЕЛЬ												
							17	17	17	17	17	17	17	17	17	17	16		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18		19
ГСЭ.В.01.01	Этикет		4	102	51	34 17 0 51				2 1 0 3									КЛ
ГСЭ.В.01.02	История и методология науки		4	102	51	51 0 0 51				3 0 0 3									ФИЛ
ГСЭ.В.01.03	Правоведение		4*	102	51	34 0 17 51				2 0 1 3									СОЦ
ГСЭ.В.02.01 ...	Русский язык и культура речи	5	4	170	85	34				1	1								КЛ

Рис. 19.8. Утвержденные списки "Дисциплин по выбору студентов"

… списки студентов второго курса специальности 190700 — Оптико-электронные приборы и системы, записавшихся на изучение дисциплин по выбору на 2003/2004 учебный год

ГСЭ.В.01.01 - Этикет

1. 10001 Бердников Борис Александрович, группа 2300
2. 10002 Васильев Алексей Владимирович, группа 2301
3. 10003 Витенберг Александр Борисович, группа 2300
4. …

ГСЭ.В.01.02 - История и методология науки

1. 10004 Выборнов Александр Григорьевич, группа 2302
2. 10005 Демин Илья Станиславович, группа 2300
3. 10006 Згурский Михаил Леонидович, группа 2301
4. …

ГСЭ.В.01.03 - Правоведение

1. 10007 Кирсанов Даниил Александрович, группа 2300
2. 10008 Кузнецов Николай Анатольевич, группа 2301
3. …

ГСЭ.В.02.01 - Русский язык и культура речи

1. 10003 Витенберг Александр Борисович, группа 2300
2. 10004 Выборнов Александр Григорьевич, группа 2302
3. 10007 Кирсанов Даниил Александрович, группа 2300
4. …

Рис. 19.9. Фрагмент приказа о закреплении за конкретными студентами выбранных ими дисциплин

19.2.5. Студенческие потоки и группы

Студенты вуза объединяются в организационные единицы — студенческие группы, входящие в состав какого-либо факультета. Обычно такие группы закрепляются также за определенными специальностями (специализациями) и (или) выпускающими кафедрами. Однако иногда в группу зачисляют студентов разных специальностей (магистерских программ) и (или) разных выпускающих кафедр.

При чтении лекций обычно формируются лекционные потоки, составленные из любого допустимого (в зависимости от вместимости аудитории) числа групп. Это могут быть группы одной или разных специальностей (например, при чтении лекций по отечественной истории или философии).

Каждая студенческая группа "привязывается" к рабочему учебному плану, в соответствии с которым должно проводиться обучение студентов, включенных в эту группу. Если, например, студенты какой-либо группы должны заниматься по разным планам (они выбрали различные магистерские программы или дисциплины по выбору), то такая группа должна быть привязана к нескольким рабочим учебным планам, а конкретные студенты группы — к одному из этих планов (см. *главу 21*).

13.07.2004 13:46:46 2004/2005 - учебный год

Факультет Компьютерных Технологий И Управления

Кафедра Вычислительной Техники

Рабочий план для специальности/специализации - 220100

Вычислительные машины, комплексы, системы и сети

Группы: 3100,3101,3102,3103,3104 - форма обучения Очная

Семестр 5

Индекс	Дисциплина	Вид раб. /контр.	№	Часов	Отдел
ГСЭ. Ф.02	Физическая культура	СРС		51	ФВиВ
		Зач			
		Аттест	1		
		Аттест	2		
ГСЭ. Р.03	Менеджмент	Лек		17	МЕНЕДЖ
		Прак		17	
		СРС		51	
		Зач			
		Аттест	1		
		Аттест	2		
ГСЭ. В.02	Политология	Лек		17	СОЦИОЛ
		Прак		17	
		СРС		68	
		Зач			
		Аттест	1		
		Аттест	2		
ОПД. Ф.04	Безопасность Жизнедеятельности	Лек		17	ЛТиЭП
		Лаб		17	
		СРС		60	
		Зач			
		Аттест	1		
		Аттест	2		
ОПД. Ф.08	Организация ЭВМ и систем	Лек		51	ВТ
		Лаб		17	
		СРС		77	
		Экз			
		Аттест	1		
		Аттест	2		
ОПД. Ф.09	Операционные системы	Лек		34	ВТ
		Лаб		17	
		СРС		85	
		Экз			
		Аттест	1		
		Аттест	2		
ОПД. Ф.10	Базы данных	Лаб		51	ВТ
		СРС		77	
		Экз			
		КР			
		Контр	1		
		Контр	2		
СД. Ф.02	Моделирование	Лек		34	ВТ
		Лаб		17	
		СРС		77	
		Экз			
		Аттест	1		
		Аттест	2		
СД. Р.04	Объектно-Ориентированное программирование	Лаб			ИПМ
		СРС		32	
		Экз			
		Аттест	1		
		Аттест	2		

Семестр 6

Индекс	Дисциплина	Вид раб. /контр.	№	Часов	Отдел
ГСЭ. Ф.02	Физическая культура	СРС		51	ФВиВ
		Зач			
		Аттест	1		
		Аттест	2		
ГСЭ. Ф.03	Философия	Лек		34	ФИЛ
		Прак		17	
		СРС		85	
		Экз			
		Аттест	1		
		Аттест	2		
ГСЭ. В.03	Экономика предприятий и маркетинг	Лек		17	ПЭиМ
		Прак		17	
		СРС		51	
		Зач			
		Аттест	1		
		Аттест	2		
ЕН. В.02	Регулярные множества и выражения	Лек		17	ВТ
		Лаб		17	
		СРС		46	
		Зач			
		Аттест	1		
		Аттест	2		
ОПД. Ф.11	Сети ЭВМ и средства телекоммуникаций	Лек		51	ВТ
		Лаб		17	
		СРС		85	
		Экз			
		Аттест	1		
		Аттест	2		
ОПД. Ф.10	Проектирование приложений к базам данных	Лаб		34	ВТ
		СРС		48	
		Экз			
		КР			
		Контр	1		
		Контр	2		
ОПД. Ф.11	Сети ЭВМ и средства телекоммуникаций	Лек		51	ВТ
		Лаб		17	
		СРС		85	
		Экз			
		Аттест	1		
		Аттест	2		
СД.Ф. 05.01	Системное программное обеспечение (Язык ассемблера)	Лек		34	ИПМ
		Лаб		17	
		СРС		77	
		Экз			
		Аттест	1		
		Аттест	2		
СД.Ф. 05.02	Системное программное обеспечение (UNIX)	Лек		34	ВТ
		Лаб		17	
		СРС		77	
		Экз			
		Аттест	1		
		Аттест	2		
ДН.Ф	Практика	СРС		109	ВТ
ФТД. 01	Военная подготовка	СРС		90	ВМК
		Зач			

Рис. 19.10. Пример рабочего учебного плана (второй вариант)

19.2.6. Еще о рабочих учебных планах

На рис. 19.6 приведен один из вариантов представления рабочего учебного плана, в котором, как и в учебных планах, присутствуют только итоговые виды контроля. Но так как по ряду дисциплин студентам планируется выполнение различных видов самостоятельных работ (домашние задания, расчетно-графические работы, курсовые работы или проекты и т. п.), то их также желательно включать в рабочие учебные планы.

Поэтому целесообразно иметь такую "резиновую" форму рабочего учебного плана, в которую можно было бы включать любое количество любых видов работ, связанных с изучением какой-либо дисциплины, и любое количество разнообразных видов текущего и итогового контроля качества усвоения ее материала.

Возможный вариант такого плана показан на рис. 19.10. В нем указано только два (из множества) текущих видов контроля: аттестация (Аттест) и контрольная работа (Контр), а также курсовая работа (КР). Так как по одной дисциплине может планироваться проведение нескольких контрольных работ (аттестаций) и даже экзаменов, то для оценивания качества выполнения каждой из них они в плане нумеруются (колонка "№").

Наверное, можно предложить более красивые представления подобного рабочего учебного плана, но здесь нам важно поточнее определить ту информацию, которая должна быть в нем размещена и, следовательно, должна содержаться в создаваемой базе данных.

Глава 20

Построение инфологической модели "Учебные планы"

20.1. Первая попытка проектирования

Легко заметить, что создаваемые в вузе учебные планы: базовые *(разд. 19.2.1 и 19.2.2)* и рабочие *(разд. 19.2.3—19.2.6)*, а также их модификации, предназначенные для планирования подготовки бакалавров, магистров, аспирантов и даже организации приема абитуриентов, очень похожи друг на друга. Во всех существует "шапка", в которой указывается:

- тип стандарта (направление подготовки дипломированных специалистов, специальность, специализация, направление подготовки бакалавров, направление подготовки магистров, магистерская программа, аспирантура);
- дата утверждения стандарта или учебного плана (в стандартах 1995 и 2000 гг. при одинаковых номерах и названиях, достаточно много различий как в именах и объемах дисциплин, так и в отнесении их к рангу федеральных, региональных и т. д.);
- номер и название направления, специальности (программы) и специализации;
- квалификация, получаемая после успешного окончания обучения (инженер, инженер-педагог, экономист, бакалавр техники или магистр техники и технологии и т. п.);
- учебный год, курс, факультет, выпускающая кафедра, форма обучения (очная, очно-заочная, заочная) и ряд других атрибутов.

Далее идет набор снабженных индексом дисциплин, по каждой из которых приводится информация о формах итогового контроля, числу часов, отводимых на различные виды занятий, и другая информация (см. рис. 19.3 и 19.6).

Примерная структура этих планов показана на рис. 20.1.

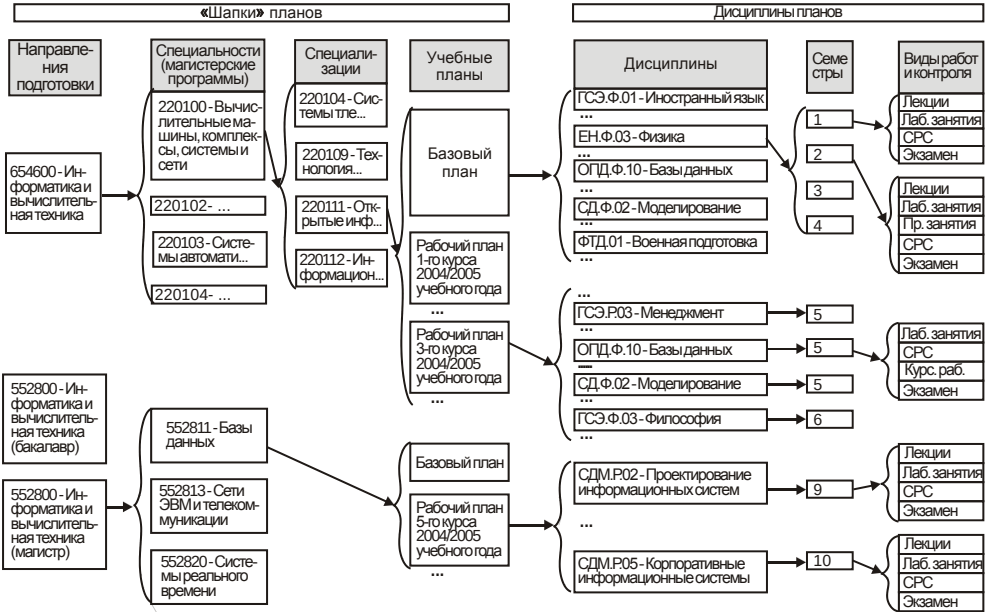


Рис. 20.1. Примерная структура учебных планов

Попробуем создать модель, состоящую всего из двух сущностей: `Шапки планов` и `Дисциплины планов`.

На рис. 20.2 показаны атрибуты и ряд записей одного из возможных вариантов сущности `Шапки планов`.

Даже при беглом рассмотрении данных, содержащихся в ее записях, видны недостатки такого проекта — *избыточность хранимых данных*. Например, здесь в четырех записях четыре раза повторяются достаточно длинные названия факультета, направления подготовки и выпускающей кафедры, по два раза повторяются названия специальности и специализации. А сколько бы было "лишних" данных в информационной системе СПбГУ ИТМО, где хранится не три, а несколько тысяч планов, если бы в ней был реализован рассматриваемый вариант?

При этом не так страшен расход памяти или снижение производительности системы при обработке больших объемов данных, страшно то, что такой проект практически не позволяет поддерживать, так называемую, *целостность* данных — их правильность на любой момент времени. Это связано с проблемами, возникающими при изменении данных (их вводе, удалении и обновлении).

Атрибуты	Записи				
Идентификатор (ИД)	1	2	3	4	6
Тип плана	Базовый	Рабочий	Базовый	Рабочий	...
Тип стандарта	Направление подготовки дипломированных специалистов	Направление подготовки дипломированных специалистов	Направление подготовки магистров	Направление подготовки магистров	...
Номер направления	654600	654600	552800	552800	...
Название направления	Информатика и вычислительная техника	Информатика и вычислительная техника	Информатика и вычислительная техника	Информатика и вычислительная техника	...
Дата утверждения	15.03.2000	01.03.2004	15.03.2000	01.03.2004	...
Квалификация (Академическая степень)	Инженер		Магистр		...
Номер специальности (магистерской программы)	220100	220100	552811	552811	...
Название специальности (магистерской программы)	Вычислительные машины, комплексы, системы и сети	Вычислительные машины, комплексы, системы и сети	Базы данных	Базы данных	...
Номер специализации	220111	220111			...
Название специализации	Открытые информационно-вычислительные системы	Открытые информационно-вычислительные системы			...
Форма обучения	Очная	Очная	Очная	Очная	...
Курс		5		5	...
Учебный год		2004/2005		2004/2005	...
Факультет	Компьютерных технологий и управления	Компьютерных технологий и управления	Компьютерных технологий и управления	Компьютерных технологий и управления	...
Выпускающая кафедра	Вычислительной техники	Вычислительной техники	Вычислительной техники	Вычислительной техники	...
Группы		5100, 5103		5110	

Рис. 20.2. Атрибуты и несколько записей сущности `Шапки планов`

Так, если потребуется изменить на новое название факультета "Компьютерных технологий и управления", то где гарантия, что нам удастся найти и обновить все записи? Как, например, сможет быть найдена и обновлена третья запись рис. 20.1 при поиске "старого" названия факультета, если в ней хранится ошибочное название — "Компьютерных техналогий и управления". Схожие проблемы возникнут и при обновлении или удалении других, многократно повторяющихся, текстовых значений атрибутов записей. (Обратите внимание на то, что на рис. 20.2 нет ошибок только в первой записи.)

Об *аномалиях обновления, включения и удаления* данных подробно рассказано в [1, 4]. Здесь же отметим, что многие проблемы исчезнут, если выделить в отдельные сущности (справочники) сведения о факультетах, кафедрах,

направлениях, специальностях и др. Такое разбиение большой сущности на несколько сущностей, обладающих лучшими свойствами при включении, изменении и удалении данных, называется *нормализацией* (см. *главу 11*). Окончательная цель нормализации сводится к получению такого проекта базы данных, в котором ***каждый факт появляется лишь в одном месте,*** т. е. исключена избыточность информации.

20.2. Вторая попытка проектирования

20.2.1. Шапки планов

Сначала создадим справочники, показанные на рис. 20.3 и 20.4.

Теперь обратим внимание на то, что три первых столбца структуры рис. 20.1 содержат однотипную информацию и, следовательно, их целесообразно объединить в единую сущность, организовав иерархические связи между некоторыми значениями этой сущности.

ИД	Наименование
	Квалификация
1	Инженер
2	Бакалавр техники и технологии
3	Магистр техники и технологии
4	Инженер-педагог
6	Менеджер
7	Инженер (математик)
8	Экономист
9	Магистр математики
10	Магистр физики
11	Бакалавр математики
	Формы обучения
1	Очная
2	Очно-заочная (вечерняя)
3	Заочная

ИД	Наименование
	Тип стандартов
1	Направление подготовки дипломированных специалистов
2	Направление подготовки бакалавров
3	Направление подготовки магистров
4	Специальность
5	Специализация
6	Магистерская программа
7	Аспирантура
	Типы планов
1	Примерный
2	Базовый
3	Рабочий
4	Переходный
5	Индивидуальный

Рис. 20.3. Атрибуты и записи справочников: `Квалификации`, `Формы обучения`, `Типы стандартов` и `Типы планов`

С целью упрощения описания некоторых особенностей структуры данной сущности, в ее представление на рис. 20.5 включено два лишних столбца (отмечены серым цветом). В них помещены те значения атрибутов `Номер` и `Наименование` справочника `Напр_спец`, которые расположены в строках с идентификатором (`ИД`), равным соответствующему значению атрибута `НС_ИД` сущности `Направления_специал`.

Направления, специальности и специализации (Напр_спец)		
ИД	Код_напрспец	Наименование
...		
5	552811	Базы данных
6	072301	Биомедицинская оптика
7	220114	Вычислительные машины и сети
8	220100	Вычислительные машины, комплексы, системы и сети
...		
16	552800	Информатика и вычислительная техника
17	654600	Информатика и вычислительная техника
18	030506	Информатика, вычислительная техника и компьютерные технологии
19	220112	Информационно-управляющие системы
...		
66	220111	Открытые информационно-вычислительные системы
...		
89	552813	Сети ЭВМ и телекоммуникации
90	510211	Системное программирование
91	220306	Системы автоматизированного проектирования
92	552820	Системы реального времени
93	220104	Системы телекоммуникаций и компьютерной безопасности
...		
109	220109	Технология разработки программных систем
...		
146	071900	Информационные системы в экономике

Рис. 20.4. Атрибуты и ряд записей справочника `Напр_спец`, содержащего данные о направлениях, специальностях и специализациях

ИД	Уровень	Дата ГОС	НАПС_ИД	НС_ИД	№ напр./спец.	Название напр./спец.	ТС_ИД	Квал_ИД
113	3	01.03.1995		8	220100	Вычислительные машины, комплексы, системы и сети	4	1
1047	3	01.03.1995	113	93	220104	Системы телекоммуникаций и компьютерной безопасности	5	1
187	3	01.03.1995	113	109	220109	Технология разработки программных систем	5	1
125	3	01.03.1995	113	66	220111	Открытые информационно-вычислительные системы	5	1
134	3	01.03.1995	113	19	220112	Информационно-управляющие системы	5	1
68	3	01.03.1995		16	552800	Информатика и вычислительная техника	3	3
143	3	01.03.1995	68	5	552811	Базы данных	6	3
144	3	01.03.1995	68	89	552813	Сети ЭВМ и телекоммуникации	6	3
145	3	01.03.1995	68	92	552820	Системы реального времени	6	3
58	3	01.03.2000		17	654600	Информатика и вычислительная техника	1	1
700	3	01.03.2000	58	8	220100	Вычислительные машины, комплексы, системы и сети	4	1
1251	3	01.03.2000	700	93	220104	Системы телекоммуникаций и компьютерной безопасности	5	1
792	3	01.03.2000	700	109	220109	Технология разработки программных систем	5	1
1250	3	01.03.2000	700	66	220111	Открытые информационно-вычислительные системы	5	1
1249	3	01.03.2000	700	19	220112	Информационно-управляющие системы	5	1
904	2	01.03.2000		16	552800	Информатика и вычислительная техника	2	2
74	3	01.03.2000		16	552800	Информатика и вычислительная техника	3	3
1257	3	01.03.2000	74	5	552811	Базы данных	6	3
1258	3	01.03.2000	74	89	552813	Сети ЭВМ и телекоммуникации	6	3
1259	3	01.03.2000	74	92	552820	Системы реального времени	6	3

Рис. 20.5. Сущность `Направления_специал`

В Направления_специал вместо текстовых значений типов стандартов, квалификаций и др. размещены указатели (ТС_ИД, Квал_ИД) на идентификаторы соответствующих значений из справочников. Следовательно, теперь ошибка в тексте одной строки справочника может привести к появлению ошибок в текстах множества документов, использующих этот справочник, но все они одновременно "исправляются" при исправлении единственной строки справочника.

В Направления_специал включены также новые атрибуты: Уровень и НАПС_ИД.

Первый используется для указания уровня высшего профессионального образования: 1 — незаконченное высшее (не менее двух лет обучения в вузе), 2 — образовательная программа бакалавра (4 года), 3 — образовательная программа специалиста (5—5,5 лет) или магистра (6 лет), 4 — послевузовское образование (аспирантура, докторантура).

Второй атрибут, являющийся указателем на идентификатор строк (ИД) собственной сущности, служит для организации цепочек подчиненности:

специализация → специальность →
→ направление подготовки дипломированных специалистов

или

магистерская программа → направление подготовки магистров.

Наличие такой цепочки позволяет указывать в рабочем учебном плане только номер специальности (см. рис. 19.6 и 19.10), специализации или магистерской программы, по которым производится определение типа стандарта, направления подготовки и квалификации. Например, возьмем рабочий учебный план, составленный на 2004/2005 учебный год для студентов 3 курса специальности 220100 (см. рис. 19.6 или 19.10). Эти студенты поступили в ИТМО в 2002 году и обучаются по рабочим учебным планам, составляемым по ГОС 2000 года. В записях сущности Направления_специал (рис. 20.5) специальность 220100 с датой ГОС, равной, 01.03.2000, имеет ИД=700 и значение НАПС_ИД=58, т. е. эта специальность входит в состав направления, основные параметры которого приведены в записи с идентификатором 58:

Уровень = 3 (законченное высшее образование)

Дата ГОС = 01.03.2000

НАПС_ИД — пусто (никому не подчиняется)

ТС_ИД = 1 (направление подготовки дипломированных специалистов)

НС_ИД = 17 (654600 — Информатика и вычислительная техника)

Квал_ИД = 1 (инженер)

Можно также определить те специализации, по которым они смогут обучаться на старших курсах. Для этого надо найти строки с НАПС_ИД=700, т. е. под-

чиненные строке с ИД=700. В этих строках будут перечислены специализации 220104, 220109, 220111 и 220112 (их названия указаны на рис. 20.4 или 20.5).

Наконец, в учебных планах, так или иначе, упоминаются факультеты и выпускающие кафедры, составившие эти планы. Так как кроме этих структур в ИТМО существуют и другие (отделы, центры и пр.) и могут появляться новые, то целесообразно создать единую сущность для хранения сведений о любых подразделениях, назвав ее, например, Отделы (рис. 20.6).

ИД	Короткое имя	ОТД_ИД	Наименование
101	СУиИ	703	кафедра систем управления и информатики
102	ВТ	703	кафедра вычислительной техники
103	ИТиКТ	705	кафедра измерительных технологии и компьютерной томографии
…			
107	ИКГ	705	кафедра инженерной и компьютерной графики
108	ПКС	703	кафедра проектирования компьютерных систем
109	ИНС	703	кафедра информационно-навигационных систем
110	ИЯ	706	кафедра иностранных языков
111	ИПМ	703	кафедра информатики и прикладной математики
…			
209	ВМ	704	кафедра высшей математики
210	ФИЗИКА	704	кафедра физики
211	ТТОЭ	702	кафедра твердотельной оптоэлектроники
212	КТ	717	кафедра компьютерных технологий
…			
400	Ректорат	730	ректорат
…			
701	ОИСТ	777	факультет оптико-информационных систем и технологий
702	ИФ	777	факультет инженерно-физический
703	КТиУ	777	факультет компьютерных технологий и управления
704	ЕН	777	факультет естественнонаучный
705	ТМиТ	777	факультет точной механики и технологий
706	Гум	777	факультет гуманитарный
…			
717	ИТиП	777	факультет информационных технологий и программирования
…			
730	АХЧ	777	административно-хозяйственная часть
…			
777	СПбГУИТМО		Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Рис. 20.6. Атрибуты и несколько записей сущности Отделы

В сущности Отделы, в отличие от большинства других сущностей, идентификатор (ИД) не является порядковым номером строки, а имеет исторически сложившееся значение. Для новых отделов этот трехзначный номер назначает администратор базы данных.

Атрибут ОТД_ИД сущности Отделы используется так же, как и атрибут НАПС_ИД в сущности Направления_специал, т. е. для организации цепочек подчиненности (например, кафедра → факультет → университет) неограниченной вложенности.

Воспользуемся еще одним универсальным приемом, с помощью которого можно без изменения структуры справочника неограниченно расширять набор свойств его значений.

Так, если какому-нибудь отделу надо дать статус факультета, право подготовки приказов о приеме, переводе и отчислении студентов, а также на проведение ряда финансовых операций, то его описание на рис. 20.6 должно быть дополнено тремя атрибутами, например такими, которые указаны в строках 1, 4 и 12 таблицы, приведенной на рис. 20.7.

ИД	Наименование	Примечание
1	Может издавать студ. приказы	Может готовить проекты приказов по движению контингента студентов
2	Университет	Университет
3	Кафедра	Отдел является кафедрой
4	Факультет	Отдел является факультетом
5	Техникум	Рассматривается как отдельная структура
6	Центр	Отдел является центром
7	Библиотека	Библиотека
8	Межкафедральный центр	Межкафедральный центр
9	Прочий УВП	Прочий УВП
10	ИКВО	Институт комплексного военного обучения
11	ВЦ	Вычислительный центр
12	Может принимать деньги от контрактных студентов	Может принимать деньги от контрактных студентов

Рис. 20.7. Атрибуты и записи сущности Свойства отделов

Возможно, что появится необходимость дополнить описание этого отдела еще рядом атрибутов, т. е. придется еще раз изменять структуру сущности Отделы, что, как и в первый раз, может привести к необходимости изменения всех приложений, в которых используется эта сущность. Поэтому *следует добиваться создания такого проекта базы данных, который бы при ее эксплуатации позволял вносить любые изменения только с помощью изменения данных, но не структуры базы данных*.

Здесь это достигается созданием двух дополнительных сущностей: Свойства отделов и Характеристики отделов (рис. 20.8).

СВОТД_ИД	ОТД_ИД
...	
3	102
...	
4	701
12	701
1	702
4	702
12	702
1	703
4	703
12	703
...	

Рис. 20.8. Атрибуты и несколько записей сущности `Характеристики отделов`

Последняя сущность позволяет привязать к любому отделу любой набор свойств. Поэтому для включения в описание отдела нового свойства не потребуется создавать новый атрибут в сущности `Отделы`, а нужно только внести описание нового свойства в `Свойства отделов` и дополнить `Характеристики отделов` описанием связи этого свойства с нужным отделом.

В завершение описания структуры хранения данных о `Шапках планов`, добавим сущности: `Планы` (рис. 20.9) и `Группы планов` (рис. 20.10).

ИД	ТПЛ_ИД	Учебный год	ОТД_ИД (Факультет)	ОТД_ИД_ЗАКРЕПЛЕН_ЗА (Кафедра)	НАПС_ИД	Курс	ФО_ИД	Дата утверждения	ПЛАН_ИД	ПЛАН_ИД ОСНОВАНИА
...										
813	3	2001/2002	703	102	700	1	1	24.05.2001		
1435	3	2002/2003	703	102	700	2	1	01.03.2002	813	
1723	3	2003/2004	703	102	700	3	1	01.03.2003	1435	
2145	3	2004/2005	703	102	700	4	1	01.03.2004	1723	
...										
1427	3	2002/2003	703	102	700	1	1	01.03.2002		
1726	3	2003/2004	703	102	700	2	1	01.03.2003	1427	
2144	3	2004/2005	703	102	700	3	1	01.03.2004	1726	
...										
962	3	2002/2003	703	111	792	1	1	01.03.2002		
1727	3	2003/2004	703	111	792	2	1	01.03.2003	962	
2156	3	2004/2005	703	111	792	3	1	01.03.2004	1727	
...										

Рис. 20.9. Атрибуты и несколько записей сущности `Планы`

Большинство студентов в процессе обучения в университете последовательно изучают наборы дисциплин, указанных в рабочих учебных пла-

нах 1, 2, 3 и т. д. курсов (например, наборы дисциплин из планов с ИД 813, 1435, 1723 и т. д. рис. 20.9). Если студент, начавший заниматься по этой цепочке планов, ушел, например, на третьем курсе, на один год в академический отпуск, то, возвратившись, он должен будет продолжать обучение по другому плану третьего курса (ИД=2144). Не исключено, что наборы дисциплин новой цепочки планов с ИД 1427 и 1726 не совпадают с теми, которые он изучал до отпуска по планам с ИД 813 и 1435. А так как при окончании обучения набор всех изученных им дисциплин должен соответствовать набору дисциплин новой цепочки планов, то при отсутствии такого соответствия (по названию, объему и т. п.) необходимо принять меры к его устранению.

Для организации указанных ранее цепочек в сущность Планы включен атрибут ПЛАН_ИД, являющийся указателем на идентификатор (ИД) предшествующего рабочего плана.

Кроме того, в Планы включен атрибут ПЛАН_ИД_ОСНОВАН_НА для организации связи базового и построенных на его основе рабочих учебных планов.

Поступили на кафедру ВТ в 2001 году

ГРУППА	ПЛАН_ИД
150	813
151	813
152	813
153	813
250	1435
251	1435
252	1435
253	1435
3100	1723
3101	1723
3102	1723
3103	1723
3104	1723
4100	2145
4101	2145
4102	2145
4103	2145
4104	2145

Поступили на кафедру ВТ в 2002 году

ГРУППА	ПЛАН_ИД
150	1427
151	1427
152	1427
153	1427
2100	1726
2101	1726
2102	1726
2103	1726
3100	2144
3101	2144
3102	2144
3103	2144
3104	2144

Поступили на кафедру ИПМ в 2002 году

ГРУППА	ПЛАН_ИД
154	962
2120	1727
3120	2156

Рис. 20.10. Атрибуты и несколько записей сущности Группы планов

По любому рабочему учебному плану обучается одна или несколько студенческих групп. Так по плану с ИД 2144 (см. рис. 20.9) в 2004/2005 учебном году будет заниматься пять групп с 3100 по 3104 (см. рис. 20.10), а по плану 2156 — только одна группа 3120.

20.2.2. Дисциплины планов

Овладев некоторыми приемами создания удовлетворительных моделей, сразу перейдем к построению многотабличной модели, описывающей дисциплины планов.

Как показывает анализ содержимого *главы 19*, где приводились описания и внешний вид различных учебных планов, структура их строк нерегулярна и достаточно сложна. Кроме того, в них достаточно много повторяющихся текстовых данных, которые целесообразно разместить в справочниках: `Циклы дисциплин`, `Компоненты` и `Дисциплины` (рис. 20.11—20.13).

На рис. 20.13 показано только 16 из полутора тысяч дисциплин. Это те дисциплины, которые входят в рабочий учебный план, приведенный на рис. 19.6.

ИД	Аббревиатура	Наименование
1	ГСЭ	Общие гуманитарные и социально-экономические дисциплины
2	ЕН	Общие математические и естественнонаучные дисциплины
3	ОПД	Общепрофессиональные дисциплины
4	СД	Специальные дисциплины, включая дисциплины специализации
5	ФТД	Факультативы
6	ДНМ	Дисциплины направления специализированной подготовки
7	СДМ	Специальные дисциплины магистерской подготовки
8	НИРМ	Научная (научно-исследовательская и (или) научно-педагогическая) работа магистра
9	ИГАМ	Итоговая государственная аттестация
10	ДН	Цикл дисциплин направления
11	ДС	Цикл дисциплин специализации
12	ДФ	Цикл факультативных дисциплин
13	НИР	Научно-исследовательская работа
14	ОД	Цикл дисциплин отраслевой подготовки

Рис. 20.11. Атрибуты и записи сущности `Циклы дисциплин`

ИД	Аббревиатура	Наименование
1	Ф	Федеральный компонент
2	Р	Национально-региональный (вузовский) компонент
3	В	Выбор студента
4	НИР	Научно-исследовательская работа
5	ФТД	Факультативы
6	.	Другие
7	С	Специализация

Рис. 20.12. Атрибуты и записи сущности `Компоненты`

ИД	Короткое имя	Наименование
...		
574	Базы данных	Базы данных
576	Безопасн_ жизнед	Безопасность жизнедеятельности
598	Военная подготовка	Военная подготовка
314	Менеджмент	Менеджмент
386	Моделирование	Моделирование
157	Объект.ор.прогр.	Объектно-ориентированное программирование
164	Операц. системы	Операционные системы
199	Орган.ЭВМ и сист	Организация ЭВМ и систем
34	Перифер. устр-ва	Периферийные устройства
40	Политология	Политология
52	Практика	Практика
100	Проект.прил.к БД	Проектирование приложений к базам данных
1619	Рег.множ.и выражения	Регулярные множества и выражения
901	Сети ЭВМ и ср.ТК	Сети ЭВМ и средства телекоммуникаций
911	Сис.прогр.UNIX	Системное программное обеспечение (UNIX)
1182	Сис.пр.об.Яз.Ассемб.	Системное программное обеспечение (Язык ассемблера)
862	Физическая культура	Физическая культура
867	Философия	Философия
681	Экон_предпр_марк	Экономика предприятия и маркетинг
...		

Рис. 20.13. Атрибуты и несколько записей сущности `Дисциплины`

ИД	КОМ_ИД	ЦД_ИД	Номер в цикле	ДИС_ИД	Номер дисциплины по выбору	ПЛАН_ИД	Наименование дисциплины (из справочника рис. 1.19)
26163	1	1	02	862		2144	Физическая культура
26164	1	1	03	867		2144	Философия
26165	2	1	03	314		2144	Менеджмент
26166	3	2	02	1619	1	2144	Регулярные множества и выражения
26167	1	3	04	576		2144	Безопасность жизнедеятельности
26168	1	3	08	199		2144	Организация ЭВМ и систем
26169	1	3	09	164		2144	Операционные системы
26170	1	3	10	574		2144	Базы данных
26171	1	3	11	901		2144	Сети ЭВМ и средства телекоммуникаций
26172	2	3	01	100		2144	Проектирование приложений к базам данных
26173	1	4	02	386		2144	Моделирование
26174	1	4	05.02	911		2144	Системное программное обеспечение (UNIX)
26175	2	4	04	157		2144	Объектно-ориентированное программирование
26176	2	4	02	34		2144	Периферийные устройства
26177	5	5	01	598		2144	Военная подготовка
26178	3	1	02	40		2144	Политология
26179	1	4	05.01	1182		2144	Системное программное обеспечение (Язык ассемблера)
26180	3	1	03	681		2144	Экономика предприятия и маркетинг
26181	1	10	01	52		2144	Практика

Рис. 20.14. Атрибуты и несколько записей сущности `Строки планов`

Перейдем теперь к построению модели строк планов (см. рис. 19.4, 19.6 и 19.9). В каждой строке плана кроме индекса и названия дисциплины существует один или несколько (по числу семестров) наборов значений для различных видов занятий и контроля. А так как мы договорились использовать реляционную СУБД (см. *предисловие*), а в реляционной модели поля могут содержать только одно значение или ничего (множественные поля недопустимы) (см. *разд. 3.2*), то нам придется использовать для описания дисциплины несколько взаимосвязанных сущностей. Назовем первую из них Строки планов (рис. 20.14).

В ней помещен указатель на сущность Планы (ПЛАН_ИД), к которому относятся данные строки. Размещены также указатели на справочники Компоненты (КОМ_ИД), Циклы дисциплин (ЦД_ИД) и Дисциплины (ДИС_ИД). Проставлены номера в цикле и номер дисциплины по выбору (см. *разд. 19.2.4*), входящие в состав индекса дисциплины. Для упрощения описания некоторых особенностей структуры данной сущности, в ее представление на рис. 20.14 включен лишний столбец с наименованием дисциплины (отмечен серым цветом).

Информацию о распределении дисциплины по семестрам, о кафедрах, которым поручено проводить обучение по этим дисциплинам в каждом из семестров, а также о числе недель, отведенных в этих семестрах на теоретическое обучение, разместим в сущности Элементы строк, показанной на рис. 20.15.

В сущности Элементы строк помещены указатели на идентификаторы строк (ИД) сущности Строки планов (СПЛ_ИД) и сущности Отделы (ОТД_ИД). Кроме них включены атрибуты Номер семестра и Недель, где указываются номера семестров, в которых запланировано изучение дисциплины, и количество недель, отведенных на теоретическое изучение дисциплины в каждом из этих семестров.

Наконец, по каждой дисциплине в каждом семестре планируется определенный набор работ (лекции, лабораторные занятия и пр.) и видов текущего и (или) итогового контроля за усвоением материала дисциплины. Для хранения этих наборов создадим еще одну сущность Содержания элементов строк (рис. 20.16), а также справочник Виды работ (рис. 20.17).

В сущности Содержания элементов строк помещены указатели на идентификаторы строк (ИД) сущности Элементы строк (ЭСТ_ИД) и сущности Виды работ (ВР_ИД). Кроме них включены атрибуты Объем и Номер контроля, в которых указывается число часов, отводимых на соответствующие виды работ, и номер указанного вида контроля (см. *разд. 19.2.9*).

В справочнике Виды работ размещены атрибуты Наименование, содержащий полные наименования вида занятий или контроля, Аббревиатура — с кратким наименованием, используемым в текстах учебных планов, Единица измерения и Порядок, который используется для упорядочения различных меню приложений.

ИД	ОТД_ИД	Номер семестра	СПЛ_ИД	Недель
33135	302	5	26163	17
33136	302	6	26163	17
33137	503	6	26164	17
33138	309	5	26165	17
33139	102	6	26166	17
33140	310	5	26167	17
33141	102	5	26168	17
33142	102	5	26169	17
33143	102	5	26170	17
33144	102	6	26171	17
33145	102	6	26172	17
33146	102	5	26173	17
33147	102	6	26174	17
33148	111	5	26175	17
33149	102	6	26176	17
33150	112	6	26177	17
33151	509	5	26178	17
33152	111	6	26179	17
33153	507	6	26180	17
33154	102	6	26181	17
33155	511	6	26181	17

Рис. 20.15. Атрибуты и несколько записей сущности `Элементы строк`

ИД	ВР_ИД	Объем	Номер контроля	ЭСТ_ИД
...				
121949	2	51,00		33143
121950	4	77,00		33143
121951	5		1	33143
121952	7		1	33143
121953	1	51,00		33144
121954	2	17,00		33144
121956	4	85,00		33144
121957	5		1	33144
...				

Рис. 20.16. Атрибуты и несколько записей сущности `Содержания элементов строк`

Напомним, что в базовых (см. рис. 19.4) и рабочих (см. рис. 19.6) учебных планах приводятся объемы работ в "часах в неделю" и "часах в семестр" соответственно. Для учета такого различия при создании процедур приложений созданы две сущности: `Свойства видов работ` (рис. 20.18) и `Характеристики видов работ` (рис. 20.19), которые по своей структуре и предназначению схожи с аналогичными сущностями `Свойства отделов` и `Характеристики отделов`, описанными в *разд. 20.2.1*.

ИД	Порядок	Аббре-виатура	Наименование	Единица измерения
0	13	Нет	Отсутствие контроля	
1	2	Лек	Лекции	час
2	4	Лаб	Лабораторные занятия	час
3	6	Прак	Практические занятия	час
4	8	СРС	Самостоятельная работа студента	час
5	9	Экз	Экзамен	
6	10	Зач	Зачет	
7	11	КР	Курсовая работа	
8	12	КП	Курсовой проект	
9	1	Лек/н	Лекций в неделю	час
10	3	Лаб/н	Лабораторных занятий в неделю	час
11	5	Прак/н	Практических занятий в неделю	час
12	7	СРС/н	Самостоятельной работы студента в неделю	час
13	0	Ауд/н	Аудиторных занятий в неделю	час
14	14	Тест	Тестирование	
15	15	Экзамен	Экзамен	
16	16	Олимп.	Олимпиада	
17	17	ЕГЭ	ЕГЭ	
18	18	Рег.олим	Региональная олимпиада	

Рис. 20.17. Атрибуты и записи сущности `Виды работ`

ИД	Наименование	Примечание
1	Аудиторные виды занятий	
2	Самостоятельная работа	
3	Полная работа (ауд+самост)	
4	Межсессионный контроль	
5	Сессионный контроль	
6	Иметь объем	
7	Иметь номер контроля	
8	Долгосрочный вид работы	Имеют виды работ использующиеся в базовом учебном плане
9	Краткосрочный вид работы	Имеют виды работ использующиеся в рабочем учебном плане

Рис. 20.18. Атрибуты и записи сущности `Свойства видов работ`

СВР_ИД	ВР_ИД
1	1
6	1
9	1
...	
2	4
6	4
9	4

Рис. 20.19. Атрибуты и несколько записей сущности `Характеристики видов работ`

20.3. Инфологическая модель "Учебные планы"

На рис. 20.20 приведена ER-диаграмма "Учебные планы", построенная по материалам *разд. 20.2*.

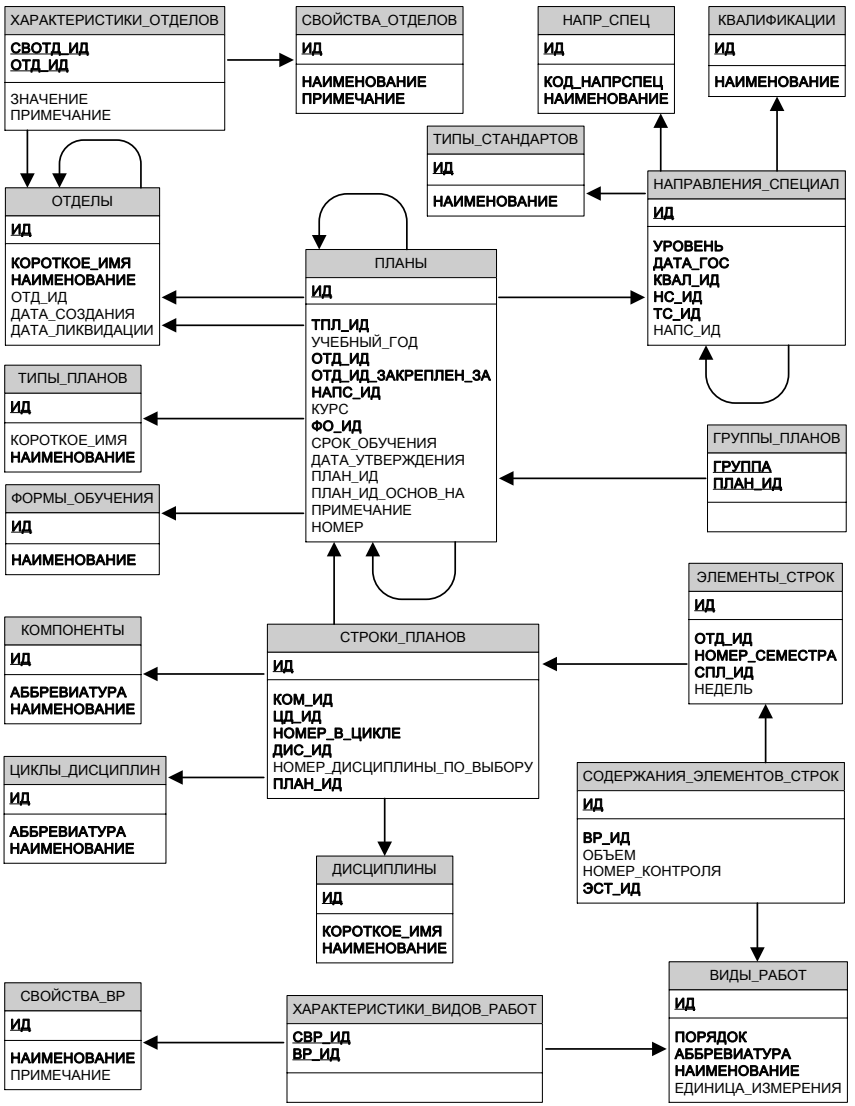


Рис. 20.20. ER-диаграмма "Учебные планы"

Глава 21

"Итоговая успеваемость"

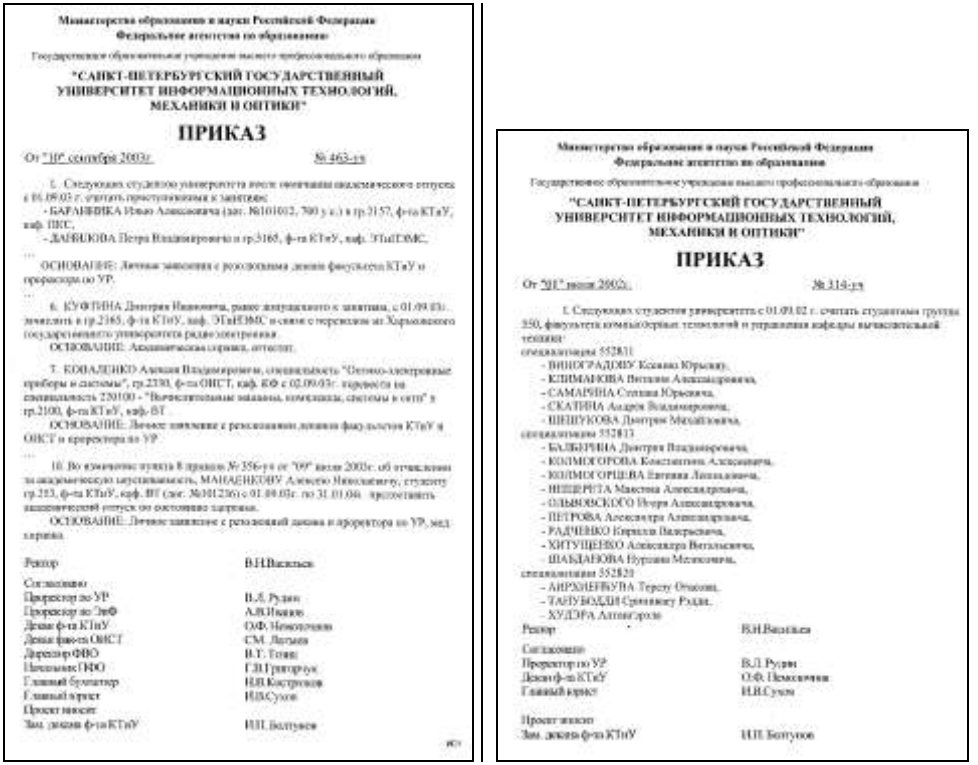
21.1. Описание предметной области "Итоговая успеваемость"

По рассмотренным ранее учебным планам производится обучение студентов. В *разд. 19.2.5* отмечалось, что каждая студенческая группа "привязывается" к рабочему учебному плану, в соответствии с которым должно проводиться обучение студентов, включенных в эту группу. Если, например, студенты какой-либо группы должны заниматься по разным планам (они выбрали различные магистерские программы или дисциплины по выбору), то такая группа должна быть привязана к нескольким рабочим учебным планам, а конкретные студенты группы к одному из этих планов.

Привязку студентов к соответствующим группам и рабочим учебным планам производят в приказах по "движению" студентов (переходы из группы в группу, уход в академический отпуск и возврат из него, отчисление и т. п.). На рис. 21.1 приведены примеры таких приказов.

В тех случаях, когда все студенты группы занимаются по одному плану, то его номер не указывается в приказе, а выбирается из базы данных по номеру группы (см. рис. 20.10). В противном случае в приказе указывается либо номер плана, либо специализация, по которой будет производиться обучение конкретного студента (см. приказ № 314 на рис. 21.1).

В пунктах приказов также указывается одна или две даты. Это может быть дата успешного завершения обучения или отчисления, зачисления (перевода) в группу (в этом случае подразумевается, что студент будет обучаться в этой группе до конца соответствующего учебного года или конца действия рабочего учебного плана), даты начала и окончания академического отпуска и т. п.



Министерство образования и науки Российской Федерации
Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

"САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ"

ПРИКАЗ

От "10" сентября 2003г. № 463-уч

1. Следующих студентов университета после окончания академического отпуска с 01.09.03 г. считать приступившими к занятиям:
- БАРАНОВКА Ивана Алексеевича (дог. №101012, 500 у.е.) в гр.3157, ф-та КТиУ, каф. ПКС;
- ДАНЬКОВА Петра Владимировича в гр.3165, ф-та КТиУ, каф. УТиПОМС;
...

ОСНОВАНИЕ: Личные заявления с резолюциями декана факультета КТиУ и проректора по УР.
...

6. КУФТИНА Дмитрия Ивановича, ранее допущенного к занятиям, с 01.09.03г. зачислить в гр.2165, ф-та КТиУ, каф. ЭТиПЭМС в связи с переводом из Харьковского государственного университета радиоэлектроники.

ОСНОВАНИЕ: Академическая справка, аттестат.

7. КОВАЛЕНКО Алексея Владимировича, специальность "Оптико-электронные приборы и системы", гр.2230, ф-та ОИСТ, каф. ЮФ с 02.09.03г. перевести на специальность 220100 - "Вычислительные машины, комплексы, системы и сети" в гр.2100, ф-та КТиУ, каф. ВТ.

ОСНОВАНИЕ: Личное заявление с резолюциями декана факультета КТиУ и ОИСТ и проректора по УР.
...

10. Во изменение пункта 8 приказа № 356-уч от "09" июля 2003г. об отчислении за академическую неуспеваемость, МАНДЕНКОВУ Александру Николаевичу, студенту гр.253, ф-та КТиУ, каф. ВТ (дог. №0012361) с 01.09.03г. по 31.01.04г. предоставить академический отпуск по состоянию здоровья.

ОСНОВАНИЕ: Личное заявление с резолюцией декана и проректора по УР, мед. справка.

Ректор В.Н.Васильев

Согласовано:
Проректор по УР В.Л.Рудин
Проректор по ЭиФ А.В.Иванов
Декан ф-та КТиУ О.Ф.Новиченков
Декан ф-та ОИСТ С.М. Латыев
Директор ФИО В.Т. Точин
Начальник ГЮО Г.В.Григорьев
Главный бухгалтер Н.В.Кострюков
Главный юрист Н.В.Сухов
Проект вносит:
Зам. декана ф-та КТиУ И.П. Болтунов

Министерство образования и науки Российской Федерации
Федеральное агентство по образованию

Государственное образовательное учреждение высшего профессионального образования

"САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ"

ПРИКАЗ

От "01" июня 2002г. № 314-уч

1. Следующих студентов университета с 01.09.02 г. считать студентами группы 350, факультета компьютерных технологий и управления кафедры вычислительной техники:
специализация 552831
- ВИНОГРАДОВУ Ксению Юрьевну,
- КЛИМАНОВА Виталия Александровича,
- САМАРИНА Степана Юрьевича,
- СКАТИНА Андрея Владимировича,
- ШЕШУКОВА Дмитрия Михайловича,
специализация 552813
- БАЛБЕРИНА Дмитрия Владимировича,
- КОЛМОГОРОВА Константина Александровича,
- КОЛМОГОРЦЕВА Евгения Анатольевича,
- НЕЩЕРЕТА Максима Александровича,
- ОЛЬХОВСКОГО Игоря Александровича,
- ПЕТРОВА Александра Александровича,
- РАДЧЕНКО Кирилла Валерьевича,
- ХИТУЩЕНКО Александра Витальевича,
- ШАБДАНОВА Нурлана Мелисовича,
специализация 552820
- АИРЭНЕРБУВА Терсту Отсоны,
- ТАНУБОДДИ Сринивасу Радж,
- ХУДЭРА Аттилу Золт

Ректор В.Н.Васильев

Согласовано:
Проректор по УР В.Л. Рудин
Декан ф-та КТиУ О.Ф. Новиченков
Главный юрист Н.В.Сухов

Проект вносит:
Зам. декана ф-та КТиУ И.П. Болтунов

Рис. 21.1. Примеры приказов по "движению" студентов

В предисловии отмечалось, что в данном курсе будет рассматриваться только часть интегрированной БД информационной системы СПбГУ ИТМО, касающаяся некоторых сторон организации и планирования учебного процесса. Поэтому затронем здесь лишь то, что непосредственно относится к итоговой успеваемости: краткие сведения о студентах (без адресов, личных документов и пр.) и некоторые аспекты чрезвычайно важной системы подготовки приказов.

При подготовке пункта студенческого приказа информационная система СПбГУ ИТМО автоматически создает по тексту пункта значение одного из признаков этого пункта: `академ` (в академическом отпуске), `обучен` (обучается), `отчисл` (отчислен), `диплом` (окончил с защитой диплома) и пр. Кроме того, система устанавливает для этого пункта состояние `проект`, которое позволяет модифицировать текст пункта до тех пор, пока он не попадет в текст приказа и не будет `утвержден` вместе с другими пунктами. Теперь все изменения пункта могут производиться только путем его отмены (перевод в состояние `отменен`) и размещения в новом приказе пункта с текстом "Во изменение …" со ссылкой на отмененный пункт.

Следует отметить, что студент может одновременно обучаться в нескольких группах по разным рабочим учебным планам (например, будучи студентом дневного отделения, получать второе образование на вечернем отделении нашего университета). В этом случае у такого студента должно быть несколько зачетных книжек и несколько учебных карточек для фиксации наборов оценок.

Напомним, что в итоговую успеваемость студента включаются оценки, которые должны быть им получены в соответствии с данными столбцов `Формы итогового контроля` базового учебного плана (см. рис. 19.3) или рабочих учебных планов, построенных по форме, показанной на рис. 19.6 (т. е. без форм текущего контроля).

Для информирования студентов и преподавателей, а также подготовки экзаменационных ведомостей (рис. 20.2), составляется расписание консультаций и экзаменов, проводимых в сессию.

Санкт-Петербургский Государственный Университет Информационных Технологий Механики и Оптики

Зачетно-экзаменационная ведомость N 3449

Семестр 5 | 2006/2007 учебный год | Дата

Факультет КТиУ | Кафедра ВТ | 3 курс | Группа 3100

Дисциплина Моделирование | Вид контроля КР

Фамилия И.О. преподавателя Алиев Т.И.

№	Номер зачетной книжки	Фамилия И.О.	Оценка / зачет прописью	Оценка / зачет буквой	Подпись преподавателя
1	041077	Ан Д.П.	удовл	E	
2	041078	Бойков Д.Д.	удовл	E	
3	051432	Дмитров А.Ю.	отл	BA	
4	041086	Дмитриев А.Г.	н. я.		
5	041089	Дорофеева Е.И.	хор	C	
6	041090	Дрожкин А.И.	удовл	D	
7	041101	Киселев Д.А.	удовл	D	
8	041102	Козлова О.А.	хор	C	
9	041106	Магденков Д.В.	хор	C	
10	041108	Мазуров М.Б.	хор	DC	
11	041109	Маркин Д.А.	хор	DC	
12	041111	Машков И.В.	хор	C	
13	051431	Михайлов В.В.	удовл	E	
14	031344	Павлов А.В.	н. я.		
15	041230	Попов Р.И.	отл	A	
16	041118	Пузиков П.А.	н. я.		
17	041121	Румянцев А.С.	отл	A	
18	041123	Савицкая И.А.	н. я.		
19	041129	Стадникова О.И.	хор	C	
20	041268	Тальвинский Г.В.	хор	C	

ИТОГО:

отлично –
хорошо –
удовлетворительно –
неудовлетворительно –

отлично (A) –
очень хорошо (B) –
хорошо (C) –
удовлетворительно (D) –
посредственно (E) –
неудовлетворительно (FX) –

ДЕКАН ФАКУЛЬТЕТА

Рис. 21.2. Экзаменационная ведомость

Если по тем или иным причинам студент не явился на плановый экзамен (зачет) или получил неудовлетворительную оценку, то для внеплановой сдачи он должен получить в деканате экзаменационный лист, позволяющий ему явиться в установленный срок к указанному преподавателю (рис. 21.3) для сдачи (пересдачи) экзамена (зачета).

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Экзаменационный лист № 3637 - (1)

Для сдачи экзамена или зачета ниже группа, указанная к зачетно-экзаменационной ведомости группа

зав.каф. Алиев Тауфик Измайлович

Должность, ФИО преподавателя

Факультет	За курс	За семестр	Группа	Дисциплина	Форма проверки знаний (зачет, экзамен)
КТиУ	3	5	3100	Моделирование	КР

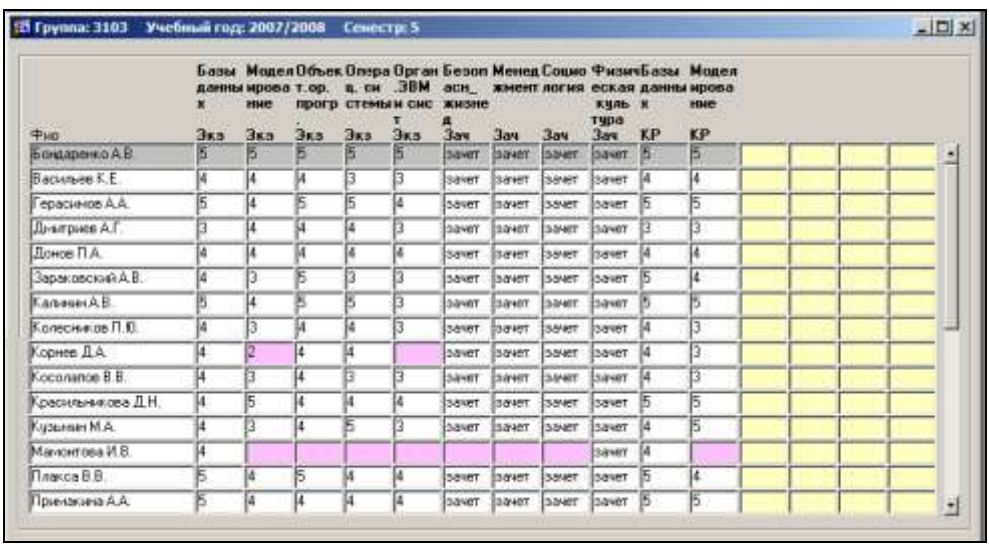
Фамилия и инициалы студента	Российская оценка (прописью)	Оценка ECTS (буквой)	Подпись экзаменатора, фамилия и инициалы	Дата	Подпись зав. кафедрой, фамилия и инициалы
Дмитриев А.Г.	удов.	E	Ал-Т.Алиев	30.1.07	Ал-Т.Алиев

Действителен до 02.02.2007 200 г.

Декан факультета " 29.01.2007 " 200 г.

Направление сдается кафедрой в деканат не позднее трех дней после приема экзамена или зачета

Рис. 21.3. Экзаменационный лист



Группа: 3103 Учебный год: 2007/2008 Семестр: 5

Фио	Базы данных Экз	Моделирование Экз	Объект.ор. прогр. Экз	Операц. системы Экз	Орган.ЭВМ и сист Экз	Безопасн_жизнед Зач	Менеджмент Зач	Социология Зач	Физическая культура Зач	Базы данных КР	Моделирование КР
Бондаренко А.В.	5	5	5	5	5	зачет	зачет	зачет	зачет	5	5
Васильев К.Е.	4	4	4	3	3	зачет	зачет	зачет	зачет	4	4
Герасимов А.А.	5	4	5	5	4	зачет	зачет	зачет	зачет	5	5
Дмитриев А.Г.	3	4	4	4	3	зачет	зачет	зачет	зачет	3	3
Донов П.А.	4	4	4	4	4	зачет	зачет	зачет	зачет	4	4
Зараковский А.В.	4	3	5	3	3	зачет	зачет	зачет	зачет	5	4
Кальянов А.В.	5	4	5	5	3	зачет	зачет	зачет	зачет	5	5
Колесников П.Ю.	4	3	4	4	3	зачет	зачет	зачет	зачет	4	3
Корнев Д.А.	4	2	4	4		зачет	зачет	зачет	зачет	4	3
Косолапов В.В.	4	3	4	3	3	зачет	зачет	зачет	зачет	4	3
Красильникова Д.Н.	4	5	4	4	4	зачет	зачет	зачет	зачет	5	5
Кузьмин М.А.	4	3	4	5	3	зачет	зачет	зачет	зачет	4	5
Мамонтова И.В.	4								зачет	4	
Плакса В.В.	5	4	5	4	4	зачет	зачет	зачет	зачет	5	4
Приказина А.А.	5	4	4	4	4	зачет	зачет	зачет	зачет	5	5

Рис. 21.4. Журнал успеваемости группы

Оценки из экзаменационных (зачетно-экзаменационных или зачетных) ведомостей и листов переносятся сотрудниками деканатов в групповые журналы успеваемости (рис. 21.4) и учебные карточки студентов, где кроме общих сведений о студенте помещаются все данные об его оценках по всем изученным дисциплинам.

Анализ этих оценок позволяет выявить неуспевающих студентов и те дисциплины, по которым ими не получены положительные оценки.

При завершении обучения по этим оценками составляется приложение к диплому или справка о прослушанных курсах, если студент отчисляется из университета.

21.2. Инфологическая модель "Итоговая успеваемость"

Здесь, в отличие от *разд. 20.3*, сначала представим ER-диаграмму "Итоговая успеваемость" (рис. 21.5), а затем приведем ее описание.

Так как оценивается успешность выполнения студентом учебных планов, то в ER-диаграмму включено восемь уже рассмотренных ранее сущностей, которые, так или иначе, связаны с вновь созданными.

Начнем описание с сущности `Люди`, в которую включены атрибуты с основными данными не только о студентах, но и о любом человеке, связанном с университетом. Дело в том, что студент может быть одновременно и сотрудником (например, студент оформлен на должность лаборанта кафедры, сотрудник обучается на вечернем отделении университета и пр.), выпускник может остаться в университете в качестве аспиранта или сотрудника и т. п. Кроме того, ведомости и экзаменационные листы выписываются на преподавателей, сведения о которых также должны существовать в указанной сущности. Идентификатор этой сущности (`ИД`) — шестизначное целое число, которое используется в качестве табельного номера сотрудника и (или) номера (учетного имени) студента.

Так как некоторые люди меняют фамилии (а иногда имена, отчества и даже пол), то в модель включена сущность `Изм_люди`. Когда производится какое-либо изменение записи сущности `Люди`, то ее данные, существовавшие до внесения изменений, переносятся в `Изм_люди` с указанием даты, до которой они были актуальными. Например, студентка Слепнева (номер 119743) до 11.06.2004 имела в информационной системе фамилию Притчина.

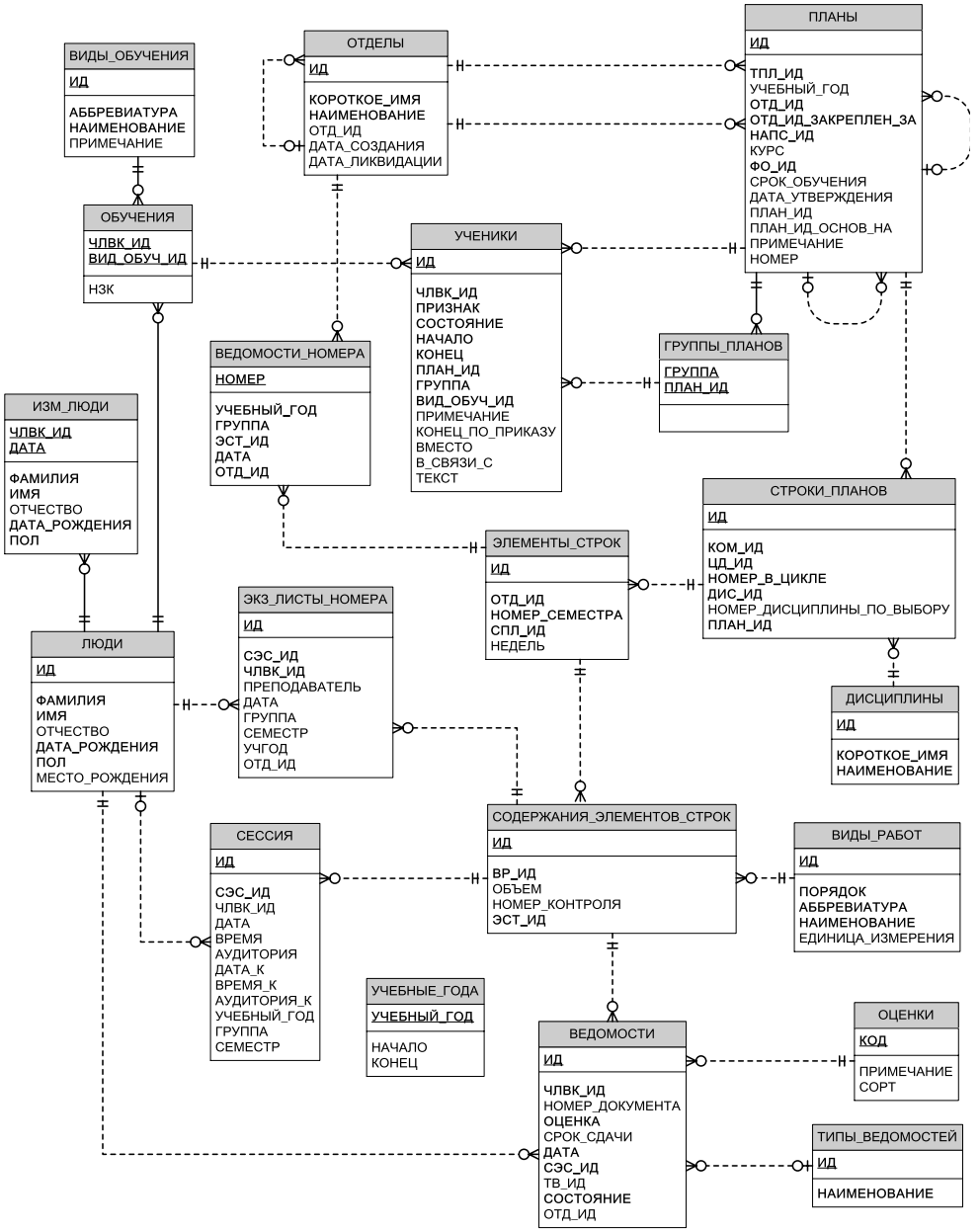


Рис. 21.5. ER-диаграмма "Итоговая успеваемость"

В *разд. 21.1* рассматривались приказы по "движению" контингента студентов. В момент создания пунктов таких приказов производится заполнение

одной или нескольких записей сущности Ученики. Мы ограничимся упоминанием информации о пунктах приказов, данной в описании сущности Ученики (рис. 21.6), не рассматривая здесь достаточно сложной системы подготовки приказов.

УЧЕНИКИ

Используется для хранения данных о "движении" контингента студентов

Ключи	Атрибуты	Типы данных	Обязательность	Описания атрибутов
PK	ИД	NUMBER(9)	not null	Искусственный первичный уникальный идентификатор (суррогатный ключ)
FK	ЧЛВК_ИД	NUMBER(9)	not null	Указатель на составной уникальный идентификатор ЧЛВК_ИД, ВИД_ОБУЧ_ИД сущности ОБУЧЕНИЯ
	ПРИЗНАК	VARCHAR2(10)	not null	Состояние студента (обучен,отчисл, академ,диплом…)
	СОСТОЯНИЕ	VARCHAR2(9)	not null	Состояние пункта приказа (проект, утвержден, отменен)
	НАЧАЛО	DATE	not null	Начало действия записи пункта приказа
	КОНЕЦ	DATE	not null	Окончание действия пункта приказа
FK	ПЛАН_ИД	NUMBER(9)	not null	Указатель на идентификатор ПЛАНЫ.ИД справочника учебных планов
FK	ГРУППА	VARCHAR2(4)	not null	Номер студенческой группы и указатель на составной уникальный идентификатор ПЛАН_ИД, ГРУППА сущности ГРУППЫ_ПЛАНОВ
FK	П_ПРКОК_ИД	NUMBER(9)	not null	Указатель на идентификатор ПУНКТЫ_ПРИКАЗОВ_ОК.ИД
FK	ВИД_ОБУЧ_ИД	NUMBER(9)	not null	Указатель на составной уникальный идентификатор ЧЛВК_ИД, ВИД_ОБУЧ_ИД сущности ОБУЧЕНИЯ
	ПРИМЕЧАНИЕ	VARCHAR2(200)	null	Примечание
	ВМЕСТО	NUMBER(9)	null	Ссылка на запись, вместо которой введена данная
	КОНЕЦ_ПО_ПРИКАЗУ	DATE	null	Конец периода действия записи по приказу
	В_СВЯЗИ_С	NUMBER(9)	null	Указатель ИД записи, вызвавшей изменение текущего атрибута конец
	ТЕКСТ	VARCHAR2(200)	null	Текст, используемый в пункте приказа

Рис. 21.6. Описание сущности Ученики

Так как студент может обучаться более чем по одному учебному плану (см. *разд. 21.1*), то Ученики связаны с сущностью Люди не прямо, а через сущность Обучения. В этой сущности кроме первичного составного ключа (ЧЛВК_ИД и ВИД_ОБУЧ_ИД) хранится и номер зачетной книжки (НЗК), соответствующий тому или иному виду обучения, данные о которых размещены в справочнике Виды_обучения (1 — Основное образование, 2 — Второе образование и т. д.).

Итоговые оценки, получаемые студентами по изучаемым ими дисциплинам, вводятся в сущность Ведомости, снабженную двумя справочниками

`Типы_ведомостей` (`1` — Ведомость, `2` — Экзаменационный лист, `3` — Перезачет) и `Оценки` (`'5'` — отлично, и т. д., `'зачет'` — зачет, `'незач'` — незачет и пр.), а также в сущность `Экз_листы_номера`.

Атрибутами сущности `Ведомость` являются:

- `'члвк_ид'` — идентификатор студента;
- `'тв_ид'` — указатель на `ТИПЫ_ВЕДОМОСТЕЙ.ИД` сущности `Типы_ведомостей`;
- `'номер_документа'` — номер ведомости или экзаменационного листа;
- `'срок_сдачи'` — дата, до которой должен быть сдан экзамен или зачет;
- `'оценка'` — оценка, полученная при сдаче;
- `'дата'` — дата сдачи;
- `'состояние'` — признак, позволяющий включать (`актуальна`) или не включать (`неактуальна`, например уже пересдана) оценку в процедуры анализа успеваемости;
- `'сэс_ид'` — указатель на `СОДЕРЖАНИЯ_ЭЛЕМЕНТОВ_СТРОК.ИД` сущности `Содержания элементов строк`, с помощью которого (через сущности `Элементы строк`, `Строки планов` и `Дисциплины`) определяется название дисциплины, по которой получена данная оценка;
- `'отд_ид'` — указатель на идентификатор `ОТДЕЛЫ.ИД` сущности `Отделы`, с помощью которого определяется деканат, который сформировал ведомость;
- `'ид'` — искусственный первичный уникальный идентификатор сущности.

Атрибутами сущности `Экз_листы_номера` являются:

- `'члвк_ид'` и `'группа'` — идентификатор студента и номер его группы;
- `'преподаватель'` — идентификатор преподавателя, которому поручается принять экзамен (зачет);
- `'дата'` — дата окончания срока действия экзаменационного листа;
- `'учгод'` и `'семестр'` — учебный год и семестр (весенний или осенний), в котором проводился плановый экзамен (зачет) по указанной в экзаменационном листе дисциплине;
- `'сэс_ид'` — указатель на `СОДЕРЖАНИЯ_ЭЛЕМЕНТОВ_СТРОК.ИД` сущности `Содержания элементов строк`, с помощью которого (через сущности `Элементы строк`, `Строки планов` и `Дисциплины`) определяется название дисциплины, по которой должен быть принят экзамен (зачет);
- `'отд_ид'` — указатель на идентификатор `ОТДЕЛЫ.ИД` сущности `Отделы`, с помощью которого определяется деканат, который выдал экзаменационный лист;

- 'ид' — искусственный первичный уникальный идентификатор сущности, одновременно являющийся номером экзаменационного листа.

Кроме рассмотренных ранее основных сущностей есть еще три вспомогательных: Ведомости_номера, Сессия и Учебные_года.

Первая из них содержит сведения, позволяющие создать "шапку" ведомости (рис. 21.2): номер ведомости ('номер'), учебный год ('учгод'), группу ('группа'), дату ('дата') и факультет ('отд_ид'), а также атрибут 'эст_ид' — указатель на идентификатор ЭЛЕМЕНТЫ_СТРОК.ИД, сущности Элементы строк, с помощью которого (через сущности Строки планов и Дисциплины) определяется название дисциплины.

Вторая содержит сведения для создания расписаний экзаменов и консультаций.

Третья содержит даты начала и конца учебных годов, начиная с 1996/1997.

21.3. Объединенная инфологическая модель "UCHEB"

Напомним, что практическое изучение данной дисциплины будет проводиться с использованием тех таблиц базы данных информационной системы СПбГУ ИТМО, аналоги которых были рассмотрены в *разд. 20.3* и *21.2*.

Необходимость в описании объединенной модели (рис. 21.7) возникла в связи с тем, что при описании моделей "Учебные планы" и "Итоговая успеваемость" для упрощения были опущены некоторые детали, чрезвычайно важные для эксплуатации системы, но почти не влияющие на понимание особенностей ее функционирования.

Все сущности этой модели снабжены префиксом "Н_", который позволяет отличать их от сущностей, не входящих в состав основных сущностей информационной системы СПбГУ ИТМО. Кроме того, все слова в их именах и именах их атрибутов соединены символом "_" (подчеркивание), так как по правилам Oracle имена должны состоять из символьной строки длиной не более 30 символов или из нескольких слов, заключенных в двойные кавычки: "…".

Практически во все сущности добавлены атрибуты кто_создал, когда_создал, кто_изменил и когда_изменил. Когда в сущность добавляется новая запись, то в атрибуты кто_создал и кто_изменил вводится имя пользователя, создавшего эту запись (например, КТУ — имя деканата факультета КТиУ), а в атрибуты когда_создал и когда_изменил — дата и время создания записи (например, 28.06.2004 16:27:54).

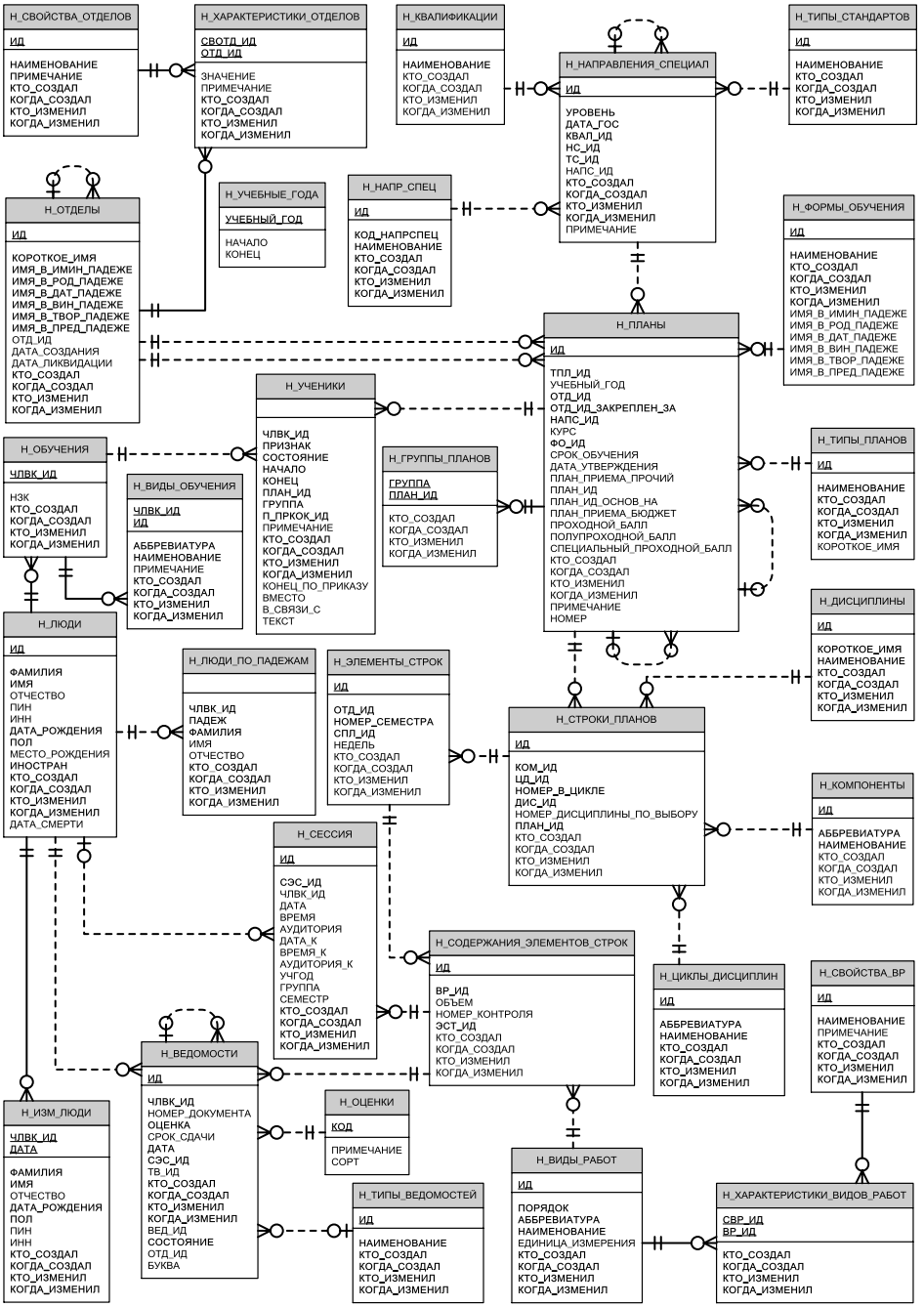


Рис. 21.7. ER-диаграмма "Учебный процесс"

В сущностях Н_ОТДЕЛЫ и Н_ФОРМЫ_ОБУЧЕНИЯ вместо атрибутов НАИМЕНОВАНИЕ включены атрибуты: ИМЯ_В_ИМИН_ПАДЕЖЕ, ИМЯ_В_РОД_ПАДЕЖЕ, ИМЯ_В_ДАТ_ПАДЕЖЕ, ИМЯ_В_ВИН_ПАДЕЖЕ, ИМЯ_В_ТВОР_ПАДЕЖЕ и ИМЯ_В_ПРЕД_ПАДЕЖЕ. Это сделано для того, чтобы при создании текстов приказов (см. рис. 21.1) можно было бы ввести в него наименование отдела и (или) формы обучения в нужном падеже. (Отметим, что фамилии, имена и отчества вводятся в приказ в нужном падеже с помощью специальной функции.)

В сущность Н_ПЛАНЫ добавлены атрибуты с планами приема и проходными баллами, которые используются при планировании приема (планы для абитуриентов).

В сущности Н_ЛЮДИ и Н_ИЗМ_ЛЮДИ добавлены атрибуты:

- 'ПИН' — номер страхового свидетельства Государственного пенсионного страхования;
- 'ИНН' — идентификационный номер налогоплательщика;
- 'ИНОСТРАН' — признак гражданства (0 — гражданин России, 1 — не гражданин России);
- 'ДАТА_СМЕРТИ' — дата из свидетельства о смерти или запредельная дата 09.09.9999.

Глава 22

Работаем с SQL

Здесь приведен перечень запросов, которые должны быть "переведены" на язык SQL и реализованы во время лабораторных занятий по дисциплине "Базы данных".

22.1. Запросы

Инфологическая модель базы данных "Учебный процесс" показана на рис. 21.7.

Простая выборка

1. Выдать содержимое всех столбцов таблицы `Н_ЦИКЛЫ_ДИСЦИПЛИН`.
2. Выдать содержимое столбцов `Аббревиатура` и `Наименование` той же таблицы.
3. Получить перечень квалификаций, присваиваемых выпускникам нашего университета.

Исключение дубликатов

4. Выдать неповторяющиеся имена людей из таблицы `Н_ЛЮДИ`.
5. Какие состояния студентов (признаки) используются в таблице `Н_УЧЕНИКИ`? Выборка вычисляемых значений.
6. Из таблицы `Н_УЧЕБНЫЕ_ГОДА` получить названия и продолжительность каждого учебного года (напомним, что единица измерения дат — одни сутки).
7. Используя сведения из таблицы `Н_ИЗМ_ЛЮДИ`, определить:
   - целое число дней, прошедших с момента изменения фамилии до текущей даты (`SYSDATE`);

- целое число месяцев, прошедших с момента изменения фамилии до текущей даты.

Для округления используется функция `ROUND(expr [,m])`, возвращающая `expr`, округленное до `m`-го десятичного знака; если `m` опущено, то оно принимается равным `0`, а если `m<0`, то округляются цифры левее десятичной точки.

Для получения количества месяцев между датами `d1` и `d2` используется функция `MONTHS_BETWEEN (d1,d2)`. Если `d1>d2`, то результат положителен, иначе отрицателен.

Выборка с использованием фразы *WHERE* и упорядочением

8. Выдать фамилию и инициалы людей (запрос оформить в виде выражения, чтобы получить в результате один столбец, например, "Сидоров С.С.", а не три: "Сидоров","С." и "С.").

Для выделения из строки `str len` символов, начиная с `pos`, используется функция `SUBSTR(str,pos[,len])`. Если `len` отсутствует, то выдаются символы от `pos` до конца `str`. При `pos<0` первый выделяемый символ определяется не от начала, а от конца `str`.

Для соединения текстовых значений используется оператор `||`, а значения текстовых констант должны быть заключены в апострофы (например, точка и пробел: `'.'` и `' '`).

Для ограничения числа строк используйте псевдостолбец `ROWNUM` (`WHERE ROWNUM < …` или `WHERE <= …`) и ограничьте результат 50-ю строками.

9. Выдать номер, фамилию и инициалы людей (запрос оформить в виде выражения, чтобы получить, в результате один столбец, например, "104567 Сидоров С.С.").

Для преобразования числового значения или даты (`expr`) в текстовое значение по формату, заданному в `fmt` (с необязательным указанием национального языка `nlsparam`), используется функция `TO_CHAR (expr [, fmt [, 'nlsparam']])`. Если `fmt` опущено, то `expr` преобразуется в строку такой длины, которая вмещает только значащие цифры. С форматами можно познакомиться в *разд. 4.6*. Для ограничения числа строк используйте псевдостолбец `ROWNUM` (`WHERE ROWNUM < …` или `WHERE <= …`) и ограничьте результат 50-ю строками.

10. Выдать номер (`ИД`), `Короткое_имя` и название (`Имя_в_имин_падеже`) всех отделов факультета Компьютерных технологий и управления.

11. Выдать фамилию, имя, отчество всех людей, фамилии которых начинаются на "Яков".
12. Выдать фамилию, имя, отчество всех людей из запроса номер 11, отсортировав строки по: 1) имени; 2) имени и отчеству; 3) фамилии, имени и отчеству.
13. Повторить предыдущие запросы, задавая сортировку позициями, а не именами столбцов.
14. Выдать фамилию, имя, отчество всех людей с фамилиями, начинающимися на "Э", "Ю" и "Я", с упорядочением по фамилии (составить не менее двух вариантов запроса).
15. Выдать номера, короткие имена и названия отделов с номерами 102, 111 и 212.
16. Получить из таблицы `Н_НАПР_СПЕЦ` код и наименование специализаций специальности 230101 (напомним, что номера специализаций отличаются от номера специальности двумя последними цифрами).
17. По таблице `Н_УЧЕНИКИ` познакомиться с особенностями процесса обучения студентов с номерами 118843,119299,119457,119490,120224.

    Для получения ФИО этих студентов можно воспользоваться функцией `Человек (члвк_ид, padej , priz IN NUMBER DEFAULT 0, nach DATE DEFAULT SYSDATE)`, выводящей фамилию, имя и отчество (`priz=0`) или фамилию и инициалы (`priz=1`) человека с номером `члвк_ид` в заданном падеже (`'И','Р','Д','В','Т','П'`) и на заданную дату (по умолчанию устанавливается `priz=0` и текущая (системная) дата — `SYSDATE`. Для соединения текстовых значений используется оператор `||`, а значения текстовых констант должны быть заключены в апострофы (например, точка и пробел: `'.'` и `' '`).

18. Вывести из таблицы `Н_ВЕДОМОСТЬ` ваши оценки. Список должен содержать оценку, ее преобразованное значение (`5` → `отлично`, `4` → `хорошо`, `3` → `удовлетворительно`, `2` → `неудовлетворительно`, `зачет` → `зачет`, `незач` → `незачет`, `осв` → `освобождение`, `неявка` → `неявка`, `99` → `диплом с отличием`, `.` → `выдача экзаменационного листа`, `-` → `отсутствие данных`), которому необходимо дать псевдоним — `Эквивалент`, дату получения оценки и ссылку (`сэс_ид`) на идентификатор таблицы `Н_СОДЕРЖИМОЕ_ЭЛЕМЕНТОВ_СТРОК`. Зная `сэс_ид`, можно получить имя дисциплины, по которой получена оценка. Список упорядочить по `сэс_ид` и дате.

    Преобразование оценок можно осуществить с помощью функции `DECODE`: `DECODE (expr, search1, result1 [, search2, result2] ... [default])`, где значение выражения `expr` сравнивается с каждым из зна-

чений search. Если expr совпадает с каким-либо search, возвращается соответствующее значение result. Если ни одного совпадения не найдено, возвращается значение default (или NULL, если значение default опущено). Expr может иметь любой тип данных, но значения search должны иметь тот же тип, как у expr. Возвращаемое значение принудительно приводится к тому типу данных, как у result.

19. Вывести список тех ваших оценок, которые заданы цифрами 5, 4, 3, 2. Список должен содержать оценку, ее удвоенный цифровой эквивалент (псевдоним — Оценка*2), дату получения оценки и ссылку (сэс_ид) на идентификатор таблицы Н_СОДЕРЖИМОЕ_ЭЛЕМЕНТОВ_СТРОК. Зная сэс_ид, можно получить имя дисциплины, по которой получена оценка. Список упорядочить по сэс_ид и дате.

    Следует иметь в виду, что любое имя Oracle может в чистом виде содержать лишь буквы, цифры и три символа: $, #, _. Имена, содержащие другие символы (включая пробелы), необходимо заключать в кавычки, например, "Оценка*2".

    Для преобразования текстового значения (char) в число по формату в fmt (с необязательным указанием национального языка nlsparam), используется функция TO_NUMBER (char [, fmt [, 'nlsparam']]). С форматами можно познакомиться в *разд. 4.6*.

20. Выполнить запрос 4, удалив из результата пробелы и точки (отсутствие имени у некоторых студентов), а также однобуквенные имена, получившиеся при вводе инициалов: ' ', '.', 'А', 'А.', 'Б', 'Б.', 'В', 'В.', 'Г', 'Г.', 'Д', 'Д.', 'Е', 'Е.', 'Ж', 'Ж.', 'З', 'З.', 'И', 'И.', 'К', 'К.', 'Л', 'Л.', 'М', 'М.', 'Н.', 'Н.', 'О', 'О.', 'П', 'П.', 'Р', 'Р.', 'С', 'С.', 'Т', 'Т.', 'У', 'У.', 'Ф', 'Ф.', 'Х', 'Х.', 'Ц', 'Ц.', 'Ш', 'Ш.', 'Э', 'Э.', 'Ю', 'Ю.'.

Агрегирование данных

21. Сколько Алексеев в таблице Н_ЛЮДИ? Результат выдать в виде одной текстовой строки вида: "Алексей - 123".

22. Определить количество различных имен людей в таблице Н_ЛЮДИ, удалив из результата пробелы, точки и однобуквенные имена (см. запрос 20).

23. Определите вашу среднюю оценку (естественно, что в расчет должны входить лишь те оценки, которые имеют цифровой эквивалент). Создайте два запроса, в которых средняя оценки определяется с помощью функции среднее значение (функция AVG) и путем деления суммы (функция SUM)

оценок на их количество (функция COUNT). В результате необходимо оставить два десятичных знака после запятой.

Для округления используется функция ROUND(expr [,m]), возвращающая expr, округленное до m-го десятичного знака; если m опущено, то оно принимается равным 0, а если m<0, то округляются цифры левее десятичной точки.

24. Сколько всего людей с фамилией Иванов и сколько у них различных имен и различных отчеств. Для получения результата использовать один запрос со следующими псевдонимами столбцов: Всего, Разных_имен, Разных_отчеств. В результат не должны включаться имена (отчества) в виде пробелов, точек и однобуквенных имен (см. запрос 20).

Проверку можно осуществить путем вывода списков Ивановых с неповторяющимися именами и с неповторяющимися отчествами.

Агрегирование данных с использованием фразы *HAVING*

25. Выдать различные:
    - фамилии людей и число людей с каждой из этих фамилий, ограничив список фамилиями, встречающимися не менее 50 раз;
    - имена людей и число людей с каждым из этих имен, ограничив список именами, встречающимися не менее 300 раз;
    - отчества людей и число людей с каждым из этих отчеств, ограничив список отчествами, встречающимися не менее 300 раз.

    Списки упорядочить по уменьшению количества фамилий (имен или отчеств).
26. Найти группы, в которых 4.7.2004 было менее десяти обучающихся студентов.
27. В таблице Н_ГРУППЫ_ПЛАНОВ найти номера планов, по которым обучается (обучалось) более 4 групп.

Естественное соединение таблиц

28. Используя опыт, полученный при реализации запроса 23, выведите таблицу со средними оценками студентов вашей группы (Номер, ФИО, Ср_оценка).

При составлении запроса следует учитывать, что группы с таким же номером существовали и в прошлые годы, а присутствие студента в вашей группе должно быть засвидетельствовано утвержденным приказом.

29. Выполнить предыдущий запрос, используя для получения номера, фамилии, имени и отчества студента не функцию `Человек`, а текстовую строку, полученную из таблицы `н_люди` примерно так, как формировался запрос 8.
30. Составить запрос, позволяющий получить таблицу направлений (специальностей и специализаций), имеющую вид, показанный в табл. 22.1.

Таблица 22.1. *Направления, специальности и специализации*

ИД	Уровень	Дата_ГОС	Направление/ специальность		Тип стандарта
68	3	01.03.1995	552800	Информатика и вычислительная техника	Направление подготовки магистров
143	3	01.03.1995	552811	Базы данных	Магистерская программа
144	3	01.03.1995	552813	Сети ЭВМ и телекоммуникации	Магистерская программа
145	3	01.03.1995	552820	Системы реального времени	Магистерская программа
700	3	01.03.2000	220100	Вычислительные машины, комплексы, системы и сети	Специальность
1251	3	01.03.2000	220104	Системы телекоммуникаций и компьютерной безопасности	Специализация
792	3	01.03.2000	220109	Технология разработки программных систем	Специализация
1250	3	01.03.2000	220111	Открытые информационно-вычислительные системы	Специализация
1249	3	01.03.2000	220112	Информационно-управляющие системы	Специализация

31. Преобразовать предыдущий запрос так, чтобы в получаемой таблице перед столбцом `Тип стандарта` появился столбец `Квалификация`.
32. Получить список студентов, зачисленных первого сентября позапрошлого учебного года на первый курс очной формы обучения специальности 230101. В результат включить:
    - номер группы;
    - номер, фамилию, имя и отчество студента;
    - номер и состояние пункта приказа;
    - признак, характеризующий состояние студента;
    - дату конца действия этого пункта.

    Результат упорядочить по номеру группы и фамилии.
33. Получить список студентов, по условиям п. 32, сохранив в нем только те строки, в которых состояние пунктами приказа равно `'утвержден'`.
34. Получить список студентов по условиям п. 33, но только на конец учебного года (31 августа).

Соединение таблицы со своей копией

35. Выявить людей с одинаковыми фамилиями, именами и отчествами, но разными номерами. Список должен содержать упорядоченные строки с уникальными сочетаниями номера, фамилии, имени и отчества.

Вложенные подзапросы

36. Преобразовать запрос п. 32 так, чтобы во фразе `FROM` осталось соединение только тех таблиц, столбцы которых входят в списки фраз `SELECT` и `ORDER BY`. Остальные таблицы, данные из которых нужны для отбора требуемых строк результата, следует разместить во фразе `WHERE`, не используя их соединений. Для организации подзапросов использовать предикат `IN` (проверка на принадлежность).
37. Преобразовать предыдущий запрос, используя для организации подзапросов предикат `EXISTS` (проверка на существование).
38. Преобразовать запрос п. 29 так, чтобы во фразе `FROM` осталось соединение только тех таблиц, столбцы которых входят в списки фраз `SELECT` и `GROUP BY`. Остальные таблицы, данные из которых нужны для отбора нужных строк результата, необходимо разместить во фразе `WHERE`.
39. Вывести упорядоченный по ФИО список людей (с фамилиями, начинающимися на "До"), не являющихся или не являвшихся студентами

СПбГУ ИТМО (т. е. данные о которых отсутствуют в таблице Н_УЧЕНИКИ). Составить не менее двух вариантов запросов с условиями IN и EXISTS.

40. Выполнить запрос п. 35, используя для его реализации вложенный подзапрос, а не соединение таблицы со своей копией (в запросе нельзя использовать DISTINCT).

Объединение запросов

41. Сформировать запрос (единственный) для получения табл. 22.2.

Таблица 22.2. *Итоговая успеваемость*

	Кол-во
1. Круглых отличников	362
2. Учеников без троек	1692
3. Круглых троечников	281

22.2. Ответы к некоторым запросам

8.

```
SELECT фамилия||' '|| SUBSTR (имя,1,1)||'.'|| SUBSTR (отчество,1,1)||'.'
  FROM н_люди WHERE ROWNUM <= 50;
```

17.

```
SELECT члвк_ид,человек(члвк_ид,'И'),признак,состояние,начало,конец,группа
  FROM н_ученики
 WHERE члвк_ид IN (118843,119299,119457,119490,120224)
ORDER BY члвк_ид,начало;
26
SELECT группа,COUNT(*)
  FROM н_ученики
 WHERE '4.7.2007' BETWEEN начало AND конец
   AND признак = 'обучен'
   AND состояние = 'утвержден'
GROUP BY группа HAVING COUNT(*) < 10;
```

28.

```
SELECT v.члвк_ид номер, человек(v.члвк_ид,'И') ФИО,
       ROUND(AVG(TO_NUMBER(v.оценка)),2) Ср_оценка
  FROM н_ведомости v, н_ученики u
 WHERE v.члвк_ид = u.члвк_ид
   AND SYSDATE BETWEEN u.начало AND u.конец
   AND u.состояние = 'утвержден'
   AND v.оценка IN('5','4','3','2')
   AND v.состояние = 'актуальна'
   AND u.группа = '5101'
GROUP BY v.члвк_ид;
```

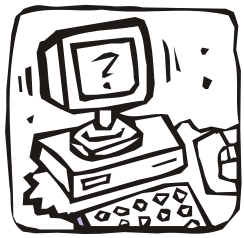
32.

```
SELECT
u.группа,u.члвк_ид,l.фамилия,l.имя,l.отчество,u.п_пркок_ид,u.признак,u.ко
нец
  FROM н_ученики u,н_люди l,н_планы p,н_напр_спец n,н_направления_специал
ns,
       н_формы_обучения f
 WHERE u.план_ид = p.ид
   AND p.напс_ид = ns.ид
   AND ns.нс_ид = n.ид
   AND p.фо_ид = f.ид
   AND u.члвк_ид = l.ид
   AND n.код_напрспец = '230101'
   AND p.курс = 1 AND f.наименование = 'Очная'
   AND p.учебный_год = '2005/2006'
   AND u.начало = '1.9.2005'
   AND u.состояние = 'утвержден'
ORDER BY u.группа,l.фамилия;
```

36.

```
SELECT u.группа,u.члвк_ид,l.фамилия,l.имя,l.отчество,u.п_пркок_ид,
       u.признак,u.конец
  FROM н_ученики u,н_люди l
 WHERE u.члвк_ид = l.ид
   AND u.начало = '1.9.2005'
   AND u.состояние='утвержден'
   AND u.план_ид IN
```

```
       (SELECT  p.ид
        FROM н_планы p
        WHERE p.ид = u.план_ид
        AND p.курс = 1
        AND p.учебный_год = '2005/2006'
        AND p.фо_ид IN
            (SELECT f.ид
             FROM н_формы_обучения f
             WHERE f.наименование = 'Очная')
             AND p.напс_ид IN
                 (SELECT ns.ид
                  FROM н_направления_специал ns
                  WHERE ns.ид = p.напс_ид
                  AND ns.нс_ид IN
                      (SELECT n.ид
                       FROM н_напр_спец n
                       WHERE n.код_напрспец = '230101')))
ORDER BY u.группа,l.фамилия;
```



Глава 23

Некоторые приложения базы данных "UCHEB"

Электронные версии текстов, приведенных далее функций и пакетов можно получить непосредственно из базы данных "UCHEB", выгрузка которой расположена на компакт-диске. В *приложении А* приведены процедуры загрузки этой базы данных и других приложений на персональный компьютер.

23.1. Функции *Человек* и *Decline*

Функция `Человек` предназначена для вывода фамилии, имени и отчества человека в задаваемом падеже (`padej`).

```
CREATE OR REPLACE FUNCTION Человек(chel NUMBER, padej VARCHAR2,
  priz IN NUMBER DEFAULT 0,nach DATE DEFAULT SYSDATE) RETURN VARCHAR2 IS
  -- вывод фамилии, имени, отчества или фамилии и инициалов
  -- человека на указанную дату в заданном падеже (padej)
    fio VARCHAR2(50) := ' ';
    t_KNR VARCHAR2(60);
    test NUMBER(9);
BEGIN
  -- Проверяем, можно ли склонять его ФИО
  SELECT COUNT(*) INTO t_KNR FROM н_люди l WHERE ид = chel
  AND (UPPER(l.место_рождения) LIKE 'КНР%' OR UPPER(l.место_рождения)
                               LIKE 'ВЬЕТНАМ%');
  -- Если именительный падеж или склонять нельзя
  IF padej = 'И' OR t_KNR > 0 THEN
    IF priz = 0 THEN
      SELECT UPPER(RTRIM(фамилия))||' '||DECODE(RTRIM(имя),'.',NULL,
         ' ',NULL,RTRIM(имя))||' '||DECODE(RTRIM(отчество),'.',NULL,
         ' ',NULL,RTRIM(отчество))
```

```
        INTO fio FROM н_люди WHERE ид = chel;
  -- Для формы с подписями
  ELSIF priz = 9 THEN
    SELECT DECODE(SUBSTR(имя,1,1),'.',NULL,' ',NULL, NULL, NULL,
          SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.',
          NULL,' ',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')||Фамилия
        INTO fio FROM н_люди WHERE ид = chel;
  ELSIF priz = 1 THEN
    SELECT Фамилия||' '||DECODE(SUBSTR(имя,1,1),'.',NULL,' ',NULL,NULL,
           NULL, SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.'
            ,NULL,' ', NULL, NULL, NULL, SUBSTR(отчество,1,1)||'.')
        INTO fio FROM н_люди WHERE ид = chel;
  ELSIF priz = 2 THEN
    SELECT RTRIM(фамилия)||' '||DECODE(RTRIM(имя),'.',NULL,' ',NULL,
           RTRIM(имя))||' '||DECODE(RTRIM(отчество),'.',NULL,' ',NULL,
           RTRIM(отчество))
        INTO fio FROM н_люди WHERE ид = chel;
  ELSE
    SELECT UPPER(RTRIM(фамилия))||' '|| DECODE(SUBSTR(имя,1,1),'.' ,
           NULL, ' ',NULL, NULL, NULL, SUBSTR(имя,1,1)||'.')||
           DECODE(SUBSTR(отчество,1,1),'.',NULL,' ', NULL, NULL, NULL,
           SUBSTR(отчество,1,1)||'.')
        INTO fio FROM н_люди WHERE ид = chel;
  END IF;
ELSE
  -- Проверяем, есть ли человек в таблице Н_ЛЮДИ_ПО_ПАДЕЖАМ, если есть,
  -- то все берем из нее, если нет, то используем функцию DECLINE
  SELECT COUNT(*) INTO test FROM н_люди_по_падежам lp
  WHERE lp.члвк_ид = chel AND lp.падеж = padej;
  IF test > 0 THEN
    IF priz = 0 THEN
      SELECT UPPER(RTRIM(фамилия))||' '||DECODE(RTRIM(имя),'.',
          NULL,' ', NULL, RTRIM(имя))||' '||DECODE(RTRIM(отчество),'.',
          NULL,' ',NULL,RTRIM(отчество))
          INTO fio FROM н_люди_по_падежам lp
      WHERE lp.члвк_ид = chel AND lp.падеж = padej;
    -- Для формы с подписями
    ELSIF priz = 9 THEN
      SELECT DECODE(SUBSTR(имя,1,1),'.',NULL,' ', NULL, NULL, NULL,
```

```
            SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.',
             NULL,' ',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')||Фамилия
          INTO fio FROM н_люди_по_падежам lp
       WHERE lp.члвк_ид = chel AND lp.падеж = padej;
     ELSIF priz = 1 THEN
       SELECT Фамилия||' '||DECODE(SUBSTR(имя,1,1),'.',NULL,' ', NULL,
        NULL,NULL,SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),
          '.',NULL,' ',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')
          INTO fio FROM н_люди_по_падежам lp
        WHERE lp.члвк_ид = chel AND lp.падеж = padej;
     ELSIF priz = 2 THEN
       SELECT RTRIM(фамилия)||' '||DECODE(RTRIM(имя),'.',NULL,'
',NULL,RTRIM(имя))||' '||DECODE(RTRIM(отчество),'.',NULL,'
',NULL,RTRIM(отчество))
          INTO fio FROM н_люди_по_падежам lp WHERE lp.члвк_ид = chel
AND lp.падеж = padej;
     ELSE
       SELECT UPPER(RTRIM(фамилия))||'
'||DECODE(SUBSTR(имя,1,1),'.',NULL,'
',NULL,NULL,NULL,SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.',
NULL,' ',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')
          INTO fio FROM н_люди_по_падежам lp WHERE lp.члвк_ид = chel
AND lp.падеж = padej;
     END IF;
   ELSE
     IF priz = 0 THEN
     SELECT de-
cline(UPPER(RTRIM(фамилия))||'#'||DECODE(RTRIM(имя),'.',NULL,'
',NULL,RTRIM(имя))||'#'||DECODE(RTRIM(отчество),'.',NULL,'
',NULL,RTRIM(отчество)),пол,padej)
         INTO fio FROM н_люди WHERE ид = chel;
     -- Для формы с подписями
     ELSIF priz = 9 THEN
       SELECT DECODE(SUBSTR(имя,1,1),'.',NULL,'
',NULL,NULL,NULL,SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.',NULL,'
',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')||decline(Фамилия,пол,padej,'Ф')
          INTO fio FROM н_люди WHERE ид = chel;
     ELSIF priz = 1 THEN
       SELECT decline(Фамилия,пол,padej,'Ф')||'
'||DECODE(SUBSTR(имя,1,1),'.',NULL,'
',NULL,NULL,NULL,SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.',
NULL,' ',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')
```

```
            INTO fio FROM н_люди WHERE ид = chel;
      ELSIF priz = 2 THEN
        SELECT decline(RTRIM(фамилия)||'#'||RTRIM(имя)||'#'||
RTRIM(отчество),пол,padej)
           INTO fio FROM н_люди WHERE ид = chel;
      ELSE
        SELECT decline(UPPER(RTRIM(фамилия)),пол,padej,'Ф')||'
'||DECODE(SUBSTR(имя,1,1),'.',NULL,'
',NULL,NULL,NULL,SUBSTR(имя,1,1)||'.')||DECODE(SUBSTR(отчество,1,1),'.',
NULL,' ',NULL,NULL,NULL,SUBSTR(отчество,1,1)||'.')
            INTO fio FROM н_люди WHERE ид = chel;
      END IF;
    END IF;
  END IF;
  RETURN RTRIM(LTRIM(fio));
END Человек;
/
```

Функция `Decline` предназначена для вывода фамилии, имени и отчества человека в задаваемом падеже (`padej`).

```
CREATE OR REPLACE FUNCTION Decline(str in varchar2, sex in varchar2,
  pad in varchar2, lfm in varchar2 default 'ФИО') RETURN varchar2 IS
    p1 varchar2(1);
    p number(1);
    s1 varchar2(1);
    s number(1);
    x varchar2(1);
    l number(2,0);
    ul varchar2(1) := 'N';
    uf varchar2(1) := 'N';
    um varchar2(1) := 'N';
    lname varchar2(35);
    fname varchar2(35);
    mname varchar2(35);
    fullname varchar2(100) := ltrim(rtrim(str));
    pos number;
    lfm2 varchar2(3) := upper(substr((lfm),1,3));
    FUNCTION UppLow(st1 in varchar2, st2 in varchar2, st3 in varchar2)
      RETURN varchar2 IS
    BEGIN
      if st2 <> substr(st1,-1) then
```

```
    RETURN UPPER(st3);
  else
    RETURN st3;
  end if;
END UppLow;
--
FUNCTION complast(s in varchar2, t in varchar2) RETURN BOOLEAN IS
BEGIN
  if length(s) < length(t) then RETURN FALSE; end if;
      if substr(lower(s),-length(t)) = t then
        RETURN TRUE;
      else
        RETURN FALSE;
      end if;
END complast;
--
PROCEDURE change(s in out varchar2, n in number
                ,pzROD in varchar2
                ,pzDAT in varchar2
                        ,pzVIN in varchar2
                ,pzTVO in varchar2
                        ,pzPRE in varchar2) IS
BEGIN
  if p = 1 then
    null;
  else
    SELECT substr(s,1,length(s) - n) ||
          DECODE(p,2,pzROD,3,pzDAT,4,pzVIN,5,pzTVO,6,pzPRE) INTO s
      FROM DUAL;
  end if;
END change;
--
PROCEDURE prepare(fnm in out varchar2, lfm2 in out varchar2) IS
  pp varchar2(1) := substr(lfm2,1,1);
      name varchar2(35);
BEGIN
  pos := instr(fnm, '#');
  if pos > 0 then
    name := substr(fnm,1,pos - 1);
```

```
        fnm := ltrim(substr(fnm,pos + 1));
        pos := instr(fnm, '#');
      else
        name := fnm;
        fnm := null;
      end if;
      if pp = 'Ф' then
        lname := name;
        lfm2 := replace(lfm2,'Ф');
      elsif pp = 'И' then
        fname := name;
        lfm2 := replace(lfm2,'И');
      elsif pp = 'О' then
        mname := name;
        lfm2 := replace(lfm2,'О');
      end if;
    END;
  BEGIN
    if fullname is null then RETURN null; end if;
    if substr(lfm2,1,1) not in ('Ф','И','О') then RETURN str; end if;
    if NVL(substr(lfm2,2,1),'Ф') not in ('Ф','И','О') then RETURN str;
end if;
    if NVL(substr(lfm2,3,1),'Ф') not in ('Ф','И','О') then RETURN str;
end if;
    LOOP
      if fullname is not null then
        prepare(fullname,lfm2);
      else
        exit;
      end if;
    END LOOP;
    l := length(lname);
    p1 := upper(substr(pad,1,1));
    SELECT DECODE(p1,'И',1,'Р',2,'Д',3,'В',4,'Т',5,'П',6,0) INTO p FROM DUAL;
    if p = 0 then
      BEGIN
        p := to_number(p1);
      EXCEPTION
        WHEN OTHERS THEN RETURN str;
      END;
```

```
end if;
s1 := upper(substr(sex,1,1));
SELECT DECODE(s1,'M',1,'М',1,'Ж',0,'F',0,-1) INTO s FROM DUAL;
if s = -1 then RETURN str; end if;
if lname is not null then  -- фамилия
  if UPPER(lname) = lname then ul := 'Y'; end if;
  x := lower(substr(lname,-1));
  if s = 1 then  -- мужчины
    if x not in ('о','е','у','ю','и','х','э','ы') then
      if x = 'в' then
            change(lname,0,'а','у','а','ьм','е');
      elsif x = 'н' and complast(lname,'ин') then
            change(lname,0,'а','у','а','ьм','е');
      elsif x in ('б','г','д','ж','з','л','м','н','п'
                  ,'р','с','т','ф','ч','ш','щ') then
        change(lname,0,'а','у','а','ом','е');
      ELSIF x = 'ц' THEN
        IF lower(substr(lname,-2)) = 'ец' THEN
          change(lname,2,'ца','цу','ца','цом','цу');
        ELSE
          change(lname,0,'а','у','а','ом','у');
        END IF;
      elsif x = 'я' and not(complast(lname,'ия')
                  or complast(lname,'ая')) then
            change(lname,1,'и','е','ю','ей','е');
      elsif x = 'а' and not(complast(lname,'иа')
                  or complast(lname,'уа')) then
            change(lname,1,'и','е','у','ой','е');
      elsif x = 'ь' then
        if (complast(lname,'чь') or complast(lname,'щь')) then
            change(lname,1,'а','у','а','ем','е');
        ELSE
            change(lname,1,'я','ю','я','ем','е');
        END IF;
      elsif x = 'к' then
            if l > 4 and complast(lname,'ок') then
          change(lname,2,'ка','ку','ка','ком','ке');
            elsif l > 4 and (complast(lname,'лек')
                        or complast(lname,'рек')) then
```

```
                change(lname,2,'ька','ьку','ька','ьком','ьке');
              else
                change(lname,0,'а','у','а','ом','е');
              end if;
          elsif x = 'й' then
              if l > 4 then
                if (complast(lname,'ский') or complast(lname,'цкий')) then
                  change(lname,2,'ого','ому','ого','им','ом');
                elsif complast(lname,'ой') then
                  change(lname,2,'ого','ому','ого','им','ом');
                elsif complast(lname,'ей') AND NOT complast(lname,'вей') then
                  change(lname,2,'ея','ею','ея','еем','ее');
                elsif complast(lname,'эй') then
                  change(lname,2,'эя','эю','эя','эем','эе');
                elsif complast(lname,'ый') then
                  change(lname,2,'ого','ому','ого','ым','ом');
                elsif complast(lname,'ий') then
                  IF complast(lname,'чий') THEN
                    change(lname,2,'его','ему','его','им','ем');
                  ELSIF complast(lname,'лий') THEN
                    change(lname,1,'я','ю','я','ем','е');
                  ELSE
                    change(lname,2,'ого','ому','ого','им','ом');
                  END IF;
                elsif substr(lname,-3) in
('рий','жий','лий','вий','дий','бий','гий','зий','мий','ний','пий','сий',
'фий','хий') then
                   change(lname,1,'я','ю','я','ем','и');
                end if;
              else
                change(lname,1,'я','ю','я','ем','е');
              end if;
          end if;
        end if;
      elsif s = 0 then  -- женщины
        if lower(substr(lname,-3)) in
('ова','ева','ина','ена','ына','ёва') then
          change(lname,1,'ой','ой','у','ой','ой');
        elsif complast(lname,'ая') then
              change(lname,2,'ой','ой','ую','ой','ой');
```

```
      end if;
    end if;
  end if;
  if fname is not null then  -- имя
    if UPPER(fname) = fname then uf := 'Y'; end if;
    x := lower(substr(fname,-1));
    if s = 1 then  -- мужчины
      if x not in ('о','е','и','у') then
        if x in ('б','в','г','д','ж','к','м','н','п','р','с','т'
                    ,'ф','х','ц','ч','ш','щ') then
              IF lower(fname) = 'лев' THEN
                change(fname,3,'Льва','Льву','Льва','Львом','Льве');
              ELSE
                change(fname,0,'а','у','а','ом','е');
              END IF;
        elsif x = 'я' then
              if complast(fname,'ья') then
                change(fname,1,'и','е','ю','ей','е');
              else
             change(fname,0,'и','е','ю','ей','е');
          end if;
        elsif x = 'й' then
              if complast(fname,'ай') then
                change(fname,1,'я','ю','я','ем','е');
              else
                if complast(fname,'ей') then
                  change(fname,1,'я','ю','я','ем','е');
                else
                 change(fname,1,'я','ю','я','ем','и');
                end if;
              end if;
            elsif x = 'ь' then
              change(fname,1,'я','ю','я','ем','е');
        elsif x = 'а' then
              IF complast(fname,'та') THEN
                change(fname,1,'ы','е','у','ой','е');
              -- типа Данила
              ELSIF complast(fname,'ла') THEN
                change(fname,1,'ы','е','у','ой','е');
```

```
                ELSE
                  change(fname,1,'и','е','у','ей','е');
                END IF;
          elsif x = 'л' then
                if complast(fname,'авел') then
              change(fname,2,'ла','лу','ла','лом','ле');
                else
              change(fname,0,'а','у','а','ом','е');
                end if;
              end if;
        end if;
      elsif s = 0 then  -- женщины
        if x = 'а' then
          if lower(substr(fname,-2)) in
('га','ха','ка','ша','ча','ща','жа') then
            change(fname,1,'и','е','у','ой','е');
          else
            change(fname,1,'ы','е','у','ой','е');
          end if;
        elsif x = 'я' then
          if complast(fname,'ия') then
            change(fname,1,'и','и','ю','ей','и');
          else
            change(fname,1,'и','е','ю','ей','е');
          end if;
        elsif x = 'ь' then
          if complast(fname,'вь') then
            change(fname,1,'и','и','ь','ью','и');
          else
            change(fname,1,'и','и','ь','ью','ье');
          end if;
        end if;
          end if;
    end if;
    if mname is not null then  -- отчество
      if UPPER(mname) = mname then um := 'Y'; end if;
      x := lower(substr(mname,-1));
      if s = 1 then  -- мужчины
        if x = 'ч' then
```

```
          change(mname,0,'а','у','а','ем','е');
        end if;
      elsif s = 0 then  -- женщины
        if x = 'а' then
          change(mname,1,'ы','е','у','ой','е');
        end if;
      end if;
    end if;
    -- окончательная конкатенация
    lfm2 := upper(substr((lfm),1,3));
    pos := 1;
    LOOP
      if pos > 1 then fullname := fullname || ' '; end if;
      SELECT fullname ||

DECODE(substr(lfm2,pos,1),'Ф',DECODE(ul,'Y',UPPER(lname),lname)

,'И',DECODE(uf,'Y',UPPER(fname),fname)

,'О',DECODE(um,'Y',UPPER(mname),mname))
        INTO fullname FROM DUAL;
      pos := pos + 1;
      if pos > length(lfm2) then EXIT; end if;
    END LOOP;
    RETURN fullname;
  END Decline;
```

23.2. Пакет для просмотра успеваемости

Фрагмент пакета `pk_dek_ocenki`, предназначеного для обеспечения работы деканатов.

Для работы этого пакета были созданы три рабочих таблицы, не входящих в инфологическую модель (см. рис. 21.7):

```
CREATE TABLE HT_ОЦЕНКИ_СТУДЕНТА
( ДИСЦИПЛИНА  VARCHAR2(200),
  СЕМЕСТР     NUMBER(2),
  КОНТР       VARCHAR2(8),
  ОЦЕНКА      VARCHAR2(8),
  УЧЕБНЫЙ_ГОД CHAR(9),
```

```
  ВР_ИД       NUMBER(9),
  СЭС_ИД      NUMBER(9),
  ВЕД_ИД      NUMBER(9) );


CREATE TABLE НТ_ОЦЕНКИ_КАРТОЧЕК_СТУД
( ЧЛВК_ИД     NUMBER(9),
  ГРУППА      VARCHAR2(4),
  КУРС        NUMBER(1),
  СЕМЕСТР     NUMBER(2),
  ДИСЦИПЛИНА  VARCHAR2(20),
  ЧАСЫ        VARCHAR2(20),
  ЭКЗ         VARCHAR2(4),
  ЗАЧ         VARCHAR2(4),
  КР          VARCHAR2(4),
  ЭСТ_ИД      NUMBER(9),
  ЭКЗ_ИД      NUMBER(9),
  ЗАЧ_ИД      NUMBER(9),
  КР_ИД       NUMBER(9) );


CREATE TABLE НТ_СПРАВКА_О_ОЦЕНКАХ
( СЕМЕСТР     NUMBER(2),
  ДИСЦИПЛИНА  VARCHAR2(200),
  ЧАСЫ        VARCHAR2(8),
  КОНТР       VARCHAR2(8),
  ОЦЕНКА      VARCHAR2(8),
  ДАТА        DATE,
  ЭСТ_ИД      NUMBER(9),
  СЭС_ИД      NUMBER(9),
  БУКВА       VARCHAR2(20) );
```

Спецификация пакета.

```
CREATE OR REPLACE PACKAGE pk_dek_ocenki IS
  /* Заполняет временную таблицу НТ_ОЦЕНКИ_СТУДЕНТА его оценками, полу-
ченными за все обучение в университете */
  PROCEDURE оценки_студента (п_члвк_ид IN NUMBER, п_вид_обуч_ид IN
NUMBER);
  /* Процедура заполняет временную таблицу НТ_СПРАВКА_О_ОЦЕНКАХ оценками
студента */
  PROCEDURE справка_о_оценках (п_члвк_ид IN NUMBER);
  /* Процедура заполняет таблицу НТ_ОЦЕНКИ_КАРТОЧЕК_СТУД выписками оценок
     для личных карточек студентов */
  PROCEDURE оценки_для_карточек_студентов
```

```
    (п_учгод IN VARCHAR2, п_курс IN NUMBER, п_факультет IN NUMBER);
/*****************************************************************/
  /* Функция возвращает дату проведения итогового контроля */
  FUNCTION ведомости_дата (п_группа VARCHAR2, п_сэс_ид NUMBER) RETURN
DATE;
  /* Функция возвращает номер семестра по дате */
  FUNCTION номер_семестра (п_дата DATE) RETURN NUMBER;
END pk_dek_ocenki;
```

Тело пакета.

```
CREATE OR REPLACE PACKAGE BODY pk_dek_ocenki IS
  /* Заполняет временную таблицу НТ_ОЦЕНКИ_СТУДЕНТА его оценками,
полученными за все обучение в университете */
  PROCEDURE оценки_студента (п_члвк_ид NUMBER, п_вид_обуч_ид NUMBER) AS
    п_учебный_год  CHAR(9);
    п_курс         NUMBER(2);
    п_сем_осень    DATE;
    п_сем_весна    DATE;
  BEGIN
    /* Вставляем все дисциплины, которые должен был сдать студент
по цепочке планов */
    DELETE FROM нт_оценки_студента;
    INSERT INTO нт_оценки_студента (дисциплина,семестр,контр,учебный_
год,вр_ид,сэс_ид)
    SELECT дис.наименование, эст.номер_семестра,
вр.аббревиатура,план.учебный_год,сэс.вр_ид,сэс.ид
    FROM н_планы план, н_строки_планов спл,
      н_элементы_строк эст, н_содержания_элементов_строк сэс,
      н_дисциплины дис, н_виды_работ вр
    WHERE план.ид = спл.план_ид
      AND спл.ид = эст.спл_ид
      AND эст.ид = сэс.эст_ид
      AND спл.дис_ид = дис.ид
      AND сэс.вр_ид = вр.ид
      AND спл.ком_ид <> 5
      AND сэс.вр_ид IN (5,6,7,8)
      AND план.ид IN
        (SELECT план.ид FROM н_планы план
         CONNECT BY PRIOR план.план_ид = план.ид
         START WITH план.ид =
```

```
           (SELECT план.ид FROM н_ученики учн, н_планы план,
н_направления_специал нап
            WHERE план.ид = учн.план_ид
              AND план.напс_ид = нап.ид
              AND учн.члвк_ид = п_члвк_ид
              AND учн.вид_обуч_ид = п_вид_обуч_ид
              AND нап.квал_ид <> 5
              AND признак IN ('обучен','продлен')
              AND состояние = 'утвержден'
              AND начало =
                (SELECT max(начало) FROM н_ученики
                 WHERE признак IN ('обучен','продлен')
                   AND состояние = 'утвержден'
                   AND члвк_ид = учн.члвк_ид
                   AND вид_обуч_ид = учн.вид_обуч_ид)));
  -- Определяем текущий учебный год
  SELECT учебный_год INTO п_учебный_год
  FROM н_учебные_года WHERE sysdate BETWEEN начало AND конец;
  -- Определяем текущий план студента
  BEGIN
    SELECT план.курс INTO п_курс
    FROM н_ученики учн, н_планы план
    WHERE учн.план_ид = план.ид
      AND члвк_ид = п_члвк_ид
      AND вид_обуч_ид = п_вид_обуч_ид
      AND sysdate BETWEEN начало AND конец
      AND признак IN ('обучен','проден')
      AND состояние = 'утвержден';
    п_сем_осень := to_date('01.01.'||substr(п_учебный_год,6,9),'DD.MM.YYYY');
    п_сем_весна := to_date('01.06.'||substr(п_учебный_год,6,9),'DD.MM.YYYY');
    -- Удаляем дисциплины весеннего семестра
    IF sysdate <= п_сем_весна THEN
      DELETE FROM нт_оценки_студента
      WHERE учебный_год = п_учебный_год
        AND семестр = 2*п_курс;
    END IF;
    -- Удаляем дисциплины осеннего семестра
    IF sysdate <= п_сем_осень THEN
      DELETE FROM нт_оценки_студента
```

```
    WHERE учебный_год = п_учебный_год
      AND семестр = 2*п_курс-1;
  END IF;
EXCEPTION
  WHEN no_data_found THEN null;
END;
-- Проставляем оценки студента
UPDATE нт_оценки_студента t
  SET оценка = (SELECT оценка
                FROM н_ведомости
                WHERE члвк_ид = п_члвк_ид
                AND сэс_ид = t.сэс_ид
                AND состояние = 'актуальна'
                AND дата =
                  (SELECT max(дата)
                   FROM н_ведомости
                   WHERE члвк_ид = п_члвк_ид
                   AND сэс_ид = t.сэс_ид
                   AND состояние = 'актуальна')),
      вед_ид = (SELECT ид
                FROM н_ведомости
                WHERE члвк_ид = п_члвк_ид
                AND сэс_ид = t.сэс_ид
                AND состояние = 'актуальна'
                AND дата =
                  (SELECT max(дата)
                   FROM н_ведомости
                   WHERE члвк_ид = п_члвк_ид
                   AND сэс_ид = t.сэс_ид
                   AND состояние = 'актуальна'));
-- Вставляем оценки, не привязанные к плану
INSERT INTO нт_оценки_студента (дисциплина,семестр,контр,оценка)
SELECT дис.наименование,эст.номер_семестра,вр.аббревиатура,вед.оценка
FROM н_строки_планов спл, н_элементы_строк эст,
  н_содержания_элементов_строк сэс,
  н_дисциплины дис, н_виды_работ вр, н_ведомости вед
WHERE спл.ид = эст.спл_ид
  AND эст.ид = сэс.эст_ид
  AND спл.дис_ид = дис.ид
```

```
      AND сэс.вр_ид = вр.ид
      AND спл.ком_ид <> 5
      AND сэс.вр_ид IN (5,6,7,8)
      AND сэс.ид = вед.сэс_ид
      AND вед.состояние = 'актуальна'
      AND вед.члвк_ид = п_члвк_ид
      AND NOT EXISTS (SELECT * FROM нт_оценки_студента t WHERE вед_ид = вед.ид);
      COMMIT;
  END оценки_студента;
  /* Процедура заполняет временную таблицу НТ_СПРАВКА_О_ОЦЕНКАХ оценками
студента */
  PROCEDURE справка_о_оценках (п_члвк_ид IN NUMBER) AS
  BEGIN
    DELETE FROM нт_справка_о_оценках;
    -- Последние оценки по семестрам
    INSERT INTO нт_справка_о_оценках
    SELECT
с.номер_семестра,е.наименование,'',f.аббревиатура,a.оценка,a.дата,c.ид,''
, a.буква
    FROM н_ведомости a,н_содержания_элементов_строк b,н_элементы_строк
c,н_строки_планов d,н_дисциплины e,н_виды_работ f
    WHERE a.сэс_ид = b.ид AND b.эст_ид = c.ид AND c.спл_ид = d.ид AND
d.дис_ид = e.ид AND b.вр_ид = f.ид AND
    члвк_ид = п_члвк_ид
    AND a.дата = (SELECT max(дата) FROM н_ведомости WHERE члвк_ид =
п_члвк_ид AND сэс_ид = b.ид);
    -- Часы
    UPDATE нт_справка_о_оценках t SET часы =
    (SELECT
sum(decode(вр_ид,1,объем,2,объем,3,объем,4,объем,0))||'/'||sum(decode(вр_
ид,1,объем,2,объем,3,объем,0))
     FROM н_содержания_элементов_строк WHERE эст_ид = t.эст_ид);
    COMMIT;
 END справка_о_оценках;
  /* Процедура заполняет таблицу НТ_ОЦЕНКИ_КАРТОЧЕК_СТУД выписками оценок
     для личных карточек студентов */
  PROCEDURE оценки_для_карточек_студентов
    (п_учгод IN VARCHAR2, п_курс IN NUMBER, п_факультет IN NUMBER) AS
    п_конец_учгода DATE;
  BEGIN
    -- Определяем конец учебного года
```

```
    SELECT конец
      INTO п_конец_учгода
    FROM н_учебные_года
    WHERE учебный_год = п_учгод;
    -- Вставляем шаблоны выписок без оценок
    DELETE FROM нт_оценки_карточек_студ;
    INSERT INTO нт_оценки_карточек_студ
      (члвк_ид, группа, курс, семестр, дисциплина, часы, эст_ид)
    SELECT члвк_ид,
      учн.группа,
      курс,
      номер_семестра,
      дис.короткое_имя,
      SUM(decode( вр_ид,1,объем,2,объем,3,объем,4,объем,0))||'/'||
        SUM(decode( вр_ид,1,объем,2,объем,3,объем,0)) Объем,
      эст.ид
    FROM н_планы план,
      н_строки_планов спл,
      н_элементы_строк эст,
      н_содержания_элементов_строк сэс,
      н_дисциплины дис,
      н_ученики учн
    WHERE план.ид = спл.план_ид
      AND спл.ид = эст.спл_ид
      AND эст.ид = сэс.эст_ид
      AND спл.дис_ид = дис.ид
      AND план.ид = учн.план_ид
      AND учн.конец = п_конец_учгода
      AND учн.состояние = 'утвержден'
      AND учн.признак IN ('обучен','продлен')
      AND план.отд_ид = п_факультет
      AND план.учебный_год = п_учгод
      AND план.курс = п_курс
    GROUP BY
члвк_ид,учн.группа,дис.короткое_имя,курс,номер_семестра,эст.ид;
    -- Определяем сэс_ид контролей
    UPDATE нт_оценки_карточек_студ окс
    SET экз_ид =
      (SELECT сэс.ид
```

```
    FROM н_содержания_элементов_строк сэс
    WHERE эст_ид = окс.эст_ид
      AND вр_ид = 5
      AND сэс.номер_контроля = 1);
UPDATE нт_оценки_карточек_студ окс
SET зач_ид =
  (SELECT сэс.ид
   FROM н_содержания_элементов_строк сэс
   WHERE эст_ид = окс.эст_ид
     AND сэс.номер_контроля = 1
     AND вр_ид = 6);
UPDATE нт_оценки_карточек_студ окс
SET кр_ид =
  (SELECT сэс.ид
   FROM н_содержания_элементов_строк сэс
   WHERE эст_ид = окс.эст_ид
     AND вр_ид IN (7,8));
-- Дисциплины, для которых не предусмотрено соответствующих контролей
UPDATE нт_оценки_карточек_студ  SET экз = '-' WHERE экз_ид IS NULL;
UPDATE нт_оценки_карточек_студ  SET зач = '-' WHERE зач_ид IS NULL;
UPDATE нт_оценки_карточек_студ  SET кр = '-' WHERE кр_ид IS NULL;
-- Вставляем оценки студентов
UPDATE нт_оценки_карточек_студ окс
SET экз =
  (SELECT DECODE(оценка,'зачет','зач',оценка)
   FROM н_ведомости
   WHERE члвк_ид = окс.члвк_ид
     AND сэс_ид = экз_ид
     AND состояние = 'актуальна'
     AND оценка IN ('5','4','3','зачет','осв')
     AND дата =
       (SELECT max(дата)
        FROM н_ведомости
        WHERE члвк_ид = окс.члвк_ид
          AND сэс_ид = экз_ид
          AND состояние = 'актуальна'))
WHERE экз_ид IS NOT NULL;
UPDATE нт_оценки_карточек_студ окс
SET зач =
```

```
      (SELECT DECODE(оценка,'зачет','зач',оценка)
       FROM н_ведомости
       WHERE члвк_ид = окс.члвк_ид
         AND сэс_ид = зач_ид
         AND состояние = 'актуальна'
         AND оценка IN ('5','4','3','зачет','осв')
         AND дата =
           (SELECT max(дата)
            FROM н_ведомости
            WHERE члвк_ид = окс.члвк_ид
              AND сэс_ид = зач_ид
              AND rownum = 1
              AND состояние = 'актуальна'))
    WHERE зач_ид IS NOT NULL;
    UPDATE нт_оценки_карточек_студ окс
    SET кр =
      (SELECT DECODE(оценка,'зачет','зач',оценка)
       FROM н_ведомости
       WHERE члвк_ид = окс.члвк_ид
         AND сэс_ид = кр_ид
         AND состояние = 'актуальна'
         AND оценка IN ('5','4','3','зачет','осв')
         AND дата =
           (SELECT max(дата)
            FROM н_ведомости
            WHERE члвк_ид = окс.члвк_ид
              AND сэс_ид = кр_ид
              AND состояние = 'актуальна'))
    WHERE кр_ид IS NOT NULL;
    COMMIT;
  END оценки_для_карточек_студентов;
  /* Функция возвращает дату проведения итогового контроля */
  FUNCTION ведомости_дата (п_группа VARCHAR2, п_сэс_ид NUMBER) RETURN
DATE IS
    п_дата VARCHAR2(200);
  BEGIN
    SELECT сес.дата
      INTO п_дата
    FROM н_сессия сес,
```

```
      н_люди члвк,
      н_содержания_элементов_строк сэс,
      н_элементы_строк эст,
      н_строки_планов спл,
      н_планы план
    WHERE члвк.ид = сес.члвк_ид
      AND сес.сэс_ид = сэс.ид
      AND сэс.эст_ид = эст.ид
      AND спл.план_ид = план.ид
      AND спл.ид = эст.спл_ид
      AND (план.учебный_год, эст.номер_семестра, спл.дис_ид, эст.отд_ид,
сэс.вр_ид, сэс.номер_контроля) =
       (SELECT план.учебный_год, эст.номер_семестра, спл.дис_ид,
эст.отд_ид, сэс.вр_ид, сэс.номер_контроля
        FROM н_планы план,
          н_строки_планов спл,
          н_элементы_строк эст,
          н_содержания_элементов_строк сэс
        WHERE план.ид = спл.план_ид
          AND спл.ид = эст.спл_ид
          AND эст.ид = сэс.эст_ид
          AND сэс.ид = п_сэс_ид)
    AND группа = п_группа;
    RETURN(п_дата);
  EXCEPTION
    WHEN no_data_found THEN RETURN(null);
  END ведомости_дата;
  /* Функция возвращает номер семестра по дате */
  FUNCTION номер_семестра (п_дата DATE) RETURN NUMBER IS
    п_номер_семестра NUMBER;
  BEGIN
    SELECT
DECODE(to_char(п_дата,'MM'),'09',1,'10',1,'11',1,'12',1,'01',1,'02',1,0)
      INTO п_номер_семестра
    FROM dual;
    RETURN(п_номер_семестра);
  END;
END pk_dek_ocenki;
**************************************************
```

```
--EXECUTE pk_dek_ocenki.оценки_студента(126287,1);
--EXECUTE pk_dek_ocenki.справка_о_оценках (126287);
--EXECUTE pk_dek_ocenki.оценки_для_карточек_студентов('2007/2008', 3,
703)
--SELECT pk_dek_ocenki.ведомости_дата (3103,121951 ) FROM dual;
PK_DEK_OCENKI.ВЕДОМОСТИ_ДАТА(3
------------------------------
05.01.2005

SELECT pk_dek_ocenki.номер_семестра('10.05.2008') FROM dual;

PK_DEK_OCENKI.НОМЕР_СЕМЕСТРА('
------------------------------
                             0
```

Литература

1. Дейт К. Дж. Введение в системы баз данных, 8-е издание / Пер. с англ. — М.: Издательский дом "Вильямс", 2005. — 1328 с.: ил.
2. Джеймс Р. Г., Вайнберг П. Н. SQL: полное руководство. — К.: BHV, 2000. — 608 с.
3. Грабер М. SQL. — М.: ЛОРИ, 2007. — 644 с.
4. Кириллов В. В. Основы проектирования реляционных баз данных. Учебное пособие. — СПб.: ИТМО, 1994. — 88 с. http://www.citforum.ru.
5. Кириллов В. В., Громов Г. Ю. Структуризированный язык запросов (SQL). Учебное пособие. — СПб.: ИТМО, 1995. — 92 с. **http://www.citforum.ru**.
6. Клайн К. при участии Клайна Д. и Ханта Бр. SQL. Справочник. 2-е издание / Пер. с англ. — М.: КУДИЦ-ОБРАЗ, 2006 — 832 с.
7. Чен Питер Пин-Чен. Модель "Сущность-связь" — шаг к единому представлению данных. СУБД №3 1995, с. 137—158.
8. Codd, E. F. "Extending the Relational Database Model to Capture More Meaning". IBM Research Report RJ2599 (August 6th, 1979). Republished in ACM Transactions on Database Systems 4(4), December 1979.
9. Codd, E. F. (1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6): 377–387.
10. Oracle Database 10g. Программирование на языке PL/SQL. Скотт Урман, Рон Хардман, Майкл МакЛафлин. — Питер, Лори, 2007. — 816 с.

ПРИЛОЖЕНИЯ

Приложение А. Инструментальные средства разработки и выполнения

Приложение Б. Описание содержимого компакт-диска

Приложение А

Инструментальные средства разработки и выполнения

A1. Oracle Database Express Edition

A1.1. Общие сведения

Система управления базами данных Oracle 10g Express Edition (Oracle Database XE) — это бесплатная версия одной из наиболее совершенных систем управления реляционными базами данных. Ее можно бесплатно скачать и легко установить, бесплатно распространять, бесплатно поставить. Она легко администрируется, и для нее легко создавать приложения.

В состав Oracle Database XE включено средство создания Web-приложений Oracle Application Express (APEX).

Редакция Oracle Database XE создана на основе исходного программного кода СУБД Oracle Database 10g Release 2 и полностью совместима с семейством программных продуктов Oracle Database, включая Oracle Standard Edition One, Oracle Standard Edition и Oracle Enterprise Edition. Пользователи имеют возможность начать работу с базовой редакции, а впоследствии модернизировать ее до промышленных редакций Oracle Database 10g и перевести свои приложения на работу с новыми редакциями СУБД без изменения кода приложения.

Oracle Database XE предоставляет те же интерфейсы SQL и PL/SQL, что и во всех остальных версиях Oracle Database 10g, а также широкий спектр программных интерфейсов. Например, предоставляется полная поддержка разработки и развертывания приложений для разработчиков, работающих на платформах Java, .NET, PHP и Windows.

Редакция Oracle Database XE выпускается для 32-разрядных операционных систем Linux и Windows на основе архитектуры Intel/AMD x86 и может быть установлена на любой поддерживаемой ими аппаратной платформе.

Oracle Database XE имеет следующие ограничения:

- не более 4 Гбайт пользовательских данных;
- может использовать не более 1 Гбайт оперативной памяти;
- может использовать только один процессор (или 1 ядро процессора).

Существует два варианта Oracle Database 10g XE:

- Oracle Database 10g Express (Western European) Edition — БД в кодировке LATIN1 западно-европейских языков;
- Oracle Database 10g Express Edition (Universal) — поддержка кодировки Unicode и, как следствие, поддержка всех языков.

Для поддержки в приложениях русского языка необходимо использовать либо второй вариант (с поддержкой кодировки Unicode), либо русифицированную версию Western European (см. *разд. A2*). При использовании кодировки Unicode возникают проблемы с именами. В *разд. 4.4.2* оговаривалось, что имена в СУБД Oracle ограничиваются длиной в 30 байт и число символов имени зависит от их набора. Поэтому в Oracle Database 10g Express Edition (Universal) нельзя загрузить существующие базы данных, где встречаются русские имена объектов (например, ограничений) с числом символов более 15, а при создании новых баз данных надо существенно ограничивать длину имен их объектов.

При инсталляции Oracle Database 10g XE (см. *разд. A2*) производится ее конфигурация и установка ярлычка на рабочий стол. Она предоставляет пользователю отличный Web-интерфейс администрирования, позволяющий с помощью мыши добавлять пользователей, просматривать таблицы и вносить данные, отслеживать производительность и выполнять десятки других повседневных задач (рис. A1). Для продвинутых пользователей имеется отличная, основанная на Web SQL-система ввода данных с быстрым досту-пом к вкладкам для объяснения запросов, сохранения SQL-предложений и просмотра истории команд.

Одно из ярких достоинств данного релиза — солидный вес поставляемой с ним документации. Нравится ли вам PHP, Java, XML или .NET, к вашим услугам под Oracle 10g Express Edition пробная документация, заботливо поддерживающая вас под руку во время ваших первых шагов по созданию скриптов. Но вот ваше мастерство возросло, и тут уже наготове XE: двухдневное руководство по администрированию проведет вас через управление ресурсами, безопасность, создание резервных копий и многое другое. Вы и не заметите, как потратите неделю только на это захватывающее чтение.

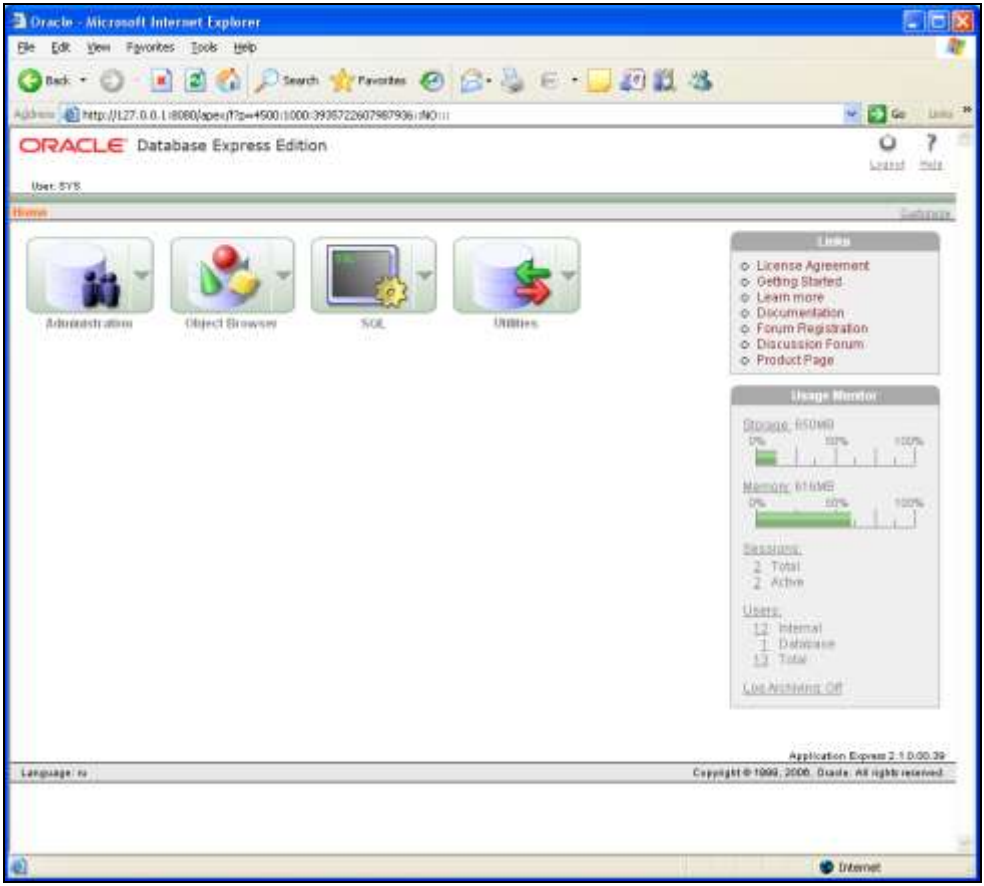


Рис. А1. Web-интерфейс администрирования Oracle Database 10g XE

Для доступа к документации необходимо:

- в Windows в меню **Start** выберите **Programs** (или **All Programs**), затем **Oracle Database 10g Express Edition**, затем **Get Help**;
- в Linux нажать **Application** (в Gnome) или **K** (в KDE), перейти к **Oracle Database 10g Express Edition** и выбрать **Get Help**.

В частности, обязательно просмотрите следующие книги.

- 2-дневное руководство администратора Oracle Database Express Edition (**http://www.oranet.ru/OraDoc10gXE/admin.102/b25107/toc.htm**). В этой книге обсуждаются общие повседневные задачи администрирования.
- 2-дневное руководство разработчика Oracle Database Express Edition (**http://www.oranet.ru/OraDoc10gXE/appdev.102/b25108/toc.htm**). В этой

книге показываются преимущества среды разработки Oracle Database Express Edition.

- Application Express — дополнительное 2-дневное руководство разработчика Oracle Database Express Edition (**http://www.oranet.ru/OraDoc10gXE/appdev.102/b25310/toc.htm**). Книга содержит подборку учебных пособий, в которых описан процесс построения Web-приложений, использующих клиента Oracle Application Express.
- PHP — дополнительное 2-дневное руководство разработчика Oracle Database Express Edition (**http://www.oranet.ru/OraDoc10gXE/appdev.102/b25317/toc.htm**). Учебное пособие, в котором описаны процессы загрузки и установки Apache и драйверов Zend Core PHP, использования PHP для установления связи с Oracle Database XE; также показано, как использовать PHP для разработки простого приложения, которое получает доступ к данным и модифицирует их.
- Java — дополнительное 2-дневное руководство разработчика Oracle Database Express Edition (**http://www.oranet.ru/OraDoc10gXE/appdev.102/b25320/toc.htm**). Учебное пособие, в котором описано, как использовать Java и JDBC для установления связи с Oracle Database XE; показано, как разработать простое Java-приложение, которое получает доступ к данным и модифицирует их.
- .NET — дополнительное 2-дневное руководство разработчика Oracle Database Express Edition (**http://www.oranet.ru/OraDoc10gXE/appdev.102/b25312/toc.htm**). В этой книге описаны ключевые возможности Oracle Data Provider for .NET и Oracle Developer Tools for Visual Studio .NET. Книга проведет вас через этапы установки и конфигурирования; покажет, как строить приложения, использующие Oracle Data Provider for .NET и Oracle Developer Tools for Visual Studio .NET, и как создавать и использовать хранимые процедуры PL/SQL и .NET.

A1.2. Состав Oracle Database XE

Oracle Database XE состоит из двух компонентов.

- **Сервер Oracle Database XE.** Серверная часть Oracle Database XE включает одновременно базу данных Oracle и инструменты для управления ею. Она также содержит компонент клиента Oracle Database XE, поэтому вы можете соединяться с базой данных с того же компьютера, на который установлен компонент сервера; а также администрировать базу данных и писать приложения Java, .NET, PHP и Oracle Application Express.

- **Клиент Oracle Database XE.** Установите компонент клиента Database XE на тех удаленных компьютерах, с которых хотите соединяться с сервером Oracle Database XE. Клиент базы данных Oracle включает драйверы, сетевые компоненты и инструменты для удаленного администрирования и создания приложений Java, .NET, PHP и Oracle Application Express. Установка этого компонента необходима только на тех компьютерах, на которых не стоит сервер Oracle Database XE.

Дополнительно можно использовать следующие инструменты в среде разработки.

- **PHP.** Открытый, сервер-ориентированный скриптовый язык, разработанный для создания Web-приложений, который может быть встроен в HTML. Вы можете использовать любой из следующих программных продуктов PHP:

  - **PHP** загрузите и установите с:

    http://www.php.net/

    PHP можно установить как на сервере Oracle Database XE, так и на клиенте Oracle Database XE.

  - **Zend Core** для Oracle загрузите и установите с:

    http://www.oracle.com/technology/tech/php/zendcore/index.html

    Установите Zend Core для Oracle на том же компьютере, на котором установлен сервер Oracle Database XE. Zend Core для Oracle не работает на удаленных компьютерах, которые используют клиент Oracle Database XE.

- Инструменты разработчика Oracle для **Visual Studio .NET**. Интегрированный "Add-in" для Visual Studio .NET, который разработчики баз данных Oracle могут использовать для проектирования приложений баз данных. Для получения более полной информации, включая дистрибутив, посетите:

  http://www.oracle.com/technology/tech/dotnet/tools/index.html

  Установите инструменты разработчика Oracle для Visual Studio .NET на компьютер, на котором установлен клиент Oracle Database XE.

После установки Oracle Database XE вы можете управлять ею с помощью управляющей консоли графического интерфейса пользователя, который описан в 2-дневном руководстве администратора Oracle Database Express Edition.

Для получения дополнительной информации по Oracle Database XE вы можете посетить следующие сайты:

- домашняя страница Oracle Database XE на Oracle Technology Network:

  http://www.oracle.com/technology/products/database/xe

- библиотека документации Oracle Database XE:

  http://www.oracle.com/technology/xe/documentation

- дискуссионный форум:

  http://www.oracle.com/technology/xe/forum

  Прежде чем начать пользоваться форумом, вам необходимо будет зарегистрировать Oracle Database XE.

A1.3. Требования к программному обеспечению

Архитектура системы: Intel (x86).

Операционная система: любая из 32-разрядных операционных систем:

- Windows 2000 Service Pack 4 или более поздняя;
- Windows Server 2003;
- Windows XP Professional Service Pack 1 или более поздняя.

Сетевой протокол: TCP/IP.

Свободное место на диске: сервер — минимум 1,6 Гбайт, клиент — 75 Мбайт.

Оперативная память: минимум 256 Мбайт (рекомендуется 512 Мбайт).

A1.4. Взаимодействие с межсетевыми экранами

При установке Oracle Database XE на компьютер с операционной системой Windows XP Service Pack 2 или Windows Server 2003 Service Pack 1 для подключения к нему удаленного компьютера проверьте, чтобы брандмауэр не был настроен на блокировку соединений по следующим портам входа, которые являются портами по умолчанию Oracle Database XE:

- 1521: слушатель сети базы данных Oracle;
- 2030: службы Oracle для Microsoft Transaction Server;
- 8080: HTTP-порт для Oracle XML DB и графического пользовательского интерфейса Oracle Database XE.

А1.5. Требования к надстройкам Oracle Database для платформы .NET

Если вы планируете использовать надстройки Oracle Database для .NET, тогда перед установкой Oracle Database XE вы должны установить следующее программное обеспечение на компьютеры сервера и клиента.

- ❒ На компьютер, на который планируете установить сервер Oracle Database XE, установите .NET Framework 1.x. Установить .NET Framework 1.x необходимо до установки сервера Oracle Database XE.
- ❒ На компьютер, на котором планируется разрабатывать хранимые процедуры .NET, установить следующее программное обеспечение:
  - сервер или клиент Oracle Database XE;
  - Visual Studio .NET 2003, включая .NET Framework 1.x.

    Установите Visual Studio .NET 2003 на компьютер-клиент до установки клиента Oracle Database XE или инструментов разработчика Oracle для Visual Studio .NET;
  - инструменты разработчика Oracle для Visual Studio .NET.

    Эти требования относятся только к проектированию и развертыванию приложений. Для выполнения программы они не являются необходимыми. Для выполнения программы необходима установка .NET Framework 1.x на сервере базы данных. Установленная Visual Studio .NET 2003 не является обязательной.

А1.6. Требования к Web-браузеру

Убедитесь, что для компонентов клиента и сервера Oracle Database XE, Web-браузер поддерживает JavaScript и стандарты HTML 4.0 и CSS 1.0. Убедитесь также, что cookies включены. Данным требованиям удовлетворяют следующие браузеры:

- ❒ Microsoft Internet Explorer 6.0 или более свежая версия;
- ❒ Netscape Navigator 7.2 или более свежая версия;
- ❒ Mozilla 1.7 или более свежая версия;
- ❒ Firefox 1.0 или более свежая версия;
- ❒ Oracle Application Express не поддерживает Corel SVG Viewer.

Настройка Web-браузера

Перед тем как запустить Oracle Database XE, вам необходимо сконфигурировать Web-браузер так, чтобы он мог соединиться с домашней страницей Oracle Database XE.

Чтобы настроить *Microsoft Internet Explorer* для соединения с домашней страницей Oracle Database XE:

1. В меню **Start** выберите **Control Panel**, затем **Internet Options**.
2. В диалоговом окне **Intenet Options** перейдите на вкладку **Security**.
3. На вкладке **Security** выберите **Local Intranet** и затем выберите **Sites**.
4. В окне **Local Intranet** нажмите **Advanced**.
5. В поле **Add this Web site to the zone** добавьте адрес 127.0.0.1.
6. Нажмите **OK**.

Чтобы настроить *Netscape Navigator* или *Mozilla* для соединения с домашней страницей Oracle Database XE:

1. В меню **Edit** выберите **Preferences**.
2. В дереве **Category** нажмите категорию **Advanced**, чтобы ее раскрыть, и затем **Proxies**.
3. При использовании прокси-сервера добавьте следующие настройки в поле **No Proxy For**: 127.0.0.1.
4. Нажмите **OK**.

Чтобы настроить *Firefox* для соединения с домашней страницей Oracle Database XE:

1. В меню **Tools** выберите **Options**.
2. На вкладке **General** выберите **Connection Settings**.
3. При использовании прокси-сервера добавьте следующие настройки в поле **No Proxy For**: 127.0.0.1.
4. Нажмите **OK**.

A2. Установка сервера Oracle Database XE

Перед установкой сервера рекомендуется создать каталог (например, C:\ImportBase) и разместить в нем файл ImportBase.exe, расположенный на прилагаемом к книге компакт-диске.

Программа установки сервера Oracle Database XE создает одновременно компоненты клиента и сервера. Если вы планируете использовать Oracle Database XE как самодостаточный программный продукт, то вам необходимо установить лишь серверный компонент.

Возможна установка с помощью графического пользовательского интерфейса или фоновая установка, которая заключается в запуске инсталляционного пакета через командную строку, используя имеющиеся ответные файлы.

Здесь мы рассмотрим установку с помощью графического пользовательского интерфейса. Она осуществляется простым запуском (с помощью двойного щелчка) исполняемого файла установки (OracleXE.exe), после чего пользователь должен отвечать на вопросы графического интерфейса программы-установщика.

На первой форме установщика (рис. А2) надо нажать кнопку **Next.**

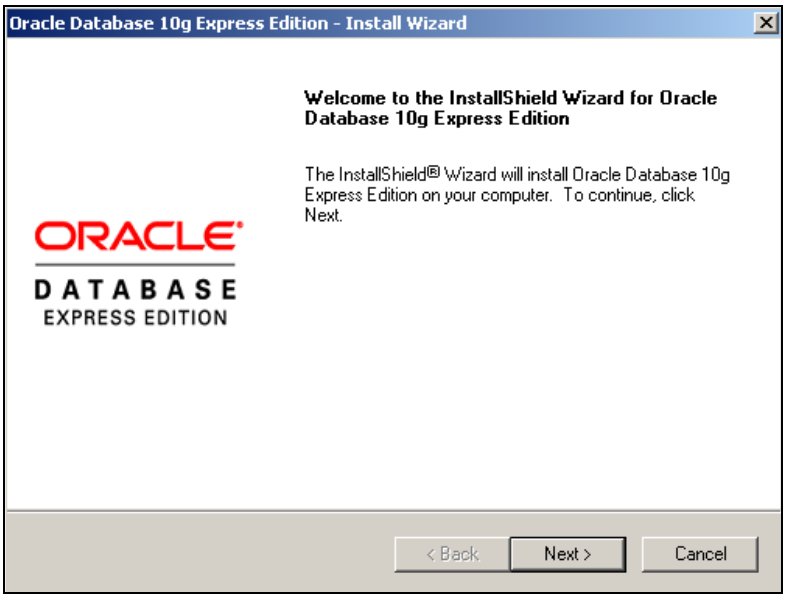


Рис. А2. Приветственное окно мастера установки Oracle Database 10g Express Edition

Прочитав условия лицензионного соглашения (рис. А3), надо выбрать пункт **I accept …** и нажать **Next**.

На экране **Choose Destination Location** (рис. А4) выберите путь по умолчанию для установки системы или с помощью команды **Browse** укажите другой путь установки. (Не выбирайте директорию, в наименовании которой существуют пробелы.). Затем нажмите **Next.**

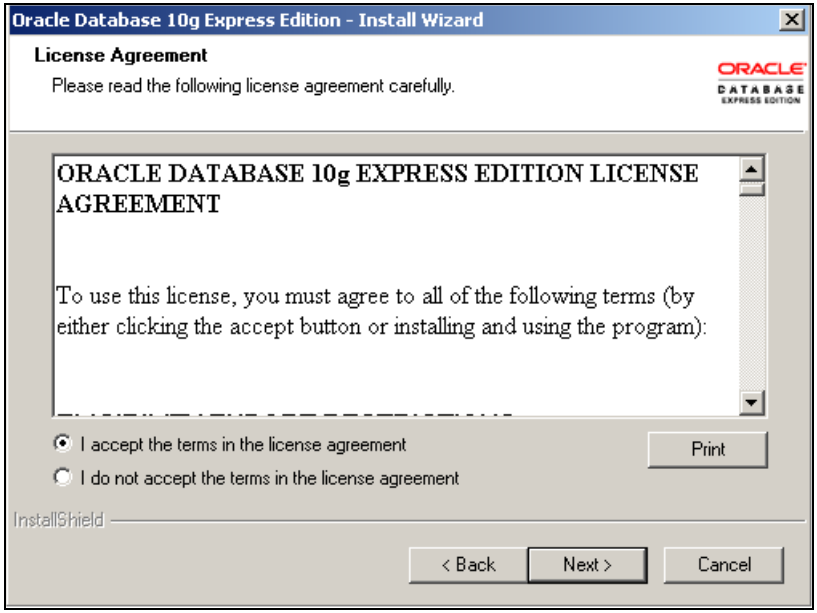


Рис. А3. Экран License Agreement

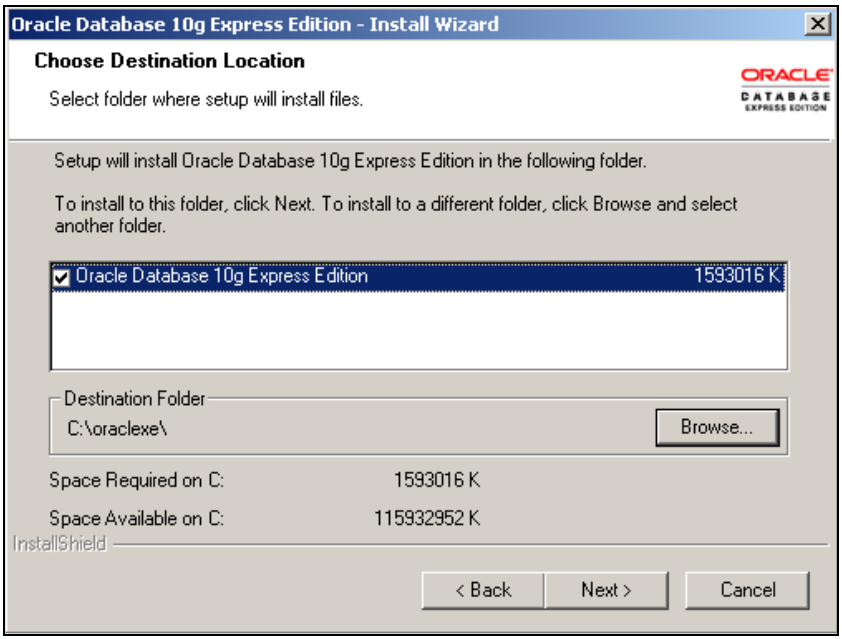


Рис. А4. Экран для выбора каталога для установки системы

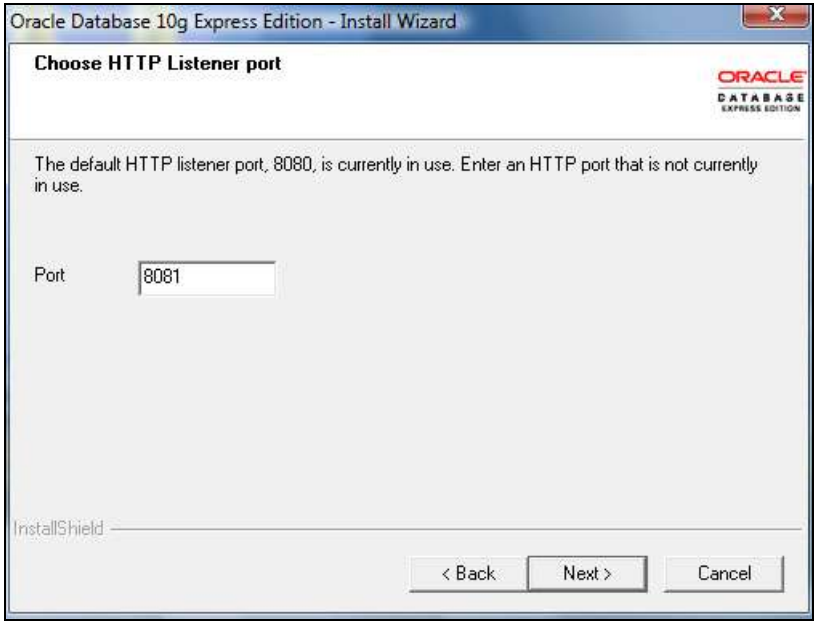


Рис. А5. Экран для ввода номеров портов

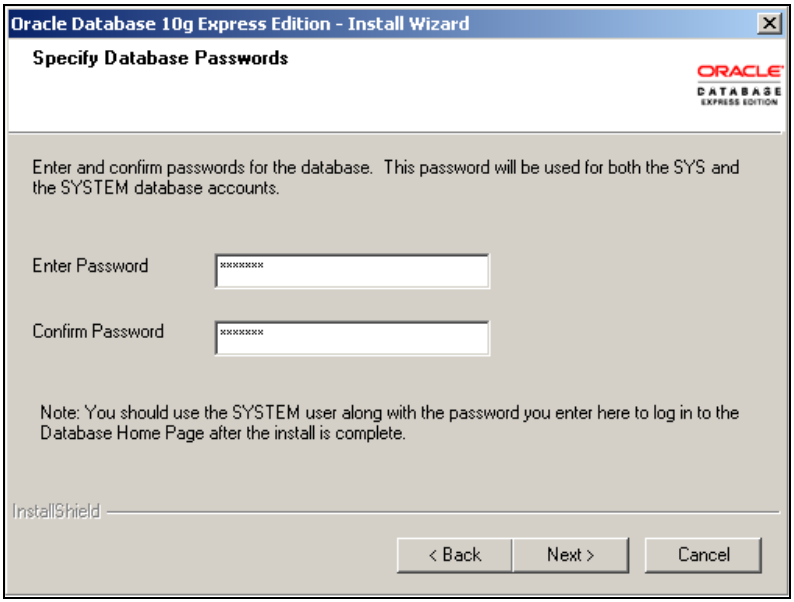


Рис. А6. Экран для ввода пароля учетных записей `SYS` и `SYSTEM`

Если мастер спросит вас о номерах портов (рис. А5), укажите нужный.

Этот экран появится только в том случае, если в момент установки заняты следующие порты (используемые Oracle по умолчанию):

- 1521: слушатель сети базы данных Oracle;
- 2030: службы Oracle для Microsoft Transaction Server;
- 8080: HTTP-порт графического пользовательского интерфейса Oracle Database XE.

Если данные порты заняты, то мастер попросит вас ввести доступный адрес порта, который будет использоваться в дальнейшем (в рассматриваемом случае был занят порт 8080).

На экране **Specify Database Passwords** (рис. А6) введите (Enter) и подтвердите (Confirm) пароли для учетных записей `SYS` и `SYSTEM`. (Авторы обычно вводят в подобных случаях пароль `manager`, рекомендованный Oracle еще для ранних версий системы.) Затем нажмите **Next**.

На экране **Summary** (рис. А7) просмотрите выбранные параметры установки и, если все в порядке, нажмите **Install**. В противном случае нажмите **Back** и измените параметры установки.

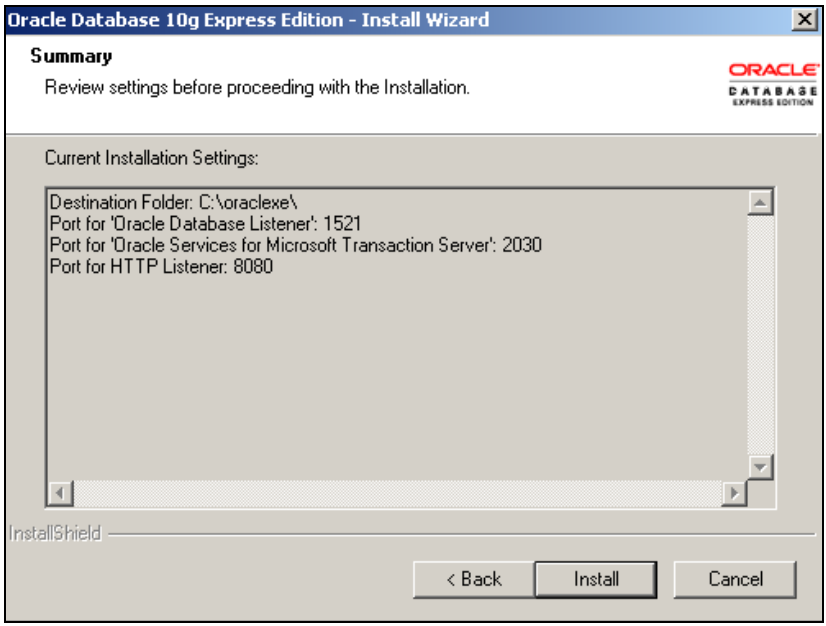


Рис. А7. Экран для проверки параметров установки

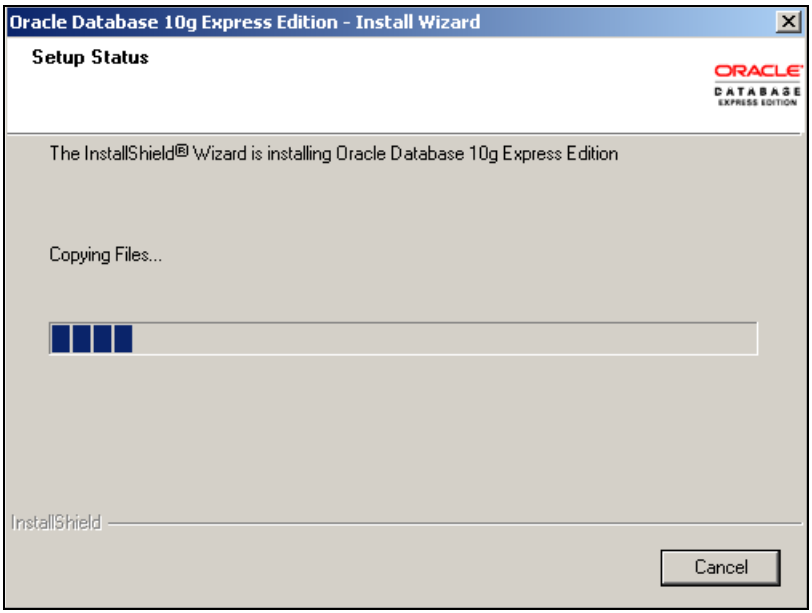


Рис. А8. Начало процесса установки

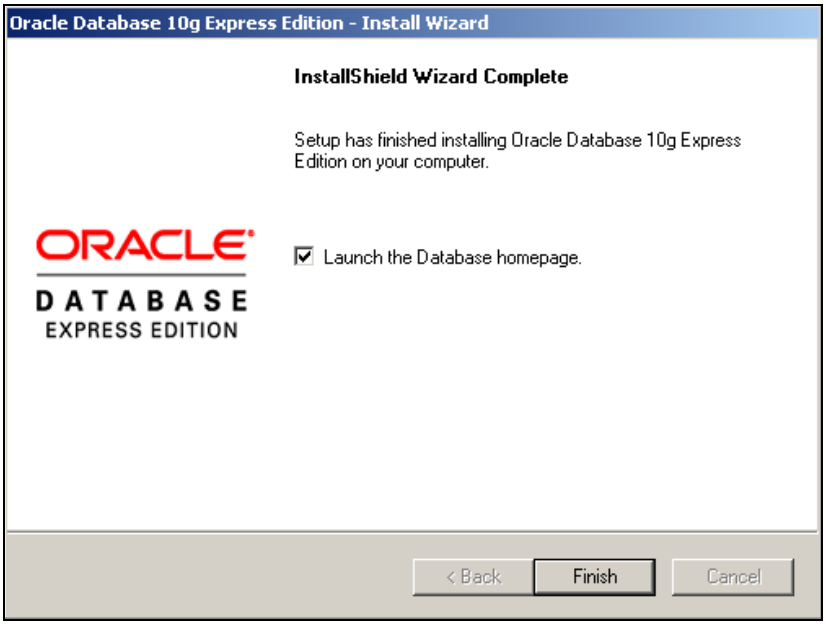


Рис. А9. Экран для завершения установки
и запуска (если необходимо) Oracle Database 10g Express

Начинается процесс установки (рис. А8).

После завершения установки появится экран, показанный на рис. А9. Для выполнения руссификации Oracle Database 10g Express (Western European) Edition, надо снять галочку **Launch the Database homepage** и нажать **Finish**.

А3. Русификация Oracle Database XE и создание баз данных "COOK" и "UCHEB"

Русификация Oracle Database 10g Express (Western European) Edition осуществляется с помощью скрипта (листинг А1), который должен выполняться сразу же после установки (пока в базе нет объектов пользователя и данных).

Листинг А1. Скрипт русификации Oracle Database 10g Express (Western European)

```
prompt patch started;
UPDATE SYS.PROPS$
SET VALUE$ = 'CL8MSWIN1251'
WHERE NAME = 'NLS_CHARACTERSET';
COMMIT;
SHUTDOWN IMMEDIATE;
STARTUP MOUNT;
ALTER SYSTEM ENABLE RESTRICTED SESSION;
ALTER DATABASE OPEN;
ALTER DATABASE XE CHARACTER SET CL8MSWIN1251;
SHUTDOWN IMMEDIATE;
STARTUP;
prompt patch completed;
```

Для запуска скрипта и установки баз данных, используется самораспаковывающийся архив ImportBase.exe (рис. А10). Этот архив предлагалось (в начале *разд. А2*) разместить в каталоге персонального компьютера, на который будет устанавливаться сервер Oracle Database XE.

После запуска ImportBase.exe надо указать место для извлечения и нажать **Извлечь**.

Когда установка завершится, то закроется окно консоли (так как производится загрузка достаточно большой базы данных "UCHEB", то импорт может продлиться несколько минут). Лог-файлы будут размещены в папке, выбранной для извлечения.

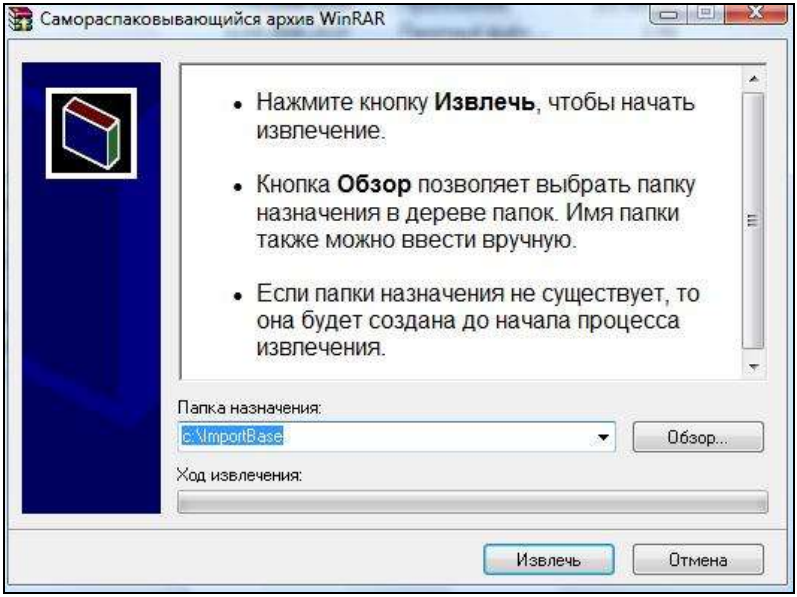


Рис. А10. Экран с самораспаковывающимся архивом

После запуска ImportBase.exe надо указать место для извлечения и нажать Извлечь.

Когда установка завершится, то закроется окно консоли (так как производится загрузка достаточно большой базы данных UCHEB, то импорт может продлиться несколько минут). Лог-файлы будут размещены в папке, выбранной для извлечения.

После завершения установки можно запустить Oracle SQL Developer (см. *разд. А4.2*).

> При написании *разд. А1* и *А2* мы воспользовались материалом, размещенным на сайте **http://www.oranet.ru/OraDoc10gXE/install.102/b25143/toc.htm**, где существуют подробные рекомендации, с которыми следует ознакомиться для расширения своих знаний.

А4. SqlDeveloper

А4.1. Введение

Блоки PL/SQL можно выполнять в различных средах (табл. А1), каждая из которых обладает собственными свойствами и возможностями.

Таблица А1. *Инструментальные среды для работы с SQL и PL/SQL*

Инструментальное средство	Производитель
SQL*Plus	Oracle Corporation
Rapid SQL	Embarcadero
DBPartner	Technologies
SQL Navigator	Compuware
TOADT	Quest Software
SQL-Programmer	Quest Software
PL/SQL Developer	BMC Software
SqlDeveloper	Oracle Corporation

Все эти инструментальные средства, кроме SQL*Plus и SqlDeveloper, — платные и, с нашей точки зрения, по своим возможностям ничуть не лучше (а иногда много хуже), чем новый продукт Oracle — SqlDeveloper.

SQL*Plus, пожалуй, самый простой из инструментов разработки программ PL/SQL. Он дает пользователям возможность вводить SQL-операторы и блоки PL/SQL в диалоговом режиме в ответ на приглашение (`SQL>`). Предложения направляются непосредственно СУБД, а результаты выводятся на экран. Эта среда функционирует в символьном режиме, и локальной системы поддержки PL/SQL в ней нет.

Обычно SQL*Plus поставляется вместе с сервером Oracle и является частью стандартной системы Oracle. Так как он разрабатывался для ранних версий Oracle и почти не видоизменялся с появлением новых версий, то единственным его достоинством, по сравнению с другими инструментальными средствами, было отсутствие цены. Однако после появления SqlDeveloper это достоинство исчезло.

SQL Developer — это бесплатный графический инструмент для разработки баз данных. С помощью SQL Developer можно просматривать объекты базы данных, запускать SQL-предложения, редактировать и отлаживать PL/SQL-программы. Вы также можете запустить любое количество предоставляемых отчетов, а также создавать и сохранять собственные. SQL Developer повышает производительность и упрощает вашу базу данных задач в области развития.

SQL Developer может подключаться к любой СУБД Oracle от версии 9.2.0.1 до Oracle 11 и может работать на Windows, Linux и Mac OSX.

SQL Developer интегрирован с Oracle Application Express (APEX) — бесплатной средой разработки Web-приложений на основе СУБД Oracle. С помощью APEX можно разрабатывать как небольшие приложения с дюжиной пользователей, так и масштабные приложения корпоративного уровня с тысячами пользователей.

APEX может использоваться с версией Oracle 9.2 и выше, а начиная с версии Oracle 11g среда APEX инсталлируется по умолчанию вместе с СУБД. В качестве Web-сервера, для отображения страниц, используется Apache или встроенный в СУБД Oracle Web-сервер — Embedded PL/SQL Gateway (EPG). EPG используется в Oracle Database Express Edition 10.2.0.1, т. е. в предлагаемой вам СУБД.

А4.2. Краткое руководство по установке и настройке

Для установки Oracle SQL Developer разархивируйте файл sqldeveloper-1.2.1.3213.zip на жесткий диск.

Перейдите в каталог, в котором находится разархивированный Oracle SQL Developer, и откройте файл sqldeveloper\bin\sqldeveloper.conf на редактирование. Добавьте в него следующую строчку:

```
AddVMOption -Duser.region=US
```

а затем сохраните исправленный файл под тем же именем.

Для установки Oracle SQL Developer запустите файл sqldeveloper.exe. Появится экран, показанный на рис. А11.

Если вы устанавливаете Oracle SQL Developer впервые, то нажмите **No**.

Для загрузки пользовательских настроек предыдущей версии Oracle SQL Developer нажмите **Yes**. Выберите путь к пользовательским настройкам (рис. А12) и нажмите **OK**.

Появится главный экран приложения, часть которого приведена на рис. А13. Установка завершена успешно.

Для начала работы с базой данных Oracle 10g XE необходимо создать новое соединение. Для этого выберите в главном меню приложения пункт **File** (рис. А14) и в нем пункт **New**.

В появившемся диалоговом окне (рис. А15) выберите пункт **Database Connection** и нажмите **OK**.

Откроется диалоговое окно **New / Select Database Connection** (рис. А16).

Рис. A11. Экран со страницей входа в Oracle SQL Developer

Рис. A12. Экран со страницей для выбора пути к пользовательским настройкам

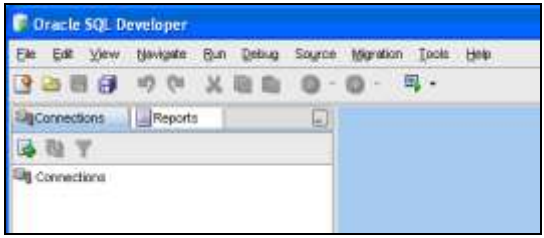


Рис. А13. Главный экран SQL Developer после его установки

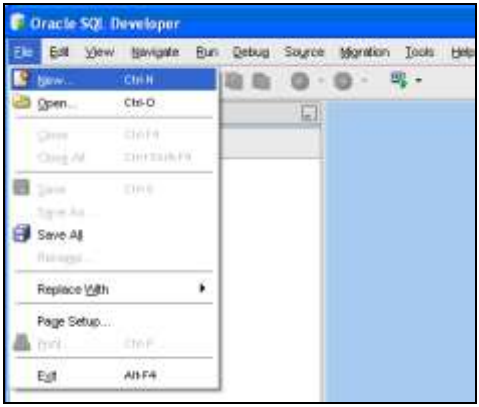


Рис. А14. Экран с подменю **File** SQL Developer

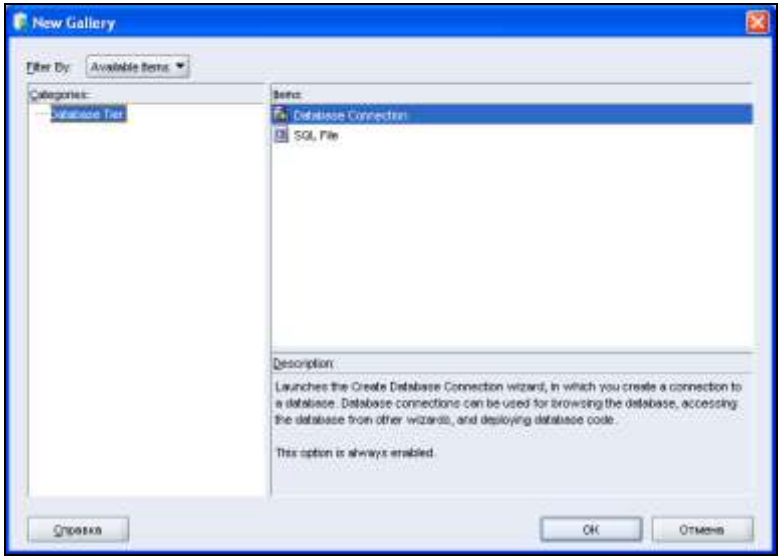


Рис. А15. Экран для перехода к созданию соединения с базой данных

Сначала подключим базу данных "COOK" (ее пароль `rjr` — "кок" русскими буквами на регистре EN). Для этого введем в поля **Connection Name** и **Username** имя пользователя (базы данных) — `COOK`, а в поле **Password** — `rjr`. Затем установим **Connection Type** — **TNS**, **Network Alias** — **XE** и нажмем **Connect**.

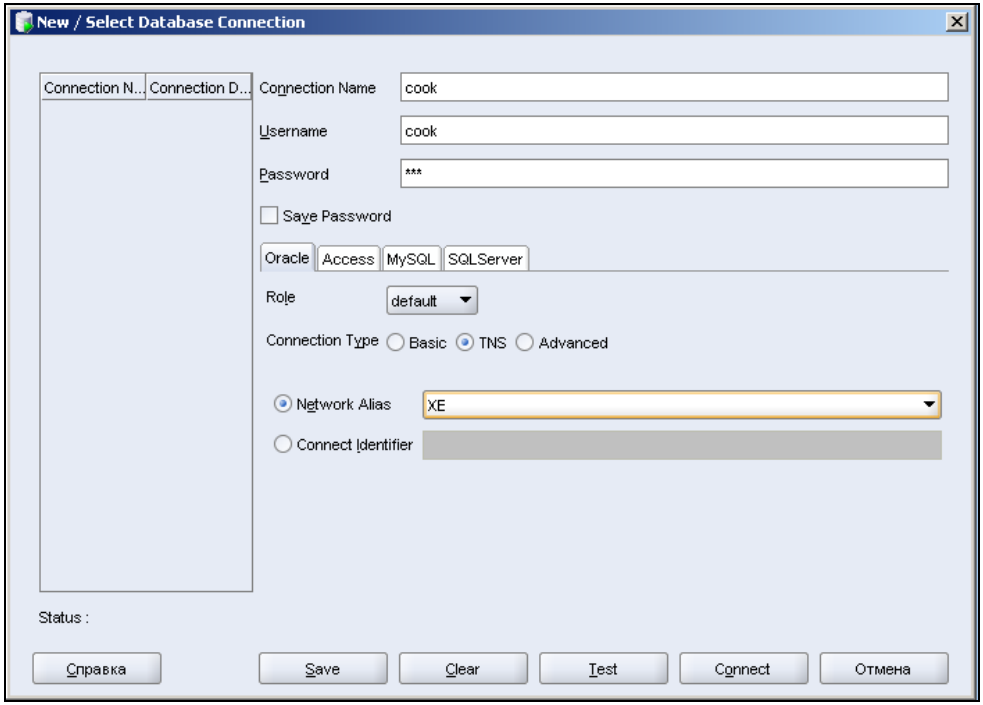


Рис. А16. Экран для перехода к созданию соединения с базой данных "COOK"

Аналогично произведем соединение с базой данных "UCHEB" (выполним действия, показанные на рис. А14, А15, заполним форму рис. А17 и нажмем **Connect**).

После этого выберем на экране SQL Developer базу данных "COOK", откроем список ее таблиц и выберем, например, таблицу `БЛЮДА` (рис. А18).

Далее можно перейти, например, на вкладку **Data** и вывести перечень блюд этой таблицы, показанный на рис. А19.

Мы не будем приводить здесь руководства пользователя SQL Developer. У него есть достаточно хороший Help на английском языке (см. главное меню, например, на рис. А13).

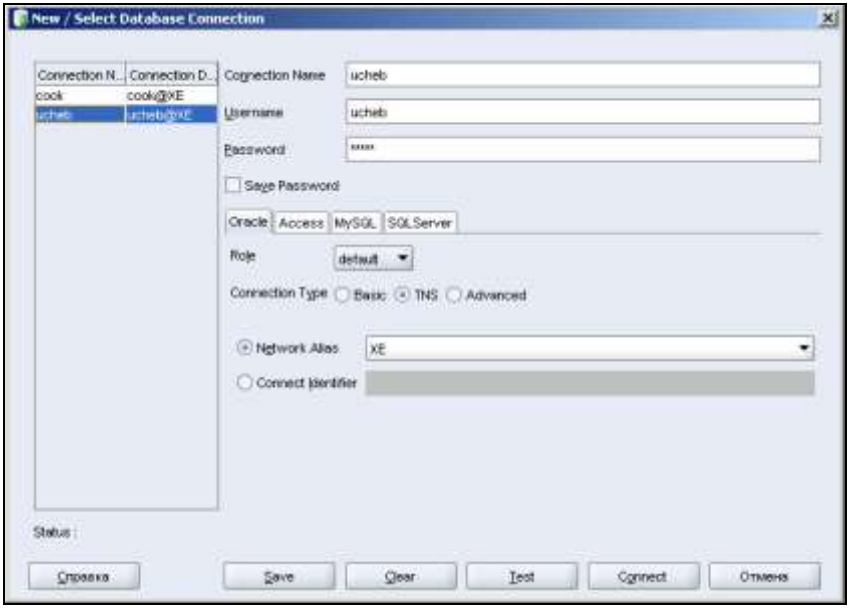


Рис. А17. Экран для перехода к созданию соединения с базой данных "UCHEB"

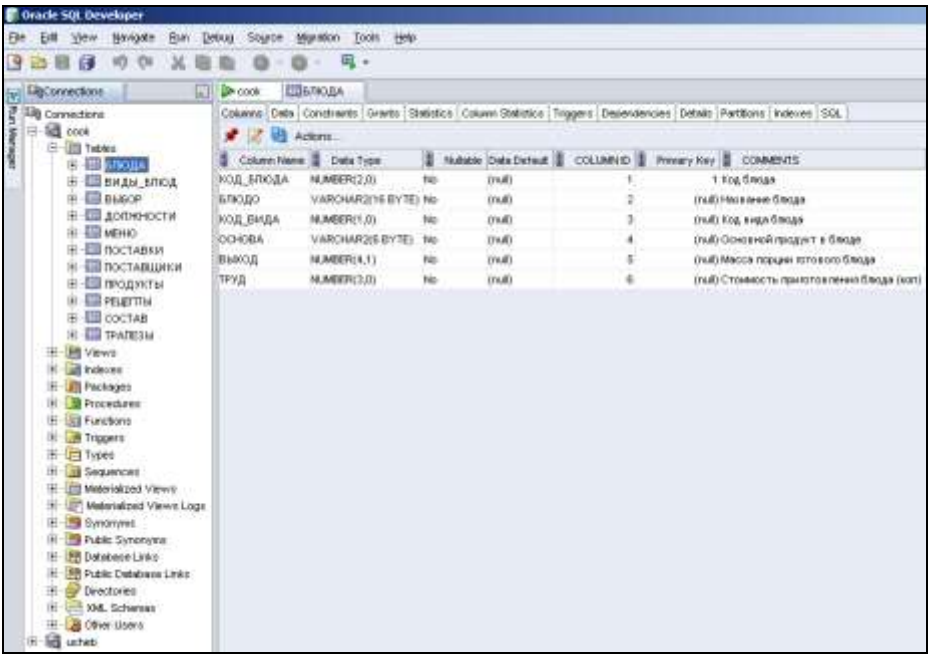


Рис. А18. Экран со структурой таблицы БЛЮДА базы данных COOK

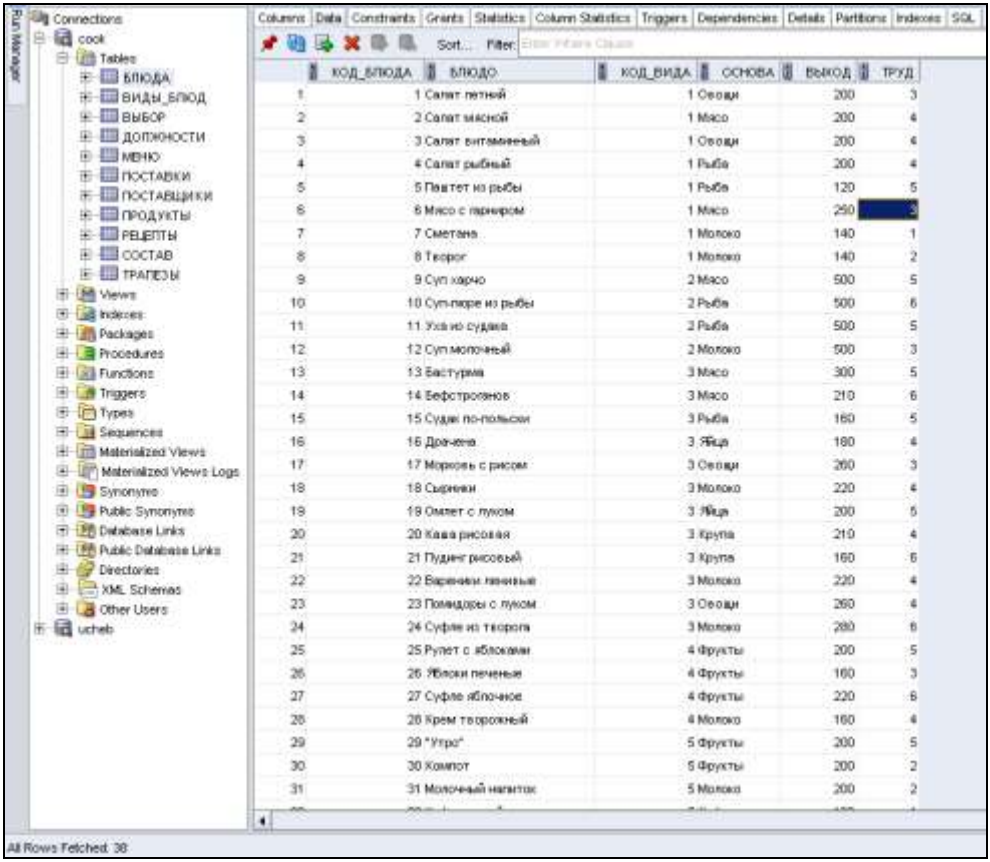


Рис. А19. Экран с данными таблицы БЛЮДА базы данных "COOK"

Приложение Б

Описание содержимого компакт-диска

Набор каталогов и файлов:

1. OracleXE:
   - Дистрибутив Oracle 10g XE для Windows, версия для Западной Европы (Oracle Database 10g Express Edition (Western European) — файл OracleXE.exe (161 457 Кбайт).
2. SqlDeveloper:
   - Oracle SQL Developer (sqldeveloper-1.2.1.3213.zip — 80 670 Кбайт);
   - Файл со сведениями по корректировке настроек Oracle SQL Developer ("В sqldeveloper.doc").
3. Документация по Oracle 10g Release 2 (10.2) (Doc10_2_htm.rar — 193 585 Кбайт).
4. Файл для русификации Oracle 10g XE и загрузки данных.

   Самораспаковывающийся архив ImportBase.exe, содержащий:
   - командный файл install.bat с последовательностью команд;
   - файл для русификации OracleXE (patch.sql);
   - файл для создания пользователя `COOK` и загрузки его данных (pansion.sql);
   - файл для создания пользователя `UCHEB` (before_import.sql);
   - файл для загрузки данных пользователя `UCHEB` (ucheb.dmp).
5. Файл с листингами из глав книги (для баз данных "COOK" и "UCHEB").

Предметный указатель