УДК 681.066

ББК 22.18

А с т а х о в а И. Ф., М е л ь н и к о в В. М., Т о л с т о б р о в А. П., Ф е р т и к о в В. В. **СУБД: язык SQL в примерах и задачах.** — М.: ФИЗМАТЛИТ, 2007. — 168 c. ISBN

Учебное пособие содержит подборку примеров, задач и упражнений различ- ной степени сложности для обеспечения практических занятий по изучению основ языка SQL в рамках учебного курса, посвященного информационным системам с базами данных по направлению подготовки и специальности «при- кладная математика и информатика».

Допущено Министерством образования и науки Российской Федерации в качестве учебного пособия для студентов высших учебных заведений, обуча- ющихся по направлению подготовки и специальности «прикладная математика и информатика».

c Астахова И. Ф., Мельников В. М., Толстобров А. П., Фертиков В. В., 2007

*§*

*§*c ФИЗМАТЛИТ, 2007

ОГЛАВЛЕНИЕ

Введение 7

Г л а в а 1. **Основные понятия и определения** 10

* 1. Основные понятия реляционных баз данных 10
  2. Отличие SQL от процедурных языков программирования 12
  3. Интерактивный и встроенный SQL 12
  4. Составные части SQL 13
  5. Типы данных 13
     1. Тип данных “строка символов” 13
     2. Числовые типы данных 14
     3. Дата и время 15
     4. Неопределенные или отсутствующие данные (**NULL**) 15
  6. Используемые термины и обозначения 16
  7. Учебная база данных 16

Г л а в а 2. **Выборка данных (оператор SELECT)** 20

* 1. Простейшие **SELECT**-запросы 20
  2. Операторы **IN**, **BETWEEN**, **LIKE**, **IS NULL** 25
  3. Преобразование вывода и встроенные функции 28
     1. Числовые, символьные и строковые константы 28
     2. Арифметические операции для преобразования числовых данных 29
     3. Символьная операция конкатенации строк 29
     4. Символьные функции преобразования букв различных слов

в строке 30

* + 1. Символьные строковые функции 30
    2. Функции работы с числами 33
    3. Функции преобразования значений 34
  1. Агрегирование и групповые функции 38
  2. Неопределенные значения (**NULL**) в агрегирующих функциях 41
     1. Влияние **NULL**-значений в функции **COUNT** 41
     2. Влияние **NULL**-значений в функции **AVG** 42
  3. Результат действия трехзначных условных операторов 42
  4. Упорядочение выходных полей (**ORDER BY**) 43
  5. Вложенные подзапросы 45

4 *Оглавление*

* 1. [Формирование связанных подзапросов 46](#_TOC_250032)
  2. [Связанные подзапросы в HAVING 49](#_TOC_250031)
  3. [Использование оператора EXISTS 50](#_TOC_250030)
  4. Операторы сравнения с множеством значений **IN**, **ANY**, **ALL** 52
  5. [Особенности применения операторов **ANY**, **ALL**, **EXISTS** при об- работке отсутствующих данных 55](#_TOC_250029)
  6. [Использование COUNT вместо EXISTS 57](#_TOC_250028)
  7. [Соединение таблиц. Оператор **JOIN** 58](#_TOC_250027)
     1. Операции соединения таблиц посредством ссылочной целостности 59
     2. Внешнее соединение таблиц 62
     3. Использование псевдонимов при соединении копий одной таблицы 65
  8. [Оператор объединения UNION 66](#_TOC_250026)
     1. Устранение дублирования в **UNION** 66
     2. Использование UNION с ORDER BY 68

Г л а в а 3. Манипулирование данными 71

* 1. [Операторы манипулирования данными 71](#_TOC_250025)
  2. [Использование подзапросов в **INSERT** 74](#_TOC_250024)
     1. Использование подзапросов, основанных на таблицах внеш-

них запросов 74

* + 1. Использование подзапросов с **DELETE** 75
    2. Использование подзапросов с **UPDATE** 76

Г л а в а 4. Создание объектов базы данных 78

* 1. [Создание таблиц базы данных 78](#_TOC_250023)
  2. [Использование индексации для быстрого доступа к данным 79](#_TOC_250022)
  3. [Изменение существующей таблицы 80](#_TOC_250021)
  4. [Удаление таблицы 80](#_TOC_250020)
  5. [Ограничения на множество допустимых значений данных 81](#_TOC_250019)
     1. Ограничение NOT NULL 82
     2. Уникальность как ограничение на столбец 83
     3. Уникальность как ограничение таблицы 83
     4. Присвоение имен ограничениям 84
     5. Ограничение первичных ключей 84
     6. Составные первичные ключи 85
     7. Проверка значений полей 85
     8. Проверка ограничивающих условий с использованием состав-

ных полей 86

* + 1. Установка значений по умолчанию 86
  1. [Поддержка целостности данных 88](#_TOC_250018)
     1. Внешние и родительские ключи 89
     2. Составные внешние ключи 89
     3. Смысл внешнего и родительского ключей 89
     4. Ограничение внешнего ключа (FOREIGN KEY) 90

Оглавление 5

* + 1. Внешний ключ как ограничение таблицы 90
    2. Внешний ключ как ограничение столбцов 91
    3. Поддержание ссылочной целостности и ограничения значе-

ний родительского ключа 93

* + 1. Использование первичного ключа в качестве уникального внешнего ключа 93
    2. Ограничения значений внешнего ключа 93
    3. Действие ограничений внешнего и родительского ключей

при использовании команд модификации 93

Г л а в а 5. Представления (VIEW) 97

* 1. [Представления — именованные запросы 97](#_TOC_250017)
  2. [Модификация представлений 98](#_TOC_250016)
  3. [Маскирующие представления 99](#_TOC_250015)
     1. Представления, маскирующие столбцы 99
     2. Операции модификации в представлениях, маскирующих столбцы 99
     3. Представления, маскирующие строки 99
     4. Операции модификации в представлениях, маскирующих строки 100
     5. Операции модификации в представлениях, маскирующих строки и столбцы 101
  4. [Агрегированные представления 102](#_TOC_250014)
  5. [Представления, основанные на нескольких таблицах 103](#_TOC_250013)
  6. [Представления и подзапросы 103](#_TOC_250012)
  7. [Удаление представлений 104](#_TOC_250011)
  8. [Изменение значений в представлениях 105](#_TOC_250010)
  9. [Примеры обновляемых и необновляемых представлений 106](#_TOC_250009)

Г л а в а 6. Определение прав доступа пользователей к данным 108

* 1. [Пользователи и привилегии 108](#_TOC_250008)
  2. [Стандартные привилегии 109](#_TOC_250007)
  3. [Команда GRANT 109](#_TOC_250006)
  4. [Использование аргументов ALL и PUBLIC 110](#_TOC_250005)
  5. [Отмена привилегий 111](#_TOC_250004)
  6. [Использование представлений для фильтрации привилегий 111](#_TOC_250003)
     1. Ограничение привилегии **SELECT** для определенных столбцов 112
     2. Ограничение привилегий для определенных строк 112
     3. Предоставление доступа только к извлеченным данным 113
     4. Использование представлений в качестве альтернативы огра- ничениям 113
  7. [Другие типы привилегий 114](#_TOC_250002)
  8. [Типичные привилегии системы 114](#_TOC_250001)
  9. [Создание и удаление пользователей 115](#_TOC_250000)

1. *Оглавление*



* 1. Создание синонимов (**SYNONYM**) 116
  2. Синонимы общего пользования **(PUBLIC)** 117

|  |  |  |
| --- | --- | --- |
| 6.12. Удаление синонимов. . . . . . . . . . . . . . . . . . . . . . . . . . . .. | .. . . | 117 |
| Г л а в а 7. **Управление транзакциями** . . .. . . . . . . . . . . . . . . . .. | .. . . | 118 |
| **Ответы к упражнениям** . . . . . . . . . . . . .. . . . . . . . . . . . . . . . .. | .. . . | 120 |

П р и л о ж е н и е. **Задачи по проектированию БД** 150

Предметный указатель 161

Введение

Информационные системы, использующие базы данных, в настоя- щее время представляют собой одну из важнейших областей современ- ных компьютерных технологий. С этой сферой связана большая часть современного рынка программных продуктов. Одной из общих тенден- ций в развитии таких систем являются процессы интеграции и стан- дартизации, затрагивающие структуры данных и способы их обработки и интерпретации, системное и прикладное программное обеспечение, средства разработки взаимодействия компонентов баз данных и т. п. Современные системы управления базами данных (СУБД) основаны на реляционной модели представления данных — в большой степени благодаря простоте и четкости ее концептуальных понятий и строгому математическому обоснованию.

Неотъемлемая и важная часть любой системы, включающей базы данных, — языковые средства, предоставляющие возможность досту- па к данным для получения необходимой информации и осуществ- ления необходимых действий над содержимым данных, определения их структур, способов использования и интерпретации. Язык SQL появился в 70-е годы ХХ века как одно из таких средств. Его прото- тип был разработан фирмой IBM и известен под названием SEQUEL (Structured English QUEry Language). SQL вобрал в себя достоинства реляционной модели, в частности, достоинства лежащего в ее основе математического аппарата реляционной алгебры и реляционного исчис- ления, используя при этом сравнительно небольшое число операторов и относительно простой синтаксис. Благодаря своим качествам язык SQL стал — вначале де-факто, а затем и официально — утвержденным в качестве стандарта языком работы с реляционными базами данных.

Учитывая место, занимаемое языком SQL в современных инфор- мационных технологиях, его знание необходимо любому специалисту, работающему в этой области. Поэтому его практическое освоение явля- ется неотъемлемой частью учебных курсов, направленных на изучение информационных систем с базами данных. В настоящее время такие курсы входят в учебные планы ряда университетских специальностей. Несомненно, что для получения студентами устойчивых навыков вла- дения языком SQL, соответствующий учебный курс, помимо теоретиче-

8 *Введение*



ского ознакомления с основами языка, должен обязательно содержать достаточно большой объем лабораторных занятий по его практиче- скому использованию. Предлагаемое учебное пособие направлено в первую очередь на методическое обеспечение именно такого рода заня- тий. В связи с этим в нем основное внимание уделяется подбору прак- тических примеров, задач и упражнений различной степени сложности по составлению SQL-запросов, позволяющих обеспечить проведение практических занятий по изучению языка в течение учебного семестра. При этом описание конструкций языка и тонкостей его применения в пособии приводится в минимальном объеме, необходимом для пони- мания студентами предлагаемых для решения упражнений и задач, и, возможно, с некоторым ущербом в строгости изложения материала.

Все приведенные в пособии задачи и упражнения составлены на примере использования одной общей базы данных, структура которой специально подобрана для обеспечения практической реализации и иллюстрации изучаемых конструкций языка. Для облегчения прак- тической организации занятий по изучению языка с использованием предлагаемых в пособии упражнений в компьютерном классе на ре- альной базе данных дополнением пособия служит файл, содержащий SQL-сценарий создания и наполнения данными учебной базы дан- ных, использованной в книге. Файл размещен на сайте издательства

[<http://www.fml.r](http://www.fml.ru/)u>. Первая часть сценария состоит из последова- тельности операторов DDL для создания таблиц, первичных и внешних ключей. Остальная часть образована командами DML, наполняющи- ми таблицы учебной информацией. Авторы надеются, что возможные случайные совпадения с реальными данными не вызовут раздражения читателей.

Сценарий ориентирован на сервер Oracle, и для его использования с другой СУБД, по-видимому, потребуется учет специфики. Следует обратить внимание на типы полей таблиц, заданные DDL-операторами сценария, а также на использованный формат представления строк и дат в командах DML. В самом трудном случае адаптация сценария, возможно, потребует использования какого-либо текстового процессора (например, для изменения длины строк в запросах INSERT). Кроме данной преодолимой трудности, авторы не предвидят препятствий к ис- пользованию сценария: достаточно квалификации пользователя СУБД, прочитавшего нашу книгу.

Возможность составления студентами запросов к реальной базе данных с достаточно большим объемом специально подобранных дан- ных позволяет существенно повысить продуктивность занятий, более наглядно увидеть особенности выполнения конкретных видов SQL- запросов, в частности, оценить реальное время их выполнения.

В пособии приведены ответы на большинство приведенных в нем задач, облегчающие преподавателю проверку результатов выполнения заданий и позволяющие использовать пособие для самостоятельной работы студентов. Примеры и задачи протестированы с использованием

*Введение* 9



СУБД Oracle и практически опробованы при проведении занятий в Воронежском госуниверситете на факультете компьютерных наук и факультете прикладной механики, математики и информатики.

В приложении приведены тексты дополнительных задач по проек- тированию баз данных. Эти задачи также могут использоваться при выполнении курсовых работ и самостоятельной работы студентов.

Авторы надеются, что пособие окажется полезным не только пре- подавателям и студентам, но и другим читателям, заинтересованным в получении начальных практических навыков использования языка SQL.

Авторы выражают искреннюю благодарность всем, кто помогал в создании этой книги, преподавателям ВГУ, использующим ее матери- алы при проведении занятий по языку SQL, за обсуждение книги и пожелания по ее содержанию. Особая благодарность Сергею Дмитри- евичу Кузнецову, профессору кафедры системного программирования факультета вычислительной математики и кибернетики МГУ, который не пожалел времени на внимательное прочтение рукописи и сделал большое число ценных замечаний, позволивших значительно улучшить эту книгу.

Авторы с благодарностью примут любые замечания, пожелания, исправления, которые будут способствовать улучшению качества по- собия, по адресу: 394693, Университетская пл., 1, Воронеж, Россия; электронный адрес: [tap@main.vsu.ru.](mailto:tap@main.vsu.ru)

Г л а в а 1

**ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ**

**1.1. Основные понятия реляционных баз данных**

Основой современных систем, использующих базы данных, являет- ся *реляционная* модель данных. В этой модели данные, представляю- щие информацию о предметной области, организованы в виде двумер- ных таблиц, называемых *отношениями*. На рис. 1 приведен пример такой таблицы-отношения и поясняются основные термины реляцион- ной модели.

,

* 1. *Основные понятия реляционных баз данных* 11



Строки таблицы-отношения называются *кортежами*, или *за- писями*. Столбцы называются *атрибутами*. Термины: атрибут, столбец, колонка, поле — обычно используются как синонимы. Каждый атрибут имеет наименование (имя), которое должно быть уникальным в конкретной таблице-отношении, однако в разных таблицах имена атрибутов могут совпадать.

*•*

Количество кортежей в таблице-отношении называется *карди- нальным числом* отношения, а количество атрибутов называется *степенью* отношения.

*•*

*Ключ*, или *первичный ключ* отношения — это уникальный иден- тификатор строк (кортежей), т. е. такой атрибут (набор атрибу- тов), для которого в любой момент времени в отношении не существует строк с одинаковыми значениями этого атрибута (на- бора атрибутов). На рис. 1 таблицы ячейка с именем ключевого атрибута имеет нижнюю границу в виде двойной черты.

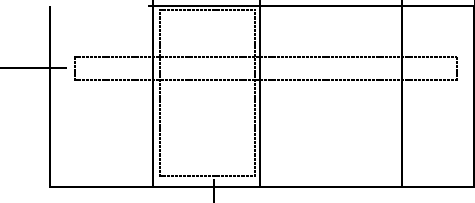
*•*

*Домен* отношения — это совокупность значений, из которых мо- гут выбираться значения конкретного атрибута, т. е. конкретный набор имеющихся в таблице значений атрибута в любой момент времени должен быть подмножеством множества значений доме- на, на котором определен этот атрибут. В общем случае на одном и том же домене могут быть определены значения разных атри- бутов. Важным является то, что домены вводят ограничения на операции сравнения значений различных атрибутов. Эти ограни-

*•*

Заголовок отношения

\



-

Ключ

v

Наименование атрибута

v

чения состоят в том, что корректным образом можно сравнивать

между собой только значения атрибутов, определенных на одном и том же домене.

Кортеж

\

|  |  |  |  |
| --- | --- | --- | --- |
| 0043 | Иванов | Физический | 1 |
|  | | | |
| 5162 | Сидоров | Физический | 2 |
| 0007 | Орлов | Химический | 4 |
| 0634 | Смирнов | Физический | 3 |

Код\_студ Имя\_студ Факультет Курс

Отношения реляционной базы данных обладают следующими свой- ствами:

- 2004 Петров Химический 2

в отношениях не должно быть кортежей-дубликатов; кортежи отношений неупорядочены;

атрибуты отношений также неупорядочены.

*•*

*•*

*•*

Из этих свойств отношения вытекают следующие важные след-

//´

/

/

0228

1735

Попов Исторический 4

Кузнецов Физический 1

\

z

ˆ

\



ствия.

Из уникальности кортежей следует, что в отношении *всегда*

*•*

имеется атрибут или набор атрибутов, позволяющий *идентифи-*

Отношение

\ Атрибут

*цировать* кортеж; другими словами, в отношении ***всегда*** есть

Значение атрибута

Рис. 1

*Отношение* — это таблица, подобная приведенной на рис. 1, и состоящая из строк и столбцов. Верхняя строка таблицы-отноше- ния называется *заголовком отношения*. Термины *отношение* и *таблица* обычно употребляются как синонимы, однако в языке SQL используется термин *таблица*.

*•*

первичный ключ.

Из неупорядоченности кортежей следует, во-первых, что в отно- шении не существует другого способа адресации кортежей, кроме адресации *по ключу*; во-вторых, что в отношении не существует таких понятий как первый кортеж, последний, предыдущий, сле- дующий и т. п.

*•*

Из неупорядоченности атрибутов следует, что единственным спо- собом их адресации в запросах является использование наимено- вания атрибута.

*•*

Относительно свойства реляционного отношения, касающегося отсут- ствия кортежей-дубликатов, следует сделать важное замечание. В этом пункте SQL не полностью соответствует реляционной модели. А имен- но, в отношениях, являющихся результатами запросов, SQL *допускает* наличие одинаковых строк. Для их устранения в запросе используется ключевое слово **DISTINCT** (см. ниже).

Информация в реляционных базах данных, как правило, хранится не в одной таблице-отношении, а в нескольких. При создании несколь- ких таблиц взаимосвязанной информации появляется возможность вы- полнения более сложных операций с данными, т. е. более сложной обработки данных. Для работы со связанными данными из нескольких таблиц важным является понятие так называемых *внешних ключей*.

*Внешним ключом* таблицы называется атрибут или набор атри- бутов этой таблицы, каждое значение которых в текущем состоянии таблицы всегда совпадает со значением атрибутов, являющихся клю- чом, в другой таблице. Внешние ключи используются для связывания значений атрибутов из разных таблиц. С помощью внешних ключей обеспечивается так называемая *ссылочная целостность* базы данных, т. е. согласованность данных, описывающих одни и те же объекты, но хранящихся в разных таблицах.

* 1. Отличие SQL от процедурных языков программирования

Язык SQL относится к классу непроцедурных языков программи- рования. В отличие от универсальных процедурных языков, которые также могут быть использованы для работы с базами данных, язык SQL ориентирован не на *записи*, а на *множества*.

Это означает следующее. В качестве входной информации для формулируемого на языке SQL запроса к базе данных используется *множество кортежей-записей* одной или нескольких таблиц-отноше- ний. В результате выполнения запроса также образуется *множество кортежей* результирующей таблицы-отношения. Другими словами, в SQL результатом любой операции над отношениями также является отношение. Запрос SQL задает не процедуру, т. е. последовательность действий, необходимых для получения результата, а условия, которым должны удовлетворять кортежи результирующего отношения, сформу- лированные в терминах входного отношения (входных отношений).

* 1. Интерактивный и встроенный SQL

Существуют и используются две формы языка SQL: *интерактив- ный SQL* и *встроенный SQL*.

*Интерактивный SQL* используется для непосредственного ввода SQL-запросов пользователем и получения результата в интерактивном режиме.

*Встроенный SQL* состоит из команд SQL, встроенных внутрь про- грамм, которые обычно написаны на некотором другом языке (Паскаль, С, С++ и др.). Это делает программы, написанные на таких языках, более мощными, гибкими и эффективными, обеспечивая их приме- нение для работы с данными, хранящимися в реляционных базах. При этом, однако, требуются дополнительные средства обеспечения интерфейса SQL с языком, в который он встраивается.

Данная книга посвящена интерактивному SQL, поэтому в ней не обсуждаются вопросы построения интерфейсов, позволяющих связать SQL с другими языками программирования.

* 1. Составные части SQL

И интерактивный, и встроенный SQL подразделяются на следую- щие составные части.

*Язык Определения Данных* — DDL (Data Definition Language): дает возможность создания, изменения и удаления различных объектов базы данных (таблиц, индексов, пользователей, привилегий и т. п.).

К числу дополнительных функций DDL могут быть отнесены сред- ства определения ограничений целостности данных, определения по- рядка структур хранения данных, описания элементов физического уровня хранения данных.

*Язык Обработки Данных* — DML (Data Manipulation Language): предоставляет возможность выборки информации из базы данных и ее преобразования.

Тем не менее это не два различных языка, а компоненты единого SQL.

* 1. Типы данных

В языке SQL имеются средства, позволяющие для каждого атрибута указывать тип данных, которому должны соответствовать все значения этого атрибута.

Следует отметить, что определение типов данных является той частью, в которой коммерческие реализации языка не полностью согла- суются с требованиями официального стандарта SQL. Это объясняется, в частности, желанием сделать SQL совместимым с другими языками программирования.

* + 1. **Тип данных “строка символов”.** Тип данных **CHARACTER** или **CHAR** представляет символьные строки фиксированной длины. Его синтаксис имеет вид:

**CHARACTER**[(*<длина>*)] или

**CHAR**[(*<длина>*)].

Текстовые значения поля таблицы, для которого определен тип

**CHAR**, имеют *фиксированную* длину, которая определяется параметром

*<длина>*. Этот параметр может принимать значения от 1 до 255, т. е. строка может содержать до 255 символов. Если во вводимой в поле текстовой константе фактическое число символов меньше числа, определенного параметром *<длина>*, то эта константа автоматически дополняется справа пробелами до заданного числа символов. Квадрат- ные скобки указывают на то, что значение параметра *<длина>* может не указываться явно. В этом случае длина строки полагается равной одному символу.

Тип данных для строк переменной длины может обозначать- ся ключевыми словами **VARCHAR**, **CHARACTER VARYING** или **CHAR VARYING**. Он описывает текстовую строку, которая может иметь *про- извольную* длину до определенного конкретной реализацией SQL мак- симума (в Oracle до 2000 символов). В отличие от типа **CHAR**, в этом случае при вводе текстовой константы, фактическая длина которой меньше заданной, ее дополнение пробелами до заданного максималь- ного значения не производится.

Константы, имеющие тип **CHARACTER** или **VARCHAR**, в выражениях SQL заключаются в одиночные кавычки, например, ‘*<текст>*’.

Следующие предложения эквивалентны:

**VARCHAR**[(*<длина>*)], **CHAR VARYING**[(*<длина>*)], **CHARACTER VARYING**[(*<длина>*)].

Если длина строки не указана явно, она полагается равной одному символу во всех случаях.

По сравнению с типом **CHAR** тип данных **VARCHAR** позволяет более экономно использовать память, выделяемую для хранения текстовых значений, и оказывается более удобным при выполнении операций, связанных со сравнением текстовых констант.

* + 1. **Числовые типы данных.** Стандартными числовыми типа- ми данных SQL являются:

**INTEGER** — используется для представления целых чисел в диа- пазоне от 231 до +231;

*−*

*•*

**SMALLINT** — используется для представления целых чисел в диапазоне меньшем, чем для **INTEGER**, а именно от 215 до +215; **DECIMAL**(*<точность>*[, *<масштаб>*]) — десятичное число с фиксированной точкой; точность указывает, сколько значащих цифр имеет число. Масштаб указывает максимальное число цифр справа от точки;

*•*

*−*

*•*

**NUMERIC**(*<точность>*[, *<масштаб>*]) — десятичное число с фиксированной точкой, такое же, как и **DECIMAL**; **FLOAT**[(*<точность>*)] — число с плавающей точкой и указанной минимальной точностью;

*•*

*•*

**REAL** — такое же число, как и **FLOAT**, за исключением того, что точность устанавливается по умолчанию в зависимости от конкретной реализации SQL**.**

*•*

**DOUBLE PRECISION** — такое же число, как и **REAL**, но точность в два раза превышает точность для **REAL**.

СУБД Oracle использует дополнительно тип данных **NUMBER** для представления всех числовых данных: целых, с фиксированной или плавающей точкой. Его синтаксис:

*•*

**NUMBER**[(*<точность>*[, *<масштаб>*])] .

Если значение параметра *<точность>* не указано явно, оно по- лагается равным 38. Значение параметра *<масштаб>* по умолчанию предполагается равным 0. Значение параметра *<точность>* может изменяться от 1 до 38; значение параметра *<масштаб>* может изме- няться от 84 до 128. Использование отрицательных значений мас- штаба означает сдвиг десятичной точки в сторону старших разрядов. Например, определение **NUMBER**(7, –3) означает округление до тысяч.

*−*

Типы **DECIMAL** и **NUMERIC** полностью эквивалентны типу **NUMBER**. Синтаксис:

**DECIMAL**[(*<точность>*[, *<масштаб>*])], **DEC**[(*<точность>*[, *<масштаб>*])], **NUMERIC**[(*<точность>*[, *<масштаб>*])].

Напоминаем, что квадратные скобки указывают на необязатель- ность заключенных в них параметров.

* + 1. **Дата и время.** Представление дат и времени в SQL зависит от конкретной СУБД. В Oracle тип данных **DATE** используется для представления даты и времени. Наличие типа данных для хранения да- ты позволяет поддерживать специальную арифметику дат. Добавление к переменной типа **DATE** целого числа означает увеличение даты на соответствующее число дней, а вычитание соответствует определению более ранней даты.

Константы типа **DATE** записываются в зависимости от форма- та, принятого в конкретной системе. Например, ‘03.05.1999’ или ‘12/06/1989’, или ‘03-nov-1999’, или ‘03-apr-99’.

* + 1. **Неопределенные или отсутствующие данные (NULL).** Для обозначения отсутствующих, пропущенных или неизвестных зна- чений атрибута в SQL используется ключевое слово **NULL**. Довольно часто можно встретить словосочетание “*атрибут имеет значение* **NULL**”. Строго говоря, **NULL** не является значением в обычном пони- мании, а используется именно для обозначения того факта, что дей- ствительное значение атрибута на самом деле по каким-либо причинам отсутствует. Это приводит к ряду особенностей, что следует учитывать при использовании значений атрибутов, которые могут находиться в состоянии **NULL**.

В агрегирующих функциях, позволяющих получать сводную ин- формацию по множеству значений атрибута, например, суммарное или среднее значение, для обеспечения точности и однозначности

*•*

толкования результатов отсутствующие или **NULL**-значения атри- бутов игнорируются.

* ки **истина/ложь** до трехзначной логики **истина/ложь/неиз-**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Условные операторы расширяются от булевой двузначной логи- 1 | | Иванов | Иван | 150 | 1 | Орел | 3/12/1988 | 10 |
| **вестно**. 3 | | Петров | Петр | 200 | 3 | Курск | 11/12/1986 | 10 |
| Все операторы возвращают пустое значение (**NULL**), если значе- 6 | | Сидоров | Вадим | 150 | 4 | Москва | 7/06/1985 | 22 |
| ние любого из операндов отсутствует (имеет “значение **NULL**”).  Для проверки на пустое значение следует использовать операто- 10 | | Кузнецов | Борис | 0 | 2 | Брянск | 8/12/1987 | 10 |
| ры **IS NULL** и **IS NOT NULL** (использование для этого опера- 12 | | Зайцева | Ольга | 250 | 2 | Липецк 21/05/1987 | | 10 |
| тора сравнения “=” является ошибкой). 265  Функции преобразования типов, имеющие **NULL** в качестве аргу- | | Павлов | Андрей | 0 | 3 | Воронеж 5/11/1985 | | 10 |
| мента, возвращают пустое значение (**NULL)**. | 32 | Котов | Павел | 150 | 5 | Белгород **NULL** | | 14 |
|  | 654 | Лукин | Артем | 200 | 3 | Воронеж 11/12/1987 | | 10 |

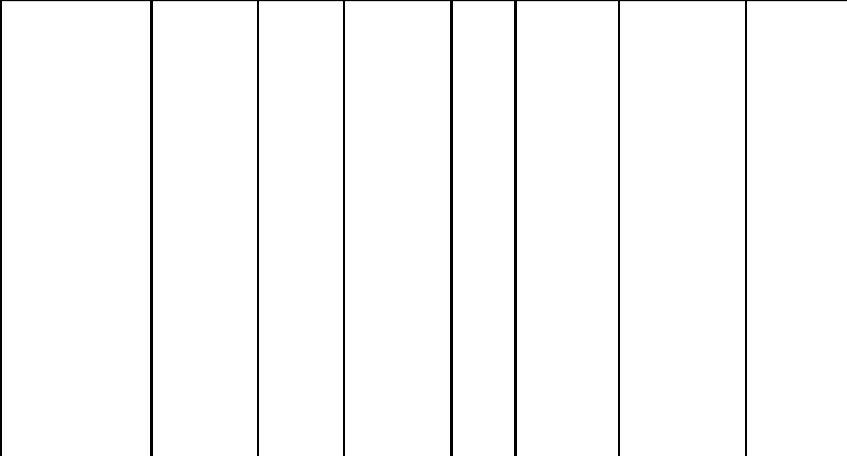
*•*

*•*

*•*

Т а б л и ц а 1.1. STUDENT (Студент)

STUDENT\_ID SURNAME NAME STIPEND KURS CITY BIRTHDAY UNIV\_ID



* 1. Используемые термины и обозначения

*Ключевые слова* — это используемые в выражениях SQL слова, имеющие специальное назначение (например, они могут обозначать конкретные команды SQL). Ключевые слова нельзя использовать для других целей, к примеру, в качестве имен объектов базы данных. В книге они выделяются шрифтом: **КЛЮЧЕВОЕСЛОВО**.

*Команды*, или *предложения*, являются инструкциями, с помощью которых SQL обращается к базе данных. Команды состоят из несколь- ких (одной или более) логических частей, называемых предложениями. Предложения начинаются ключевым словом и состоят из ключевых слов и аргументов.

Объекты базы данных, имеющие имена (таблицы, атрибуты и др.), в книге также выделяются особым образом: ТАБЛИЦА1, АТРИБУТ\_2.

В описании синтаксиса команд SQL оператор определения “**::=**” разделяет определяемый элемент (слева от оператора) и собственно его определение (справа от оператора); квадратные скобки “[ ]” указыва- ют *необязательный* элемент синтаксической конструкции; многоточие “*...*” указывает, что выражение, предшествующее ему, может повто- ряться любое число раз; фигурные скобки “{ }” объединяют последова- тельность элементов в *логическую группу*, один из элементов которой должно быть обязательно использован; вертикальная черта “ ” указы- вает, что часть определения, следующая за этим символом, является одним из возможных вариантов; в угловые скобки “*< >*” заключаются элементы, которые объясняются по мере того, как вводятся.

*|*

* 1. Учебная база данных

В приводимых в пособии примерах построения SQL-запросов и контрольных упражнениях используется база данных, состоящая из следующих таблиц.

276 Петров Антон 200 4 **NULL** 5/08/1987 22

55 Белкин Вадим 250 5 Воронеж 20/01/1986 10

. . . . . . . . . . . . . . . . . . . . . . . .

STUDENT\_ID — числовой код, идентифицирующий студента (идентификатор студента),

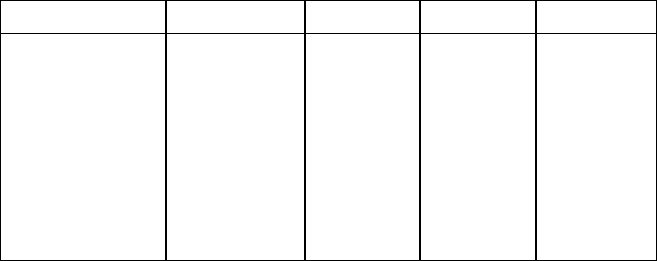
SURNAME — фамилия студента,

NAME — имя студента,

STIPEND — стипендия, которую получает студент,

KURS — курс, на котором учится студент, CITY — город, в котором живет студент, BIRTHDAY — дата рождения студента,

UNIV\_ID — идентификатор университета, в котором учится студент.

Т а б л и ц а 1.2. LECTURER (Преподаватель)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LECTURER\_ID | SURNAME | NAME | CITY | UNIV\_ID |
| 24 | Колесников | Борис | Воронеж | 10 |
| 46 | Никонов | Иван | Воронеж | 10 |
| 74 | Лагутин | Павел | Москва | 22 |
| 108 | Струков | Николай | Москва | 22 |
| 276 | Николаев | Виктор | Воронеж | 10 |
| 328 | Сорокин | Андрей | Орел | 10 |
| . . . | . . . | . . . | . . . | . . . |

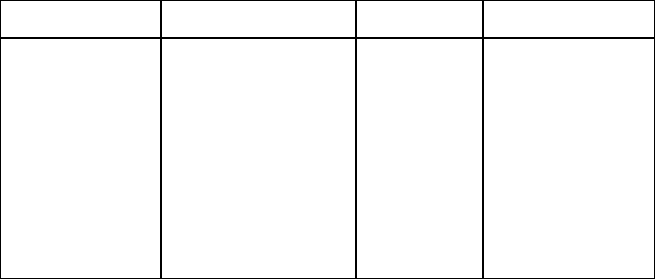
LECTURER\_ID — идентификатор преподавателя,

SURNAME — фамилия преподавателя,

NAME — имя преподавателя,

CITY — город, в котором живет преподаватель,

UNIV\_ID — идентификатор университета, в котором работает преподаватель.

Т а б л и ц а 1.3. SUBJECT (Предмет обучения)

|  |  |  |  |
| --- | --- | --- | --- |
| SUBJ\_ID | SUBJ\_NAME | HOUR | SEMESTER |
| 10 | Информатика | 56 | 1 |
| 22 | Физика | 34 | 1 |
| 43 | Математика | 56 | 2 |
| 56 | История | 34 | 4 |
| 94 | Английский | 56 | 3 |
| 73 | Физкультура | 34 | 5 |
| . . . | . . . | . . . | . . . |

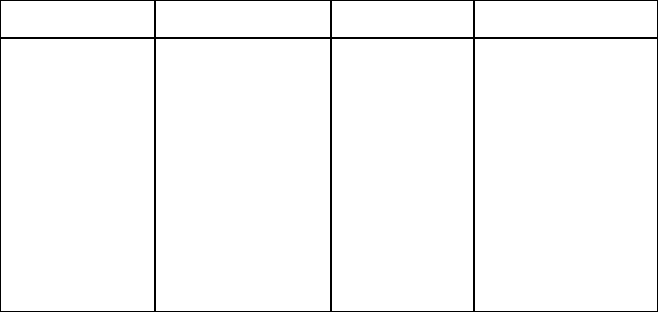
SUBJ\_ID — идентификатор предмета обучения,

SUBJ\_NAME — наименование предмета обучения,

HOUR — количество часов, отводимых на изучение предмета,

SEMESTER — семестр, в котором изучается данный предмет.

Т а б л и ц а 1.4. UNIVERSITY (Университет)



UNIV\_ID

UNIV\_NAME

RATING

CITY

18

. . .

ВГМА

. . .

327

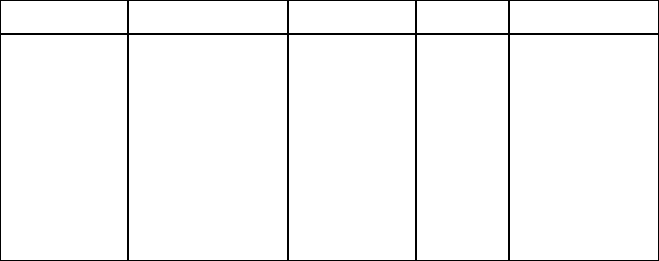
. . .

Воронеж

. . .

Т а б л и ц а 1.5. EXAM\_MARKS (Экзаменационные оценки)

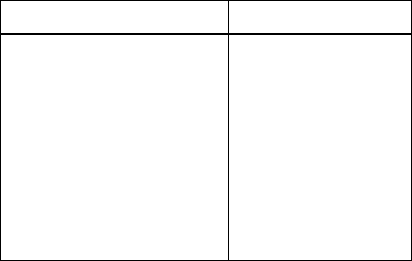
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| EXAM\_ID | STUDENT\_ID | SUBJ\_ID | MARK | EXAM\_DATE |
| 145 | 12 | 10 | 5 | 12/01/2006 |
| 34 | 32 | 10 | 4 | 23/01/2006 |
| 75 | 55 | 10 | 5 | 05/01/2006 |
| 238 | 12 | 22 | 3 | 17/06/2005 |
| 639 | 55 | 22 | **NULL** | 22/06/2005 |
| 43 | 6 | 22 | 4 | 18/01/2006 |
| . . . | . . . | . . . | . . . | . . . |

EXAM\_ID — идентификатор экзамена, STUDENT\_ID — идентификатор студента, SUBJ\_ID — идентификатор предмета обучения, MARK — экзаменационная оценка,

EXAM\_DATE — дата экзамена.

Т а б л и ц а 1.6. SUBJ\_LECT

(Учебные дисциплины преподавателей)



LECTURER\_ID

24

46

74

SUBJ\_ID

24

46

74

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | 108 | 108 |
| 22 | МГУ | 610 | Москва | 276 | 276 |
| 10 | ВГУ | 296 | Воронеж | 328 | 328 |
| 11 | НГУ | 345 | Новосибирск | . . . | . . . |
| 32 | РГУ | 421 | Ростов | LECTURER\_ID — идентификатор преподавателя, |  |
| 14 | БГУ | 326 | Белгород SUBJ\_ID — идентификатор предмета обучения. | | |
| 15 | ТГУ | 373 | Томск **ВОПРОСЫ** | | |

UNIV\_ID — идентификатор университета, UNIV\_NAME — название университета, RATING — рейтинг университета,

CITY — город, в котором расположен университет.

1. Какие поля приведенных таблиц являются первичными ключами?
2. Какие данные хранятся в столбце 2 в таблице “Предмет обуче- ния”?
3. Как по-другому называется строка? столбец?
4. Почему мы не можем запросить для просмотра первые пять строк?

Г л а в а 2

**ВЫБОРКА ДАННЫХ (ОПЕРАТОР SELECT)**

**2.1. Простейшие SELECT-запросы**

Оператор **SELECT** (ВЫБРАТЬ) языка SQL является самым важным и наиболее часто используемым оператором. Он предназначен для *выборки* информации из таблиц базы данных. Упрощенный синтаксис оператора **SELECT** выглядит следующим образом.

**SELECT [DISTINCT]***<список выражений над атрибутами*

*и константами>*

**FROM** *<список таблиц>*

**[WHERE** *<условие выборки>***] [GROUP BY** *<список атрибутов>***] [HAVING** *<условие>***]**

**[UNION** *<выражение с оператором* **SELECT***>***] [ORDER BY** *<список атрибутов>***]**;

В квадратных скобках указаны элементы, которые могут отсутство- вать в запросе.

Ключевое слово **SELECT** сообщает базе данных, что данное пред- ложение является запросом на *выборку* информации. После слова **SELECT** через запятую перечисляются *наименования полей* (список атрибутов), содержимое которых запрашивается.

Обязательным ключевым словом в предложении-запросе **SELECT** является слово **FROM** (ИЗ). За ключевым словом **FROM** указывается список разделенных запятыми имен таблиц, из которых извлекается информация.

Например,

**SELECT** NAME, SURNAME

**FROM** STUDENT;

Любой SQL-запрос должен заканчиваться символом “;” (*точка с запятой*).

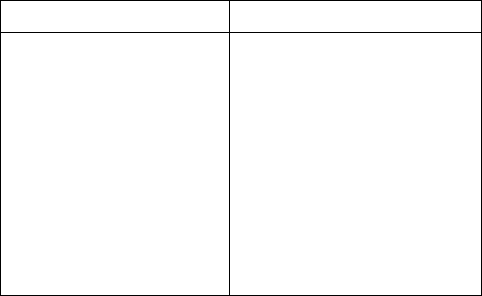
*2.1. Простейшие* ***SELECT****-запросы* 21



Приведенный запрос осуществляет выборку всех значений полей

NAME и SURNAME из таблицы STUDENT.

Его результатом является таблица следующего вида:



NAME

Иван Петр Вадим Борис Ольга Андрей Павел Артем Антон Вадим

. . .

SURNAME

Иванов Петров Сидоров Кузнецов Зайцева Павлов Котов Лукин Петров Белкин

. . .

Порядок следования столбцов в этой таблице соответствует порядку полей NAME и SURNAME, указанному в запросе, а не их порядку во входной таблице STUDENT.

Если необходимо вывести значения *всех* столбцов таблицы, то мож- но вместо перечисления их имен использовать символ “\*” (звездочка).

SELECT \*

**FROM** STUDENT;

В данном случае в результате выполнения запроса будет получена вся таблица STUDENT.

Еще раз обратим внимание на то, что получаемые в результате SQL-запроса таблицы не в полной мере отвечают определению реля- ционного отношения. В частности, в них могут оказаться кортежи с одинаковыми значениями атрибутов.

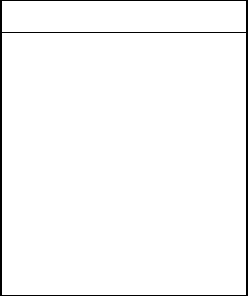
Например, запрос “Получить список названий городов, где прожи- вают студенты, сведения о которых находятся в таблице STUDENT”, можно записать в следующем виде:

**SELECT** CITY **FROM** STUDENT;

Его результатом будет таблица

22 *Гл. 2. Выборка данных (оператор* ***SELECT****)*





CITY

Орел Курск Москва Брянск Липецк **Воронеж** Белгород **Воронеж NULL**

**Воронеж**

. . .

Видно, что в таблице встречаются одинаковые строки (выделены жирным шрифтом).

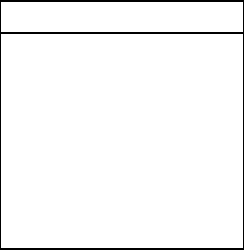
Для исключения из результата **SELECT**-запроса повторяющихся записей используется ключевое слово **DISTINCT** (ОТЛИЧНЫЙ). Если запрос **SELECT** извлекает множество полей, то **DISTINCT** *исключает* дубликаты строк, в которых значения *всех* выбранных полей идентич- ны.

Запрос “Определить список названий *различных* городов, где про- живают студенты, сведения о которых находятся в таблице STUDENT”, можно записать в следующем виде.

SELECT DISTINCT CITY

**FROM** STUDENT;

В результате получим таблицу, в которой дубликаты строк исклю- чены:



CITY

Орел Курск Москва Брянск Липецк Воронеж Белгород **NULL**

. . .

Ключевое слово **ALL** (ВСЕ), в отличие от **DISTINCT**, оказывает противоположное действие, т. е. при его использовании повторяющиеся строки *включаются* в состав выходных данных. Режим, задаваемый ключевым словом **ALL**, действует по умолчанию, поэтому в реальных запросах для этих целей оно практически не используется.

1. *Простейшие* ***SELECT****-запросы* 23



Использование в операторе **SELECT** предложения, определяемого ключевым словом **WHERE** (ГДЕ), позволяет задавать выражение усло- вия (предикат), принимающее значение *истина* или *ложь* (а также *неизвестно* при использовании **NULL**) для значений полей строк таб- лиц, к которым обращается оператор **SELECT**. Предложение **WHERE** определяет, ***какие строки*** *указанных таблиц должны быть выбраны*. В таблицу, являющуюся результатом запроса, включаются только те строки, для которых условие (предикат), указанное в предложении **WHERE**, принимает значение *истина*.

Пример.

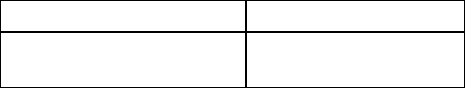
Написать запрос, выполняющий выборку имен (NAME) всех студен- тов с фамилией (SURNAME) Петров, сведения о которых находятся в таблице STUDENT.

**SELECT** SURNAME, NAME

**FROM** STUDENT

**WHERE** SURNAME = ‘Петров’;

Результатом этого запроса будет таблица:



SURNAME

Петров Петров

NAME

Петр Антон

В задаваемых в предложении **WHERE** условиях могут использо- ваться операции сравнения, определяемые следующими операторами:

= (равно), *>* (больше), *<* (меньше), *>*= (больше или равно), *<*= (мень- ше или равно), *<>* (не равно), а также логические операторы **AND**, **OR** и **NOT**.

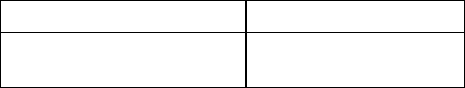
Например, запрос для получения *имен* и *фамилий* студентов, обу- чающихся на *третьем* курсе и получающих стипендию (размер сти- пендии *больше нуля*) будет выглядеть таким образом:

**SELECT** NAME, SURNAME

**FROM** STUDENT

**WHERE** (KURS = 3 **AND** STIPEND > 0);

Результат выполнения этого запроса имеет вид:



SURNAME

Петров Лукин

NAME

Петр Артем

УПРАЖНЕНИЯ

1. Напишите запрос к таблице SUBJECT, выводящий для каждой ее строки идентификатор (номер) предмета обучения, его наимено- вание, семестр, в котором он читается, и количество отводимых на него часов.
2. Напишите запрос, позволяющий вывести все строки табли- цы EXAM\_MARKS, в которых предмет обучения имеет номер (SUBJ\_ID), равный 12.
3. Напишите запрос, выбирающий все данные из таблицы STUDENT, расположив столбцы таблицы в следующем порядке: KURS, SURNAME, NAME, STIPEND.
4. Напишите запрос **SELECT**, который для каждого предмета обуче- ния (SUBJECT) выполняет вывод его наименования (SUBJ\_NAME) и следом за ним количества часов (HOUR) в 4-м семестре (SEMESTR).
5. Напишите запрос, позволяющий получить из таблицы EXAM\_MARKS значения столбца MARK (экзаменационная оценка) для всех студентов, исключив из списка повторение одинаковых строк.
6. Напишите запрос, который выполняет вывод списка фамилий студентов, обучающихся на третьем и более старших курсах.
7. Напишите запрос, выбирающий данные фамилию, имя и номер курса для студентов, получающих стипендию больше 140.
8. Напишите запрос, выполняющий выборку из таблицы SUBJECT названий всех предметов обучения, на которые отводится более 30 часов.
9. Напишите запрос, который выполняет вывод списка университе- тов, рейтинг которых превышает 300 баллов.
10. Напишите запрос к таблице STUDENT для вывода списка всех студентов со стипендией не меньше 100, живущих в Воронеже — с указанием фамилии (SURNAME), имени (NAME) и номера курса (KURS).
11. Какие данные будут получены в результате выполнения запроса?

SELECT \*

**FROM** STUDENT

**WHERE** (STIPEND < 100 **OR**

**NOT** (BIRTHDAY >= ‘10/03/1980’

**AND** STUDENT\_ID > 1003));

1. Какие данные будут получены в результате выполнения запроса?

SELECT \*

**FROM** STUDENT

**WHERE NOT** ((BIRTHDAY = ‘10/03/1980’ **OR**

STIPEND > 100)

**AND** STUDENT\_ID >= 1003);

1. Напишите запрос для получения списка студентов старше 25 лет, обучающихся на 1-м курсе.
2. Напишите запрос для получения списка предметов, для которых в 1-м семестре отведено более 100 часов.
3. Напишите запрос для получения списка преподавателей, живу- щих в Воронеже.
4. Напишите запрос для получения списка университетов, распо- ложенных в Москве и имеющих рейтинг меньший, чем у ВГУ. Константу в ограничении на рейтинг можно определить по этой же таблице.
5. Напишите запрос для получения списка студентов, проживаю- щих в Воронеже и не получающих стипендию.
6. Напишите запрос для получения списка студентов моложе 20 лет.
7. Напишите запрос для получения списка студентов без опреде- ленного места жительства.
8. Операторы IN, BETWEEN, LIKE, IS NULL

При задании логического условия в предложении **WHERE** могут быть использованы операторы **IN**, **BETWEEN**, **LIKE**, **IS NULL**.

Операторы **IN** (РАВЕН ЛЮБОМУ ИЗ СПИСКА) и **NOT IN** (НЕ РА-

ВЕН НИ ОДНОМУ ИЗ СПИСКА) используются для сравнения проверя- емого значения поля с заданным списком. Этот список значений ука- зывается в скобках справа от оператора **IN**. Список значений не обяза- тельно задается в явном виде, он может представлять собой результат подзапроса.

Построенный с использованием **IN** предикат (условие) считается истинным, если значение поля, имя которого указано слева от **IN**, в точности *совпадает* с одним из значений, перечисленных в списке, указанном в скобках справа от **IN**.

Предикат, построенный с использованием **NOT IN**, считается ис- тинным, если значение поля, имя которого указано слева от **NOT IN**, *не совпадает* ни с одним из значений, принадлежащих списку, указанному в скобках справа от **NOT IN**.

Пример 1.

Получить из таблицы EXAM\_MARKS сведения о студентах, *имеющих*

экзаменационные оценки только 4 и 5.

SELECT \*

**FROM** EXAM\_MARKS

**WHERE** MARK **IN** (4, 5);

Пример 2.

Получить сведения о студентах, *не имеющих* ни одной экзамена- ционной оценки, равной 4 или 5.

SELECT \*

**FROM** EXAM\_MARKS

**WHERE** MARK **NOT IN** (4, 5);

Оператор **BETWEEN** используется для проверки условия вхождения значения поля в заданный интервал, т. е. вместо списка значений атрибута этот оператор задает границы его изменения.

Например, запрос, выполняющий вывод записей о предметах обуче- ния, для которых количество отводимых часов лежит в пределах между 30 и 40, имеет вид:

SELECT \*

**FROM** SUBJECT

**WHERE** HOUR **BETWEEN** 30 **AND** 40;

Граничные значения, в данном случае значения 30 и 40, *входят* во множество значений, с которыми производится сравнение. Оператор **BETWEEN** может использоваться как для числовых, так и для символь- ных типов полей.

Оператор **LIKE** применим только к символьным полям типа **CHAR** или **VARCHAR**. Этот оператор осуществляет просмотр строковых зна- чений полей с целью определения, входит ли заданная в операторе **LIKE** подстрока (образец поиска) в символьную строку, являющуюся значением проверяемого поля.

Для того чтобы осуществлять выборку строковых значений по заданному образцу подстроки, можно применять шаблон искомого об- разца строки, использующий следующие символы:

символ подчеркивания “**\_**”, указанный в шаблоне образца, опре- деляет возможность наличия в указанном месте *одного любого* символа,

*•*

символ “%” допускает присутствие в указанном месте проверя- емой строки последовательности любых символов произвольной длины.

*•*

Пример.

Написать запрос, выбирающий из таблицы **STUDENT** сведения о студентах, у которых фамилии начинаются на букву “Р”.

SELECT \*

**FROM** STUDENT

**WHERE** SURNAME **LIKE** ‘Р%’;

Так как символы “**\_**” и “%” выполняют в языке SQL указанные выше специальные функции, возникает проблема, когда необходимо их указывать в текстовом образце в качестве обычных, а не служебных символов. В этих случаях применяют специальный механизм, позволя- ющий при интерпретации системой строки-образца отключить управ- ляющие функции этих символов. Отключить служебные функции си- волов “**\_**” и “%” можно путем вставки непосредственно перед ними так

называемого escape-символа (escape character). Этот символ, который можно еще назвать знаком перехода или знаком отключения, является служебным знаком, используемым для указания того, что должен быть изменен характер интерпретации следующего непосредственно за ним символа. В нашем случае если такой символ предшествует знаку “**\_**” или “%”, то этот знак будет интепретироваться уже буквально, как любой другой символ, а не как служебный символ. В SQL в качестве такого переключающего (escape) символа может быть назначен любой символ. для этих целей предназначено специальное ключевое слово **ESCAPE**.

Например, можно задать образец поиска с помощью следующего выражения:

**LIKE** ‘\_***\***\_Р’ **ESCAPE** ‘***\***’.

В этом выражении символ “ ” с помощью ключевого слова **ESCAPE** объявляется escape-символом. Первый символ “**\_**” в заданном шаблоне поиска “\_ \_Р” будет соответствовать, как и ранее, любому набору символов в проверяемой строке. Однако второй символ “**\_**”, следующий после символа “ ”, объявленного escape-символом, уже будет интер- претироваться буквально как обычный символ, так же как и символ Р в заданном шаблоне.

***\***

***\***

***\***

Обращаем ваше внимание на то, что в операторах сравнения =, *<*,

*>*, *<*=, *>*=, *<>* и операторах **IN**, **BETWEEN** и **LIKE** при использовании **NULL** в качестве операнда будет возвращаться также **NULL**. В связи с этим, для проверки содержимого поля на наличие (отсутствие) в нем пустого значения **NULL** следует использовать специально пред- назначенные для этого операторы **IS NULL** (ЯВЛЯЕТСЯ ПУСТЫМ) и **IS NOT NULL** (НЕ ЯВЛЯЕТСЯ ПУСТЫМ), а не выражения = **NULL** или

*<>* **NULL**.

УПРАЖНЕНИЯ

1. Напишите запрос, выполняющий вывод находящихся в таблице EXAM\_MARKS номеров предметов обучения, экзамены по которым сдавались между 10 и 20 января 2005 г.
2. Напишите запрос, выбирающий данные обо всех предметах обу- чения, экзамены по которым сданы студентами, имеющими иден- тификаторы 12 и 32.
3. Напишите запрос, который выполняет вывод названий предметов обучения, начинающихся на букву ‘И’.
4. Напишите запрос, выбирающий сведения о студентах, у которых имена начинаются на букву ‘И’ или ‘С’.
5. Напишите запрос для выбора из таблицы EXAM\_MARKS записей, для которых отсутствуют значения оценок (поле MARK).
6. Напишите запрос, выполняющий вывод из таблицы EXAM\_MARKS

записей, для которых в поле MARK проставлены значения оценок.

1. Напишите запрос для получения списка преподавателей, прожи- вающих в городах, в названиях которых присутствует дефис.
2. Напишите запрос для получения списка учебных заведений, в названиях которых использованы кавычки.
3. Напишите запрос для получения списка предметов, названия которых оканчиваются на ‘ия’.
4. Напишите запрос для получения списка учебных заведений, в названиях которых содержится слово ‘университет’.
5. Напишите запрос для получения списка студентов, фамилии ко- торых начинаются на ‘Ков’ или на ‘Куз’.
6. Напишите запрос для получения списка предметов обучения, названия которых состоят из более одного слова.
7. Напишите запрос для получения списка учебных заведений, на- звания которых состоят как минимум из 7 слов.
8. Напишите запрос для получения списка студентов, фамилии ко- торых состоят из трех букв.
9. Преобразование вывода и встроенные функции

В SQL реализованы операторы преобразования данных и встроен- ные функции, предназначенные для работы со значениями столбцов и/или константами в выражениях. Использование этих операторов допустимо в запросах везде, где можно использовать выражения.

* 1. **Числовые, символьные и строковые константы.** Несмот- ря на то, что SQL работает с данными в понятиях строк и столб- цов таблиц, имеется возможность применения значений выражений, построенных с использованием встроенных функций, констант, имен столбцов, которые определяются как своего рода виртуальные столбцы. Они помещаются в списке столбцов и могут сопровождаться псевдони- мами.

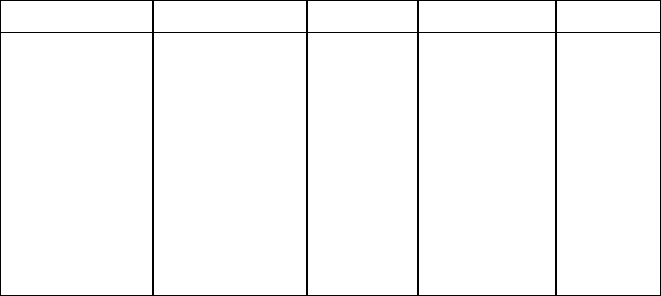
Если в запросе вместо спецификации столбца SQL обнаруживает

*число*, то оно интерпретируется как *числовая константа*.

*Символьные константы* должны указываться в одинарных кавыч- ках. Если одинарная кавычка должна выводиться как часть строковой константы, то ее нужно предварить другой одинарной кавычкой.

Например, результатом выполнения запроса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | SURNAME |  | NAME |  |
| Фамилия | Иванов | Имя | Иван | 100 |
| Фамилия | Петров | Имя | Петр | 100 |
| Фамилия | Сидоров | Имя | Вадим | 100 |
| Фамилия | Кузнецов | Имя | Борис | 100 |
| Фамилия | Зайцева | Имя | Ольга | 100 |
| Фамилия | Павлов | Имя | Андрей | 100 |
| Фамилия | Котов | Имя | Павел | 100 |
| Фамилия | Лукин | Имя | Артем | 100 |
| Фамилия | Петров | Имя | Антон | 100 |
| Фамилия | Белкин | Имя | Вадим | 100 |
| . . . | . . . | . . . | . . . | . . . |

* 1. Арифметические операции для преобразования число- вых данных.

Унарный (одиночный) оператор (знак минус) изменяет знак числового значения, перед которым он стоит, на противополож- ный.

* *−*

Бинарные операторы +, , **\*** и **/** предоставляют возможность вы- полнения арифметических операций сложения, вычитания, умно- жения и деления.

* *−*

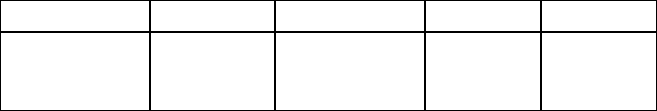
Например, результат запроса

**SELECT** SURNAME, NAME, STIPEND, -(STIPEND\*KURS)/2

**FROM** STUDENT

**WHERE** KURS = 4 **AND** STIPEND > 0;

будет выглядеть следующим образом



. . .

. . .

. . .

. . .

*−*400

. . .

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SURNAME | NAME | STIPEND | KURS |  |
| Сидоров Петров | Вадим Антон | 150  200 | 4  4 | *−*300 |

* 1. **Символьная операция конкатенации строк.** Операция конкатенации, обозначаемая символом “ ”, позволяет соединять (“скле- ивать”) значения двух или более столбцов символьного типа или сим- вольных констант в одну строку.

***"***

Эта операция имеет синтаксис

*<значимое символьное выражение>*

***"***

*<значимое символьное выражение>*.

**SELECT** ‘Фамилия’, SURNAME, ‘Имя’, NAME, 100

**FROM** STUDENT;

является таблица следующего вида:

Например:

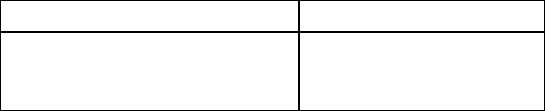
**SELECT** SURNAME ‘\_’ NAME, STIPEND

***" "***

**FROM** STUDENT

**WHERE** KURS = 4 **AND** STIPEND > 0;

Результат запроса будет выглядеть следующим образом:



Сидоров\_Вадим Петров\_Антон

. . .

STIPEND

150

200

. . .

* 1. Символьные функции преобразования букв различных слов в строке.
* **LOWER** — перевод в строчные символы (нижний регистр)

**LOWER**(*<строка>*)

* **UPPER** — перевод в прописные символы (верхний регистр)

**UPPER**(*<строка>*)

**INITCAP** — перевод первой буквы каждого слова строки в за- главную (прописную)

*•*

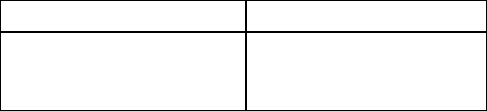
**INITCAP**(*<строка>*) Например:

**SELECT LOWER**(SURNAME), **UPPER**(NAME)

**FROM** STUDENT

**WHERE** KURS = 4 **AND** STIPEND > 0;

Результат запроса будет выглядеть следующим образом



SURNAME

Сидоров Петров

. . .

NAME

ВАДИМ АНТОН

. . .

* 1. Символьные строковые функции.
* **LPAD** — дополнение строки слева

**LPAD**(*<строка>*, *<длина>*[, *<подстрока>*])

* + *<строка>* дополняется ***слева*** указанной в *<подстроке>* по- следовательностью символов до указанной *<длины>* (воз- можно, с повторением последовательности);
  + если *<подстрока>* не указана, то по умолчанию *<строка>*

дополняется пробелами;

* + если *<длина>* меньше длины *<строки>*, то исходная *<стро- ка>* усекается слева до заданной *<длины>*.
* **RPAD** — дополнение строки справа

**RPAD**(*<строка>*, *<длина>*[, *<подстрока>*])

* + *<строка>* дополняется ***справа*** указанной в *<подстроке>* последовательностью символов до указанной *<длины>* (воз- можно, с повторением последовательности);
  + если *<подстрока>* не указана, то по умолчанию *<строка>*

дополняется пробелами;

* + если *<длина>* меньше длины *<строки>*, то исходная *<стро- ка>* усекается справа до заданной *<длины>*.
* **LTRIM** — удаление левых граничных символов

**LTRIM**(*<строка>*[, *<подстрока>*])

* + из *<строки>* удаляются слева символы, указанные в *<под- строке>*;
  + если *<подстрока>* не указана, то по умолчанию удаляются пробелы;
  + в *<строку>* справа добавляется столько пробелов, сколько символов слева было удалено, т. е. длина *<строки>* остается неизменной.
* **RTRIM** — удаление правых граничных символов

**RTRIM**(*<строка>*[, *<подстрока>*])

* + из *<строки>* удаляются справа символы, указанные в *<под- строке>*;
  + если *<подстрока>* не указана, то по умолчанию удаляются пробелы;
  + в *<строку>* слева добавляется столько пробелов, сколько символов справа было удалено, т. е. длина *<строки>* оста- ется неизменной.

Функции **LTRIM** и **RTRIM** рекомендуется использовать при написа- нии условных выражений, в которых сравниваются текстовые строки. Дело в том, что наличие начальных или конечных пробелов в сравни- ваемых операндах может исказить результат сравнения.

Например, константы ‘ ААА’ и ‘ ААА ’ не равны друг другу.

* **SUBSTR** — выделение подстроки

**SUBSTR**(*<строка>*, *<начало>*[, *<количество>*])

* + из *<строки>* выбирается заданное *<количество>* символов, начиная с указанной позиции в строке *<начало>*;
  + если *<количество>* не задано, символы выбираются с *<на- чала>* и до конца *<строки>*;
  + возвращается подстрока, содержащая число символов, за- данное параметром *<количество>*, либо число символов от позиции, заданной параметром *<начало>*, до конца *<стро- ки>*;
  + если указанное *<начало>* превосходит длину *<строки>*, то возвращается строка, состоящая из пробелов. Длина этой строки будет равна заданному *<количеству>* или исходной длине *<строки>* (при не заданном *<количестве>*).
* **INSTR** — поиск подстроки

**INSTR**(*<строка>*, *<подстрока>*[, *<начало поиска>*[,

*<номер вхождения>*]])

* + *<начало поиска>* задает начальную позицию в строке для поиска *<подстроки>*; если не задано, то по умолчанию принимается значение 1;
  + *<номер вхождения>* задает порядковый номер искомой под- строки; если не задан, то по умолчанию принимается значе- ние 1;
  + значимые выражения в *<начале поиска>* или в *<номере вхождения>* должны иметь беззнаковый целый тип или при- водиться к этому типу;
  + тип возвращаемого значения — **INT**;
  + функция возвращает позицию найденной подстроки.
* **LENGTH** — определение длины строки

**LENGTH**(*<строка>*)

* + длина *<строки>*, тип возвращаемого значения — **INT**;
  + функция возвращает **NULL**, если *<строка>* имеет **NULL**-зна- чение.

Пример 1.

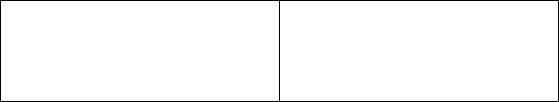
Результат запроса

**SELECT LPAD**(SURNAME, 10, ‘@’), **RPAD**(NAME, 10, ‘$’)

**FROM** STUDENT

**WHERE** KURS = 3 **AND** STIPEND > 0;

будет выглядеть следующим образом



@@@@Петров

@@@@Павлов @@@@@Лукин

. . .

Петр$$$$$$

Андрей$$$$ Артем$$$$$

. . .

Пример 2.

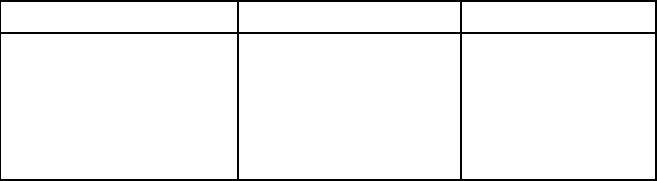
Запрос

**SELECT SUBSTR**(NAME, 1, 1) ‘.’ SURNAME, CITY,

***" "***

**LENGTH**(CITY) **FROM** STUDENT

**WHERE** KURS **IN**(2, 3, 4) **AND** STIPEND > 0;

выдаст результат

|  |  |  |
| --- | --- | --- |
|  | CITY |  |
| П.Петров | Курск | 5 |
| С.Сидоров | Москва | 6 |
| О.Зайцева | Липецк | 6 |
| А.Лукин | Воронеж | 7 |
| А.Петров | **NULL** | **NULL** |
| . . . | . . . | . . . |

* 1. Функции работы с числами.
* **ABS** — абсолютное значение

**ABS**(*<значимое числовое выражение>*)

**FLOOR** — наибольшее целое, не превосходящее заданное число с плавающей точкой

*•*

**FLOOR**(*<значимое числовое выражение>*)

**CEIL** — наименьшее целое, которое равно или больше заданного числа

*•*

**CEIL**(*<значимое числовое выражение>*)

* Функция округления — **ROUND**

**ROUND**(*<значимое числовое выражение>*, *<точность>*) аргумент *<точность>* задает точность округления (см. пример ниже).

* Функция усечения — **TRUNC**

**TRUNC**(*<значимое числовое выражение>*, *<точность>*)

* Тригонометрические функции — **COS**, **SIN**, **TAN COS**(*<значимое числовое выражение>*)

**SIN**(*<значимое числовое выражение>*)

**TAN**(*<значимое числовое выражение>*)

* Гиперболические функции — **COSH**, **SINH**, **TANH COSH**(*<значимое числовое выражение>*)

**SINH**(*<значимое числовое выражение>*)

**TANH**(*<значимое числовое выражение>*)

2 И. Ф. Астахова, В. М. Мельников, А. П. Толстобров, В. В. Фертиков

* Экспоненциальная функция — **EXP EXP**(*<значимое числовое выражение>*)
* Логарифмические функции — **LN**, **LOG**

**LN**(*<значимое числовое выражение>*)

**LOG**(*<значимое числовое выражение>*)

* Функция возведения в степень — **POWER POWER**(*<значимое числовое выражение>*,

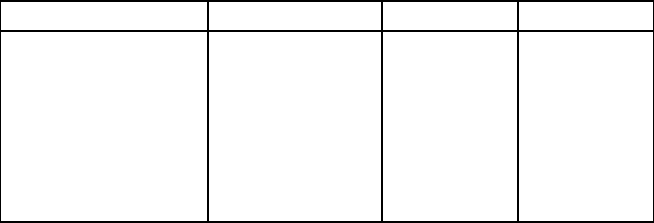
*<показатель степени>*)

* Определение знака числа — **SIGN SIGN**(*<значимое числовое выражение>*)
* Вычисление квадратного корня — **SQRT SQRT**(*<значимое числовое выражение>*)

Пример.

Запрос

**SELECT** UNIV\_NAME, RATING, **ROUND**(RATING, –1), **TRUNC**(RATING, –1) **FROM** UNIVERSITY;

вернет результат

|  |  |  |  |
| --- | --- | --- | --- |
| UNIV\_NAME | RATING |  | |
| МГУ | 610 | 610 | 600 |
| ВГУ | 296 | 300 | 290 |
| НГУ | 345 | 350 | 340 |
| РГУ | 421 | 420 | 410 |
| БГУ | 326 | 330 | 320 |
| ТГУ | 373 | 370 | 360 |
| ВГМА | 327 | 330 | 320 |
| . . . | . . . | . . . | . . . |

* 1. Функции преобразования значений.
* Преобразование в символьную строку — **TO\_CHAR TO\_CHAR**(*<значимое выражение>*[, *<символьный формат>*])
  + *<значимое выражение>* должно представлять числовое зна- чение или значение типа дата-время;
  + для числовых значений *<символьный формат>* должен иметь синтаксис [S]9[9...][.9[9...]], где S — представление знака числа (при отсутствии предполагается без отображе-

ния знака), 9 — представление цифр-знаков числового зна- чения (для каждого знакоместа). Символьный формат опре- деляет вид отображения чисел. По умолчанию для числовых значений используется формат ‘999999.99’;

* + для значений типа **ДАТА-ВРЕМЯ** *<символьный формат>*

имеет вид (т. е. вид отображения значений даты и времени)

— в части даты:

‘DD-Mon-YY’

‘DD-Mon-YYYY’ ‘MM/DD/YY’ ‘MM/DD/YYYY’ ‘DD.MM.YY’

‘DD.MM.YYYY’

— в части времени:

‘HH24’ ‘HH24:MI’

‘HH24:MI:SS’

‘HH24:MI:SS.FF’

где:

НН24 — часы в диапазоне от 0 до 24;

MI — минуты;

SS — секунды;

FF — тики (сотые доли секунды).

При выводе времени в качестве разделителя по умолчанию исполь- зуется двоеточие (“:”), но при желании можно использовать любой другой символ.

Возвращаемое значение — символьное представление *<значимого выражения>* в соответствии с заданным *<символьным форматом>* преобразования.

Преобразование из символьного значения в числовое —

*•*

TO\_NUMBER

**TO\_NUMBER**(*<значимое символьное выражение>*)

При этом *<значимое символьное выражение>*должно задавать символьное значение числового типа.

* Преобразование символьной строки в дату — **TO\_DATE**

**TO\_DATE**(*<значимое символьное выражение>*[,

*<символьный формат>*])

* + *<значимое символьное выражение>* должно задавать сим- вольное значение типа **ДАТА-ВРЕМЯ**;

2\*

**–** *<символьный формат>* должен описывать представление значения типа **ДАТА-ВРЕМЯ** в *<значимом символьном вы- ражении>*. Допустимые форматы (в том числе и формат по умолчанию) приведены выше.

Возвращаемое значение — *<значимое символьное выражение>* во внутреннем представлении. Тип возвращаемого значения — **DATE**. Над значениями типа **DATE** разрешены следующие операции:

к значению типа **DATE** можно прибавлять значения типа **INTERVAL**, в результате чего получается значение типа **DATE**; при вычитании двух значений типа **DATE** получается значение типа **INTERVAL**;

*•*

*•*

при вычитании из значения типа **DATE** значения типа **INTERVAL**

*•*

получается значение типа **DATE**.

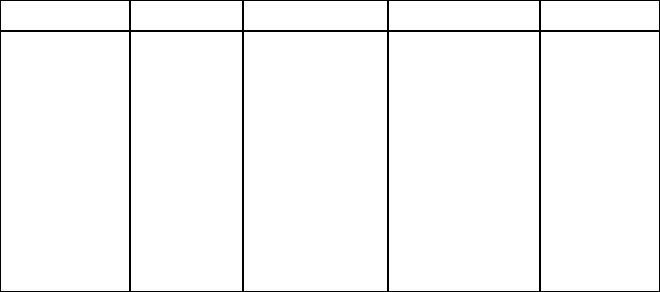
В бинарных операциях один из операндов должен иметь значение отдельного элемента даты: только год, или только месяц, или только день.

Пример.

Запрос

**SELECT** SURNAME, NAME, BIRTHDAY, **TO\_CHAR**(BIRTHDAY, ‘DD-Mon-YYYY’), **TO\_CHAR**(BIRTHDAY, ‘DD.MM.YY’)

**FROM** STUDENT;

вернет результат

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SURNAME | NAME | BIRTHDAY |  | |
| Иванов | Иван | 3/12/1988 | 3-дек-1988 | 3.12.88 |
| Петров | Петр | 11/12/1986 | 11-дек-1986 | 11.12.86 |
| Сидоров | Вадим | 7/06/1985 | 7-июн-1985 | 7.06.85 |
| Кузнецов | Борис | 8/12/1987 | 8-дек-1987 | 8.12.87 |
| Зайцева | Ольга | 21/05/1987 | 21-май-1987 | 21.05.87 |
| Павлов | Андрей | 5/11/1985 | 5-ноя-1985 | 5.11.85 |
| Котов | Павел | **NULL** | **NULL** | **NULL** |
| Лукин | Артем | 11/12/1987 | 11-дек1987 | 11.12.87 |
| Петров | Антон | 5/08/1987 | 5-авг-1987 | 5.08.87 |
| Белкин | Вадим | 20/01/1986 | 20-янв-1986 | 20.01.86 |
| . . . | . . . | . . . | . . . | . . . |

Функция **CAST** является средством явного преобразования данных из одного типа в другой. Синтаксис этой команды имеет вид

**CAST** *<значимое выражение>* **AS** *<тип данных>*

*<значимое выражение>* должно иметь числовой или символьный тип языка SQL (возможно, с указанием длины, точности и мас- штаба) или быть **NULL**-значением.

*•*

Любое числовое выражение может быть явно преобразовано в любой другой числовой тип.

Символьное выражение может быть преобразовано в любой чис- ловой тип. При этом в результате такого преобразования отсе- каются начальные и конечные пробелы, а остальные символы преобразуются в числовое значение по правилам языка SQL. Если заданная явным образом длина символьного типа недо- статочна и преобразованное значение не размещается в нем, то результативное значение усекается справа.

*•*

*•*

*•*

Возможно явное преобразование символьного типа в символь- ный с другой длиной. Если длина результата больше длины аргумента, то значение дополняется пробелами; если меньше, то усекается.

*•*

**NULL** преобразуется в **NULL**.

*•*

Числовое выражение может быть преобразовано в символьный тип.

*•*

Пример.

**SELECT CAST** STUDENT\_ID **AS CHAR**(10) **FROM** STUDENT;

УПРАЖНЕНИЯ

1. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала один столбец, содержащий после- довательность разделенных символом “;” (точка с запятой) зна- чений всех столбцов этой таблицы; при этом текстовые значения должны отображаться прописными символами (верхний регистр), т. е. быть представленными в следующем виде: 10;КУЗНЕЦОВ;БОРИС;0;БРЯНСК; 8.12.1987;10.
2. Составьте запрос для таблицы STUDENT таким образом, что- бы выходная таблица содержала всего один столбец в сле- дующем виде: Б.КУЗНЕЦОВ; место жительства - БРЯНСК; родился - 8.12.87.
3. Составьте запрос для таблицы STUDENT таким образом, что- бы выходная таблица содержала всего один столбец в сле- дующем виде: б.кузнецов; место жительства - брянск; родился: 8-дек-1987.
4. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Борис Кузнецов родился в 1987 году.
5. Составьте запрос, выводящий фамилии, имена студентов и вели- чину получаемых ими стипендий, при этом значения стипендий должны быть увеличены в 100 раз.
6. То же, что и в упр. 4, но только для студентов 1, 2 и 4 курсов и таким образом, чтобы фамилии и имена были выведены пропис- ными буквами.

38 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



1. Составьте запрос для таблицы UNIVERSITY таким образом, что- бы выходная таблица содержала всего один столбец в следующем виде: Код-10; ВГУ-г.ВОРОНЕЖ; Рейтинг=296.
2. То же, что и в упр. 7, но значения рейтинга требуется округлить до первого знака (например, значение 382 округляется до 400).

2.4. Агрегирование и групповые функции

Агрегирующие функции позволяют получать из таблицы сводную (агрегированную) информацию, выполняя операции над группой строк таблицы. Для задания в **SELECT**-запросе агрегирующих операций ис- пользуются следующие ключевые слова:

**COUNT** определяет количество строк или значений поля, выбран- ных посредством запроса и не являющихся **NULL**-значениями; **SUM** вычисляет арифметическую сумму всех выбранных значений данного поля;

*•*

*•*

**AVG** вычисляет среднее значение для всех выбранных значений данного поля;

*•*

**MAX** вычисляет наибольшее из всех выбранных значений данного поля;

*•*

**MIN** вычисляет наименьшее из всех выбранных значений данного поля.

*•*

В **SELECT**-запросе агрегирующие функции используются аналогич- но именам полей, при этом последние (имена полей) используются в качестве аргументов этих функций.

Функция **AVG** предназначена для подсчета среднего значения поля на множестве записей таблицы.

Например, для определения среднего значения поля MARK (оценки) по всем записям таблицы EXAM\_MARKS можно использовать запрос с функцией **AVG** следующего вида:

**SELECT AVG**(MARK) **FROM** EXAM\_MARKS;

Для подсчета общего количества строк в таблице следует исполь- зовать функцию **COUNT** со звездочкой:

**SELECT COUNT**(\*) **FROM** EXAM\_MARKS;

При подсчете значений конкретных атрибутов аргументы **DISTINCT** и **ALL** позволяют соответственно исключать и включать дубликаты обрабатываемых функцией **COUNT** значений. При этом необходимо учитывать, что при использовании опции **ALL** неопре- деленные значения атрибута (**NULL**) все равно не войдут в число подсчитываемых значений.

* 1. *Агрегирование и групповые функции* 39



**SELECT COUNT**(**DISTINCT** SUBJ\_ID)

**FROM** SUBJECT;

Предложение **GROUP BY** (ГРУППИРОВАТЬ ПО) позволяет группи- ровать записи в подмножества, определяемые значениями какого-либо поля, и применять агрегирующие функции уже не ко всем записям таблицы, а раздельно к каждой сформированной группе.

Предположим, требуется найти максимальное значение оценки, по- лученной каждым студентом. Запрос будет выглядеть следующим об- разом:

**SELECT** STUDENT\_ID, **MAX**(MARK) **FROM** EXAM\_MARKS

**GROUP BY** STUDENT\_ID;

Выбираемые из таблицы EXAM\_MARKS записи группируются по значениям поля STUDENT\_ID, указанного в предложении **GROUP BY**, и для каждой группы находится максимальное значение поля MARK. Предложение **GROUP BY** позволяет применять агрегирующие функции к каждой группе, определяемой общим значением поля или полей, ука- занных в этом предложении. В приведенном запросе рассматриваются группы записей, сгруппированные по идентификаторам студентов.

В конструкции **GROUP BY** для группирования может быть исполь- зовано более одного столбца. Например:

**SELECT** STUDENT\_ID, SUBJ\_ID, **MAX**(MARK) **FROM** EXAM\_MARKS

**GROUP BY** STUDENT\_ID, SUBJ\_ID;

В этом случае строки вначале группируются по значениям первого столбца, а внутри этих групп — в подгруппы по значениям второго столбца. Таким образом, **GROUP BY** не только устанавливает столбцы, по которым осуществляется группировка, но и указывает порядок раз- биения столбцов на группы.

Следует иметь в виду, что после ключевого слова **SELECT** должны быть использованы только те имена столбцов, которые указаны в предложении **GROUP BY**.

При необходимости часть сформированных с помощью **GROUP BY**

групп может быть исключена с помощью предложения **HAVING**.

Предложение **HAVING** определяет критерий, по которому группы следует включать в выходные данные (по аналогии с предложением **WHERE**, которое осуществляет это для отдельных строк):

**SELECT** SUBJ\_NAME, **MAX**(HOUR) **FROM** SUBJECT

**GROUP BY** SUBJ\_NAME

**HAVING MAX**(HOUR) >= 72;

40 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



В условии, задаваемом предложением **HAVING**, должны быть указа- ны только поля или выражения, которые на выходе имеют единствен- ное значение для каждой выводимой группы.

УПРАЖНЕНИЯ

1. Напишите запрос для подсчета количества студентов, сдававших экзамен по предмету обучения с идентификатором 20.
2. Напишите запрос, который позволяет подсчитать в таблице

EXAM\_MARKS количество различных предметов обучения.

1. Напишите запрос, который для каждого студента выполняет вы- борку его идентификатора и минимальной из полученных им оценок.
2. Напишите запрос, который для каждого студента выполняет вы- борку его идентификатора и максимальной из полученных им оценок.
3. Напишите запрос, выполняющий вывод первой по алфавиту фа- милии студента, начинающейся на букву ‘И’.
4. Напишите запрос, который для каждого предмета обучения вы- водит наименование предмета и максимальное значение номера семестра, в котором этот предмет преподается.
5. Напишите запрос, который для каждого конкретного дня сдачи экзамена выводит данные о количестве студентов, сдававших экзамен в этот день.
6. Напишите запрос, выдающий средний балл для каждого студента.
7. Напишите запрос, выдающий средний балл для каждого экзамена.
8. Напишите запрос, определяющий количество сдававших студен- тов для каждого экзамена.
9. Напишите запрос для определения количества предметов, изуча- емых на каждом курсе.
10. Для каждого университета напишите запрос, выводящий суммар- ную стипендию обучающихся в нем студентов, с последующей сортировкой списка по этому значению.
11. Для каждого семестра напишите запрос, выводящий общее количество часов, отводимое на изучение соответствующих предметов.
12. Для каждого студента напишите запрос, выводящий среднее зна- чение оценок, полученных им на всех экзаменах.
13. Для каждого студента напишите запрос, выводящий среднее зна- чение оценок, полученных им по каждому предмету.
14. Напишите запрос, выводящий количество студентов, проживаю- щих в каждом городе. Список отсортировать в порядке убывания количества студентов.
    1. *Неопределенные значения (****NULL****) в агрегирующих функциях* 41



1. Для каждого университета напишите запрос, выводящий количе- ство обучающихся в нем студентов, с последующей сортировкой списка по этому количеству.
2. Для каждого университета напишите запрос, выводящий коли- чество работающих в нем преподавателей, с последующей сорти- ровкой списка по этому количеству.
3. Для каждого университета напишите запрос, выводящий сумму стипендии, выплачиваемой студентам каждого курса.
4. Для каждого города напишите запрос, выводящий максимальный рейтинг университетов, в нем расположенных, с последующей сортировкой списка по значениям рейтингов.
5. Для каждого дня сдачи экзаменов напишите запрос, выводящий среднее значение всех экзаменационных оценок.
6. Для каждого дня сдачи экзаменов напишите запрос, выводящий максимальные оценки, полученные по каждому предмету.
7. Для каждого дня сдачи экзаменов напишите запрос, выводящий общее количество студентов, сдававших экзамены.
8. Для каждого дня сдачи экзаменов напишите запрос, выводящий общее количество экзаменов, сдававшихся каждым студентом.
9. Для каждого преподавателя напишите запрос, выводящий коли- чество преподаваемых им предметов.
10. Для каждого предмета напишите запрос, выводящий количество преподавателей, ведущих по нему занятия.
11. Напишите запрос, выполняющий вывод количества студентов, имеющих только отличные оценки.
12. Напишите запрос, выполняющий вывод количества экзаменов, сданных (с положительной оценкой) студентом с идентификато- ром 32.

2.5. Неопределенные значения (NULL) в агрегирующих функциях

Отсутствие значений в полях таблицы, обозначенное ключевым словом **NULL**, приводит к особенностям при выполнении агрегирующих операций над данными, которые следует учитывать при их использо- вании в SQL-запросах.

* + 1. **Влияние NULL-значений в функции COUNT.** Если аргу- мент функции **COUNT** является константой или столбцом без пустых значений, то функция возвращает количество строк, к которым приме- нимо определенное условие или группирование.

Если аргументом функции является *столбец*, содержащий пустое значение, то **COUNT** вернет число строк, которые не содержат пустые значения и к которым применимо определенное условие или группиро- вание.

42 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



Если бы механизм **NULL** не был доступен, то неприменимые и отсутствующие значения пришлось бы исключать с помощью конструк- ции **WHERE**.

Поведение функции **COUNT**(\*) не зависит от пустых значений. Она возвратит общее количество строк в таблице.

* + 1. **Влияние NULL-значений в функции AVG.** *Среднее* значе- ние множества чисел равно сумме чисел, деленной на число элементов множества. Однако если некоторые элементы пусты, т. е. их значения неизвестны или не существуют, то деление на количество всех элемен- тов множества приведет к неправильному результату.

Функция **AVG** вычисляет среднее значение всех *известных* зна- чений множества элементов, т. е. эта функция подсчитывает сумму *известных* значений и делит ее на количество *этих* значений, а не на общее количество значений, среди которых могут быть **NULL**-значения. Если столбец состоит только из пустых значений, то функция **AVG** также возвратит **NULL**.

* 1. Результат действия трехзначных условных операторов

Условные операторы при отсутствии пустых значений возвраща- ют либо **TRUE** (ИСТИНА), либо **FALSE** (ЛОЖЬ). Если же в столбце присутствуют пустые значения, то может быть возвращено и третье значение: **UNKNOWN** (НЕИЗВЕСТНО). В этой схеме, например, условию **WHERE** A=2, где А — имя столбца, значения которого могут быть неизвестны, при A=2 будет соответствовать **TRUE**, при A=4 в результате будет получено значение **FALSE**, а при отсутствующем значении A (**NULL**-значение) результат будет **UNKNOWN**. Пустые значения оказыва- ют влияние на использование логических операторов **NOT**, **AND** и **OR**.

Оператор NOT

Обычный унарный оператор **NOT** обращает оценку **TRUE** в **FALSE** и наоборот. Однако **NOT**, примененный к неизвестному значению **UNKNOWN**, будет возвращать **UNKNOWN**. При этом следует отличать случай **NOT UNKNOWN** от условия **IS NOT NULL**, которое является про- тивоположностью выражения **IS NULL**, отделяя известные значения от неизвестных.

Оператор AND

Если результат двух условий, объединенных оператором **AND**, известен, то применяются правила булевой логики, т. е. при обоих утверждениях **TRUE** составное утверждение также будет **TRUE**. Если же хотя бы одно из двух утверждений будет **FALSE**, то составное утверждение будет **FALSE**.

*•*

Если результат одного из утверждений неизвестен, а другой оценивается как **TRUE**, то состояние неизвестного утверждения

*•*

* 1. *Упорядочение выходных полей (****ORDER BY****)* 43



является определяющим, и, следовательно, итоговый результат также неизвестен.

Если результат одного из утверждений неизвестен, а другой оце- нивается как **FALSE**, то итоговый результат будет **FALSE**.

*•*

Если результат обоих утверждений неизвестен, то результат так- же остается неизвестным.

*•*

Оператор OR

Если результат двух условий, объединенных оператором **OR**, из- вестен, то применяются правила булевой логики, а именно: если хотя бы одно из двух утверждений соответствует **TRUE**, то и составное утверждение будет **TRUE**; если оба утверждения оце- ниваются как **FALSE**, то и составное утверждение будет **FALSE**. Если результат одного из утверждений неизвестен, а другой оце- нивается как **TRUE**, итоговый результат будет **TRUE**.

*•*

*•*

Если результат одного из утверждений неизвестен, а другой оце- нивается как **FALSE**, то состояние неизвестного утверждения иг- рает роль. Следовательно, итоговый результат также неизвестен. Если результат обоих утверждений неизвестен, то результат так- же остается неизвестным.

*•*

*•*

**Примечание.** Отсутствующие (**NULL**) значения целесообразно ис- пользовать в столбцах, предназначенных для агрегирования, чтобы из- влечь преимущества из способа обработки пустых значений в функци- ях **COUNT** и **AVG**. Так как при наличии **NULL** существенно усложняется корректное построение условий отбора, приводя иногда к неожиданным для составителя запроса результатам выборки, то во всех остальных случаях следует по возможности избегать использования пустых значе- ний. В частности, для индикации отсутствующих, неприменимых или по какой-то причине неизвестных данных можно использовать значе- ния по умолчанию, устанавливаемые заранее (например, с помощью команды **CREATE TABLE** (раздел 4.1).

* 1. Упорядочение выходных полей (ORDER BY)

Как уже отмечалось, записи в таблицах реляционной базы данных неупорядочены. Однако, данные, выводимые в результате выполнения запроса, могут быть упорядочены. Для этого в операторе **SELECT** используется предложение с ключевым словом **ORDER BY**, которое позволяет упорядочивать выводимые записи в соответствии со значе- ниями одного или нескольких выбранных столбцов. При этом можно задать возрастающую (**ASC**) или убывающую (**DESC**) последователь- ность сортировки для каждого из столбцов. По умолчанию принята возрастающая последовательность сортировки.

Запрос, позволяющий выбрать все данные из таблицы предметов обучения SUBJECT, с упорядочением по наименованиям предметов, выглядит следующим образом:

44 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



SELECT \*

**FROM** SUBJECT

**ORDER BY** SUBJ\_NAME;

Тот же список, но упорядоченный в обратном порядке, можно получить запросом:

SELECT \*

**FROM** SUBJECT

**ORDER BY** SUBJ\_NAME **DESC**;

Можно упорядочить выводимый список предметов обучения по зна- чениям семестров, а внутри семестров — по наименованиям предметов:

SELECT \*

**FROM** SUBJECT

**ORDER BY** SEMESTR, SUBJ\_NAME;

Предложение **ORDER BY** может использоваться с **GROUP BY**. При этом оператор **ORDER BY** в запросе *всегда должен быть* ***последним***:

**SELECT** SUBJ\_NAME, SEMESTR, **MAX**(HOUR) **FROM** SUBJECT

**GROUP BY** SEMESTR, SUBJ\_NAME

**ORDER BY** SEMESTR;

При упорядочивании вместо наименований столбцов можно указы- вать их номера, имея, однако, в виду, что в данном случае это номера столбцов, указанные при определении выходных данных в запросе, а не номера столбцов в таблице. Полем с номером 1 является первое поле, указанное в предложении **ORDER BY** — независимо от его рас- положения в таблице. В запросе

**SELECT** SUBJ\_ID, SEMESTR

**FROM** SUBJECT

**ORDER BY** 2 **DESC**;

выводимые записи будут упорядочены по полю SEMESTR.

Если в поле, которое используется для упорядочивания, существу- ют **NULL**-значения, то все они размещаются в конце или предшествуют всем остальным значениям этого поля.

УПРАЖНЕНИЯ

1. Предположим, что стипендия всем студентам увеличена на 20%. Напишите запрос к таблице STUDENT, выполняющий вывод но- мера студента, его фамилии и величины увеличенной стипен- дии. Выходные данные упорядочите: а) по значению последнего столбца (величине стипендии); б) в алфавитном порядке фамилий студентов.
   1. *Вложенные подзапросы* 45



1. Напишите запрос, который по таблице EXAM\_MARKS позволяет найти а) максимальные и б) минимальные оценки каждого сту- дента и выводит их вместе с идентификатором студента.
2. Напишите запрос, выполняющий вывод списка предметов обуче- ния в порядке а) убывания семестров и б) возрастания количества отводимых на предмет часов. Поле семестра в выходных данных должно быть первым, за ним должны следовать имя предмета обучения и идентификатор предмета.
3. Напишите запрос, который для каждой даты сдачи экзаменов выполняет вывод суммы баллов всех студентов и представляет результаты в порядке убывания этих сумм.
4. Напишите запрос, который для каждой даты сдачи экзаменов выполняет вывод а) среднего, б) минимального, в) максимально- го баллов всех студентов и представляет результаты в порядке убывания этих значений.
   1. Вложенные подзапросы

SQL позволяет использовать одни запросы внутри других запросов, т. е. вкладывать запросы друг в друга. Предположим, известна фами- лия студента (Петров), но неизвестно значение поля STUDENT\_ID для него. Чтобы извлечь данные обо всех оценках этого студента, можно записать следующий запрос:

SELECT \*

**FROM** EXAM\_MARKS

**WHERE** STUDENT\_ID = (**SELECT** STUDENT\_ID

**FROM** STUDENT SURNAME = ‘Петров’);

Следует обратить внимание, что этот корректен только в том слу- чае, если в результате выполнения указанного в скобках *подзапроса* возвращается *единственное значение*. Если в результате выполнения подзапроса будет возвращено несколько значений, то при выполнении запроса будет зафиксирована ошибка. В данном примере это произой- дет, если в таблице STUDENT будет несколько записей со значениями поля SURNAME = ‘Петров’.

В некоторых случаях для гарантии получения единственного зна- чения в результате выполнения подзапроса используется **DISTINCT**. Одним из видов функций, которые автоматически *всегда* выдают в результате единственное значение для любого количества строк, явля- ются агрегирующие функции.

Оператор **IN** также широко применяется в подзапросах. Он задает список значений, с которыми сравниваются другие значения для опре- деления истинности задаваемого этим оператором предиката.

Данные обо всех оценках (таблица EXAM\_MARKS) студентов из Воронежа можно выбрать с помощью следующего запроса:

46 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



SELECT \*

**FROM** EXAM\_MARKS

**WHERE** STUDENT\_ID **IN**

(**SELECT** STUDENT\_ID

**FROM** STUDENT

**WHERE** CITY = ‘Воронеж’);

Подзапросы можно применять внутри предложения **HAVING**. Пусть требуется определить количество предметов обучения с оценкой, пре- вышающей среднее значение оценки студента с идентификатором 301:

**SELECT COUNT**(**DISTINCT** SUBJ\_ID), MARK

**FROM** EXAM\_MARKS **GROUP BY** MARK **HAVING** MARK *>*

(**SELECT AVG**(MARK) **FROM** EXAM\_MARKS

**WHERE** STUDENT\_ID = 301);

УПРАЖНЕНИЯ

1. Напишите запрос, выводящий список студентов, получающих максимальную стипендию, отсортировав его в алфавитном поряд- ке по фамилиям.
2. Напишите запрос, выводящий список студентов, получающих стипендию, превышающую среднее значение стипендии.
3. Напишите запрос, выводящий список студентов, обучающихся в Воронеже, с последующей сортировкой по идентификаторам университетов и курсам.
4. Напишите запрос, выводящий список предметов, на изучение которых отведено максимальное количество часов.
5. Напишите запрос, выполняющий вывод имен и фамилий студен- тов, место проживания которых не совпадает с городом, в котором находится их университет.
6. Напишите запрос, выводящий список университетов, располо- женных в Москве и имеющих рейтинг меньший, чем у ВГУ.
   1. Формирование связанных подзапросов

При использовании подзапросов во внутреннем запросе можно ссы- латься на таблицу, имя которой указано в предложении **FROM** внешнего запроса. Такие подзапросы называются связанными.

Связанный подзапрос выполняется по одному разу для каждой строки таблицы основного запроса, а именно:

выбирается строка из таблицы, имя которой указано во внешнем запросе;

*•*

* 1. *Формирование связанных подзапросов* 47



выполняется подзапрос, и полученное в результате его выполне- ния значение применяется для анализа этой строки в условии предложения **WHERE** внешнего запроса;

*•*

по результату оценки этого условия принимается решение о включении или невключении строки в состав выходных данных; процедура повторяется для следующей строки таблицы внешнего запроса.

*•*

*•*

Пример.

Выбрать сведения обо всех предметах обучения, по которым прово- дился экзамен 20 января 2005 г.

SELECT \*

**FROM** SUBJECT SU

**WHERE** ‘20/01/2005’ **IN**

(**SELECT** EXAM\_DATE

**FROM** EXAM\_MARKS EX

**WHERE** SU.SUBJ\_ID = EX.SUBJ\_ID);

В некоторых СУБД для выполнения этого запроса, возможно, по- требуется преобразование значения даты в символьный тип. В при- веденном запросе SU и EX являются псевдонимами (алиасами), т. е. специально вводимыми именами, которые могут быть использованы в данном запросе вместо настоящих имен. В приведенном примере они используются вместо имен таблиц SUBJECT и EXAM\_MARKS.

Эту же задачу можно решить с помощью операции соединения таблиц:

**SELECT DISTINCT** FIRST.SUBJ\_ID, SUBJ\_NAME,

HOUR, SEMESTER

**FROM** SUBJECT FIRST, EXAM\_MARKS SECOND **WHERE** FIRST.SUBJ\_ID = SECOND.SUBJ\_ID **AND** SECOND.EXAM\_DATE = ‘20/01/2005’;

В этом выражении алиасами таблиц являются имена FIRST и

SECOND.

Можно использовать подзапросы, связывающие таблицу со своей собственной копией. Например, надо найти идентификаторы, фамилии и стипендии студентов, получающих стипендию выше средней на кур- се, на котором они учатся:

**SELECT DISTINCT** STUDENT\_ID, SURNAME, STIPEND

**FROM** STUDENT E1

**WHERE** STIPEND >

(**SELECT AVG**(STIPEND) **FROM** STUDENT E2

**WHERE** E1.KURS = E2.KURS);

Тот же результат можно получить с помощью следующего запроса:

48 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



**SELECT DISTINCT** STUDENT\_ID, SURNAME, STIPEND

**FROM** STUDENT E1,

(**SELECT** KURS, **AVG**(STIPEND) **AS** AVG\_STIPEND

**FROM** STUDENT E2

**GROUP BY** E2.KURS) E3

**WHERE** E1.STIPEND > AVG\_STIPEND **AND**

E1.KURS=E3.KURS;

Обратите внимание — второй запрос должен выполниться гораздо быстрее. Дело в том, что в первом варианте запроса агрегирующая функция **AVG** выполняется над таблицей, указанной в подзапросе, для *каждой* строки внешнего запроса. В другом варианте вторая таб- лица (алиас E2) обрабатывается агрегирующей функцией один раз, в результате чего формируется вспомогательная таблица (в запросе она имеет алиас E3), со строками которой затем соединяются строки первой таблицы (алиас E1). Следует иметь в виду, что на самом деле реальное время выполнения запроса в большой степени зависит от оптимизатора запросов конкретной СУБД, и, вполне возможно, что такое преобразование запроса будет выполнено оптимизатором.

УПРАЖНЕНИЯ

1. Напишите запрос для получения списка студентов, которые учат- ся в своем городе.
2. Напишите запрос для получения списка иногородних студентов (обучающихся не в своем городе), с последующей сортировкой по идентификаторам университетов и курсам.
3. Напишите запрос для получения списка преподавателей, работа- ющих не в своем городе, с последующей сортировкой по иденти- фикаторам университетов и городам проживания преподавателей.
4. Напишите запрос для получения списка предметов, на изучение которых отведено максимальное количество часов среди всех предметов, изучаемых в том же семестре. Список упорядочить по семестрам.
5. Напишите запрос для получения списка студентов, получающих стипендию, превосходящую среднее значение стипендии на их курсе.
6. Напишите запрос для получения списка студентов, получающих минимальную стипендию в своем университете, с последующей сортировкой по значениям идентификатора университета и сти- пендии.
7. Напишите запрос для получения списка университетов, в кото- рых учится более 50 студентов, с последующей сортировкой по рейтингам.
8. Напишите запрос для получения списка университетов, в кото- рых работает более 5 преподавателей, с последующей сортиров- кой по рейтингам университетов.
   1. *Связанные подзапросы в* ***HAVING*** 49



1. Напишите запрос для получения списка отличников (студентов, получивших только отличные оценки), с последующей сортиров- кой по идентификаторам университетов и курсам.
2. Напишите запрос для получения списка неуспевающих студен- тов (получивших хотя бы одну неудовлетворительную оценку), с последующей сортировкой по идентификаторам университетов и курсам.
3. Напишите запрос, выполняющий вывод списка студентов, сред- няя оценка которых превышает 4 балла.
4. Напишите запрос, выполняющий вывод списка фамилий студен- тов, имеющих только отличные оценки и проживающих в городе, не совпадающем с городом их университета.
   1. Связанные подзапросы в HAVING

В разделе 2.4 указывалось, что предложение **GROUP BY** позво- ляет группировать выводимые **SELECT**-запросом записи по значению некоторого поля. Использование предложения **HAVING** позволяет при выводе осуществлять фильтрацию таких групп. Предикат предложения **HAVING** оценивается не для каждой строки результата, а для каждой группы выходных записей, сформированной предложением **GROUP BY** внешнего запроса.

Пусть, например, необходимо по данным из таблицы EXAM\_MARKS определить сумму полученных студентами оценок (значений поля MARK), сгруппировав значения оценок по датам экзаменов и исключив те дни, когда число студентов, сдававших в течение дня экзамены, было меньше 10:

**SELECT** EXAM\_DATE, **SUM**(MARK) **FROM** EXAM\_MARKS A

**GROUP BY** EXAM\_DATE

**HAVING** 10 <

(**SELECT COUNT**(MARK) **FROM** EXAM\_MARKS B

**WHERE** A.EXAM\_DATE = B.EXAM\_DATE);

Подзапрос вычисляет количество строк, у которых значения поля EXAM\_DATE (дата экзамена) совпадает с датой, для которой сформиро- вана очередная группа основного запроса.

УПРАЖНЕНИЯ

1. Напишите запрос с подзапросом для получения всех оценок сту- дента с фамилией Иванов. Предположим, что его персональный номер неизвестен. Всегда ли такой запрос будет корректным?
2. Напишите запрос, выбирающий имена всех студентов, имеющих по предмету c идентификатором 101 балл выше общего среднего балла.

50 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



1. Напишите запрос, который выполняет выборку имен всех студен- тов, имеющих по предмету c идентификатором 102 балл ниже общего среднего балла.
2. Напишите запрос, выполняющий вывод количества предметов, по которым экзаменовался каждый студент, сдававший более 20 предметов.
3. Напишите команду **SELECT**, использующую связанные подзапро- сы и выполняющую вывод имен и идентификаторов студентов, у которых стипендия совпадает с максимальным значением стипен- дии для города, в котором живет студент.
4. Напишите запрос, который позволяет вывести имена и иденти- фикаторы всех студентов, о которых точно известно, что они проживают в городе, где нет ни одного университета.
5. Напишите два запроса, которые позволяют вывести имена и идентификаторы всех студентов, о которых точно известно, что они проживают не в том городе, где расположен их университет: один запрос с использованием связанного подзапроса, а другой — с использованием соединения.
   1. Использование оператора EXISTS

Используемое в SQL ключевое слово **EXISTS** (СУЩЕСТВУЕТ) пред- ставляет собой предикат, принимающий значение *истина* или *ложь*. Используя подзапросы в качестве аргумента, этот оператор оценивает результат выполнения подзапроса как истинный, если этот подзапрос генерирует выходные данные, то есть в случае *существования* (возвра- та) хотя бы одного найденного значения. В противном случае результат подзапроса — ложный. Оператор **EXISTS** не может принимать значе- ние *неизвестно*).

Пусть, например, нужно извлечь из таблицы EXAM\_MARKS данные о студентах, получивших хотя бы одну неудовлетворительную оценку:

**SELECT DISTINCT** STUDENT\_ID

**FROM** EXAM\_MARKS A

WHERE EXISTS

(**SELECT** \*

**FROM** EXAM\_MARKS B

**WHERE** MARK < 3

**AND** B.STUDENT\_ID=A.STUDENT\_ID);

При использовании связанных подзапросов предложение **EXISTS** анализирует каждую строку таблицы, на которую имеется ссылка во внешнем запросе. Главный запрос получает строки-кандидаты на про- верку условия. Для каждой строки-кандидата выполняется подзапрос. Как только при выполнении подзапроса находится строка, в которой значение в столбце MARK удовлетворяет условию, его выполнение пре-

* 1. *Использование оператора* ***EXISTS*** 51



кращается и возвращается значение *истина* внешнему запросу, кото- рый затем анализирует свою строку-кандидата.

Например, требуется получить идентификаторы предметов обуче- ния, экзамены по которым сдавались не одним, а несколькими студен- тами:

**SELECT DISTINCT** SUBJ\_ID **FROM** EXAM\_MARKS A **WHERE EXISTS**

(**SELECT** \*

**FROM** EXAM\_MARKS B

**WHERE** A.SUBJ\_ID = B.SUBJ\_ID

**AND** A.STUDENT\_ID <> B.STUDENT\_ID);

Часто **EXISTS** применяется с оператором **NOT** (по-русски **NOT EXISTS** интерпретируется, как “*не существует...*”). Если предыду- щий запрос сформулировать следующим образом: “Найти идентифи- каторы предметов обучения, которые сдавались одним и только од- ним студентом (другими словами, для которых не существует другого сдававшего студента),” то достаточно просто поставить **NOT** перед **EXISTS**.

Возможности применения вложенных запросов весьма разнообраз- ны. Например, пусть из таблицы STUDENT требуется извлечь строки для каждого студента, сдавшего более одного предмета:

SELECT \*

**FROM** STUDENT FIRST **WHERE EXISTS** (**SELECT** SUBJ\_ID

**FROM** EXAM\_MARKS SECOND

**GROUP BY** SUBJ\_ID

**HAVING COUNT**(SUBJ\_ID) > 1

**WHERE** FIRST.STUDENT\_ID = SECOND.STUDENT\_ID);

УПРАЖНЕНИЯ

1. Напишите запрос с **EXISTS**, позволяющий вывести данные обо всех студентах обучающихся в вузах, имеющих рейтинг выше 300.
2. Напишите предыдущий запрос, используя соединения.
3. Напишите запрос с **EXISTS**, выбирающий сведения о каждом студенте, для которого в городе его проживания имеется хотя бы один университет, в котором он не учится.
4. Напишите запрос, выбирающий из таблицы SUBJECT данные о названиях предметов обучения, экзамены по которым сданы более чем одним студентом.
5. Напишите запрос для получения списка городов проживания студентов, в которых есть хотя бы один университет.

52 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



1. Напишите запрос для получения списка городов проживания студентов, в которых нет ни одного университета.
2. Напишите запрос для получения списка предметов, для которых не назначено ни одного преподавателя.
3. Напишите запрос для получения списка предметов, изучаемых в течение одного семестра.
4. Напишите запрос для получения списка предметов, изучаемых более чем в одном семестре.
5. Напишите запрос для получения списка университетов, в кото- рых не работает ни один преподаватель.
6. Напишите запрос для получения списка университетов, в кото- рых не учится ни один студент.
7. Напишите запрос для получения списка предметов, по кото- рым на экзаменах не получено ни одной неудовлетворительной оценки.
8. Напишите запрос для получения списка предметов, по кото- рым на экзаменах получена хотя бы одна неудовлетворительная оценка.
9. Напишите запрос для получения списка студентов, не получив- ших на экзаменах ни одной неудовлетворительной оценки.
10. Напишите запрос для получения списка студентов, получивших на экзаменах хотя бы одну неудовлетворительную оценку.
11. Напишите запрос, выполняющий вывод имен и фамилий студен- тов, получивших хотя бы одну отличную оценку.
12. Напишите запрос, выполняющий вывод имен и фамилий студен- тов, не получивших ни одной отличной оценки.
13. Напишите запрос, выполняющий вывод количества студентов, не имеющих ни одной оценки.
14. Напишите запрос, выполняющий вывод количества студентов, имеющих только отличные оценки.
15. Напишите запрос, выполняющий вывод количества студентов, имеющих хотя бы одну неудовлетворительную оценку и прожи- вающих в городе, не совпадающем с городом их университета.
    1. Операторы сравнения с множеством значений

**IN, ANY, ALL**

Операторы сравнения с множеством значений имеют следующий смысл.

**IN** *Равно* хотя бы одному из значений, полученных во внутреннем запросе.

**NOT IN** *Не равно* ни одному из значений, полученных во внутреннем запросе.

=**ANY** То же, что и **IN**. Соответствует логическому опера- тору **OR**.

* 1. *Операторы сравнения с множеством значений* ***IN****,* ***ANY****,* ***ALL*** 53



>**ANY**, >=**ANY** *Больше* (либо *больше или равно*) хотя бы одного из полученных значений. Эквивалентно > (или >=) для наименьшего из полученных значений.

<**ANY**, <=**ANY** *Меньше* (либо *меньше или равно*) хотя бы одного из полученных значений. Эквивалент < (или <=) для наибольшего из полученных значений.

=**ALL** *Равно* каждому из полученных значений. Эквива- лентно логическому оператору **AND**.

>**ALL**, >=**ALL** *Больше* (либо *больше или равно*) каждого из по- лученных значений. Эквивалент > (или >=) для наибольшего из полученных значений.

<**ALL**, <=**ALL** *Меньше* (либо *меньше или равно*) каждого из по- лученных значений. Эквивалентно < (или <=) наи- меньшего из полученных значений.

Пример 1.

Выбрать сведения о студентах, обучающихся в университете, рас- положенном в городе их проживания.

SELECT \*

**FROM** STUDENT S

**WHERE** CITY = **ANY**

(**SELECT** CITY

**FROM** UNIVERSITY U

**WHERE** U.UNIV\_ID = S.UNIV\_ID);

Пример 2.

Другой вариант запроса из примера 1:

SELECT \*

**FROM** STUDENT S

**WHERE** CITY **IN**

(**SELECT** CITY

**FROM** UNIVERSITY U

**WHERE** U.UNIV\_ID = S.UNIV\_ID);

Пример 3.

Выборка данных об идентификаторах студентов, у которых оценки превосходят величину по крайней мере одной из оценок, полученных 6 октября 2005 г.:

**SELECT DISTINCT** STUDENT\_ID

**FROM** EXAM\_MARKS

**WHERE** MARK > **ANY**

(**SELECT** MARK

54 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



**FROM** EXAM\_MARKS

**WHERE** EXAM\_DATE = ‘06/10/2005’);

Оператор **ALL**, как правило, эффективно используется с неравен- ствами, а не с равенствами, поскольку значение *равно всем*, которое должно получиться в этом случае в результате выполнения подзапроса, может иметь место, только если все результаты идентичны. Такая ситуация практически не может быть реализована, так как если подза- прос генерирует множество различных значений, то никакое отдельное значение не может быть равно сразу всем значениям в обычном смысле. В SQL выражение <>**ALL** реально означает *не равно ни одному* из результатов подзапроса.

Пример 4.

Подзапрос, выбирающий данные о названиях всех университетов с рейтингом более высоким, чем рейтинг любого университета в Воро- неже:

SELECT \*

**FROM** UNIVERSITY

**WHERE** RATING > **ALL**

(**SELECT** RATING

**FROM** UNIVERSITY

**WHERE** CITY = ‘Воронеж’);

Пример 5.

В предыдущем примере вместо **ALL** можно также использовать **ANY** (проанализируйте, как в этом случае изменится смысл приведенного запроса?):

SELECT \*

**FROM** UNIVERSITY

**WHERE NOT** RATING > **ANY**

(**SELECT** RATING

**FROM** UNIVERSITY

**WHERE** CITY = ‘Воронеж’);

УПРАЖНЕНИЯ

1. Напишите запрос для получения списка названий предметов, изучаемых в нескольких семестрах.
2. Напишите запрос, выполняющий вывод имен и фамилий студен- тов, имеющих весь набор положительных (тройки, четверки и пятерки) оценок.
3. Напишите запрос, выполняющий выборку значений идентифика- торов студентов, имеющих такие же оценки, что и студент с идентификатором 12.
   1. *Особенности применения операторов* ***ANY****,* ***ALL****,* ***EXISTS****...* 55



1. Напишите запрос, выполняющий вывод количества студентов, не имеющих ни одной оценки.
2. Напишите запрос, выполняющий вывод данных о предметах обу- чения, которые преподает Колесников.
3. Напишите запрос, выполняющий вывод имен и фамилий препо- давателей, проводящих занятия на первом курсе.
4. Напишите запрос, выполняющий вывод данных о преподавате- лях, ведущих обучение хотя бы по одному из предметов обучения, которые преподает Сорокин.
5. Напишите запрос, выполняющий вывод списка фамилий студен- тов университета, расположенного в городе, название которого стоит первым в алфавитном списке городов.
6. Напишите запрос, выполняющий вывод списка фамилий студен- тов, имеющих только отличные оценки и проживающих в городе, не совпадающем с городом их университета.
   1. Особенности применения операторов ANY, ALL, EXISTS при обработке отсутствующих данных

Необходимо иметь в виду, что операторы **ANY** и **ALL** по-разному реагируют на ситуацию, когда правильный подзапрос не генерирует никаких выходных данных. В этом случае оператор **ALL** автоматически принимает значение *истина*, а оператор **ANY** — значение *ложь*.

В случае, когда в базе отсутствуют данные об университетах из города Саратова, запрос

SELECT \*

**FROM** UNIVERSITY

**WHERE** RATING > **ANY**

(**SELECT** RATING

**FROM** UNIVERSITY

**WHERE** CITY = ‘Саратов’);

не генерирует никаких выходных данных. В такой же ситуации запрос

SELECT \*

**FROM** UNIVERSITY

**WHERE** RATING > **ALL**

(**SELECT** RATING

**FROM** UNIVERSITY

**WHERE** CITY = ‘New York’);

полностью воспроизведет таблицу UNIVERSITY.

Использование **NULL**-значений также создает определенные пробле- мы для рассматриваемых операторов.

Рассмотрим в качестве примера две реализации запроса “Найти все данные об университетах, рейтинг которых меньше рейтинга любого университета в Москве”.

56 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



1. SELECT \*

**FROM** UNIVERSITY

**WHERE** RATING < **ANY**

(**SELECT** RATING

**FROM** UNIVERSITY

**WHERE** CITY = ‘Москва’);

1. SELECT \*

**FROM** UNIVERSITY A

WHERE NOT EXISTS

(**SELECT** \*

**FROM** UNIVERSITY B

**WHERE** A.RATING >= B.RATING

**AND** B.CITY = ‘Москва’);

При отсутствии в таблице **NULL**-значений оба эти запроса ведут себя совершенно одинаково.

Пусть теперь в таблице UNIVERSITY есть строка с **NULL**-значе- ниями в столбце RATING. В версии запроса c **ANY** в основном запросе, когда выбирается поле RATING с **NULL**, предикат принимает значе- ние *неизвестно* и строка, так же как и в случае, когда результатом сравнения будет *ложь*, не включается в состав выходных данных. Во втором же варианте запроса, когда в основном запросе выбирается строка с **NULL** в поле RATING, предикат, используемый в подзапро- се, примет значение *неизвестно*. Поэтому при выполнении подзапро- са не будет получено ни одной строки, в результате чего оператор **NOT EXISTS** примет значение *истина*, и, следовательно, данная стро- ка с **NULL**-значением попадет в выходные данные основного запроса. По смыслу этого запроса такой результат не является правильным, так как на самом деле рейтинг университета, описываемого данной строкой, может быть и больше рейтинга какого-либо московского уни- верситета (он просто неизвестен). Указанная проблема связана с тем, что значение **EXISTS** всегда принимает значения *истина* или *ложь*, и никогда — *неизвестно*. Это является доводом для использования в данном случае оператора **ANY** вместо **NOT EXISTS**.

УПРАЖНЕНИЕ

**1.** Ниже приведены два варианта запроса, выполняющего вывод количества студентов, имеющих только отличные оценки. Всегда ли эти запросы будут выдавать одинаковые результаты?

**SELECT COUNT**(**DISTINCT** STUDENT\_ID)

**FROM** EXAM\_MARKS S

WHERE NOT EXISTS

(**SELECT** \*

**FROM** EXAM\_MARKS

**WHERE** STUDENT\_ID=S.STUDENT\_ID **AND** MARK<5);

* 1. *Использование* ***COUNT*** *вместо* ***EXISTS*** 57



SELECT COUNT (\*)

**FROM**

(**SELECT** STUDENT\_ID, **MIN**(MARK) **FROM** EXAM\_MARKS

**GROUP BY** STUDENT\_ID

**HAVING MIN**(MARK)=5);

* 1. Использование COUNT вместо EXISTS

При отсутствии **NULL**-значений оператор **EXISTS** может быть ис- пользован вместо **ANY** и **ALL**. Также вместо **EXISTS** и **NOT EXISTS** могут быть использованы те же самые подзапросы, но с использовани- ем **COUNT**(\*) в предложении **SELECT**. Например, запрос

SELECT \*

**FROM** UNIVERSITY A

WHERE NOT EXISTS

(**SELECT** \*

**FROM** UNIVERSITY B

**WHERE** A.RATING >= B.RATING

**AND** B.CITY = ‘Москва’);

может быть представлен и в следующем виде:

SELECT \*

**FROM** UNIVERSITY A

**WHERE** 1 >

(**SELECT COUNT**(\*) **FROM** UNIVERSITY B

**WHERE** A.RATING >= B.RATING

**AND** B.CITY = ‘Москва’);

УПРАЖНЕНИЯ

1. Напишите запрос, выбирающий данные о названиях университе- тов, рейтинг которых равен или превосходит рейтинг ВГУ.
2. Напишите запрос с использованием **ANY** или **ALL**, выполняющий выборку данных о студентах, у которых в городе их постоянного местожительства нет университета.
3. Напишите запрос, выбирающий из таблицы EXAM\_MARKS назва- ния предметов обучения, для которых все оценки (поле MARK) превышают любые оценки по предмету, имеющему идентифика- тор 105.
4. Напишите этот же запрос с использованием **MAX**.
   1. Соединение таблиц. Оператор JOIN

Если в операторе **SELECT** после ключевого слова **FROM** указывается не одна, а две таблицы, то в результате выполнения запроса, в котором отсутствует предложение **WHERE**, каждая строка одной таблицы будет соединена с каждой строкой второй таблицы. Такая операция назы- вается *декартовым произведением*, или *полным соединением* таблиц базы данных. Сама по себе эта операция не имеет практического зна- чения, более того, при ошибочном использовании она может привести к неожиданным нештатным ситуациям, так как в этом случае в ответе на запрос количество записей будет равно произведению числа записей в соединяемых таблицах, т. е. может оказаться чрезвычайно большим. Соединение таблиц имеет смысл тогда, когда соединяются *не все* стро- ки исходных таблиц, а только те, которые интересуют пользователя. Такое ограничение может быть осуществлено с помощью использова- ния в запросе соответствующего условия в предложении **WHERE**. Таким образом, SQL позволяет выводить информацию из нескольких таблиц, связывая их по значениям определенных полей.

Например, если необходимо получить фамилии студентов (таблица STUDENT) и для каждого студента — названия университетов (таблица UNIVERSITY), расположенных в городе, где живет студент, то необхо- димо получить все комбинации записей о студентах и университетах в обеих таблицах, в которых значение поля CITY совпадает. Это можно сделать с помощью следующего запроса:

**SELECT** STUDENT.SURNAME, UNIVERSITY.UNIV\_NAME, STUDENT.CITY

**FROM** STUDENT, UNIVERSITY

**WHERE** STUDENT.CITY = UNIVERSITY.CITY;

Соединение, использующее предикаты, основанные на равенствах, называется *эквисоединением*. Рассмотренный пример соединения таб- лиц относятся к виду так называемого *внутреннего* (**INNER**) *соедине- ния*. При таком типе соединения соединяются только те строки таблиц, для которых является истинным предикат, задаваемый в предложении **ON** выполняемого запроса.

Приведенный выше запрос может быть записан иначе, с использо- ванием ключевого слова **JOIN**:

**SELECT** STUDENT.SURNAME, UNIVERSITY.UNIV\_NAME, STUDENT.CITY

**FROM** STUDENT **INNER JOIN** UNIVERSITY

**ON** STUDENT.CITY = UNIVERSITY.CITY;

Ключевое слово **INNER** в запросе может быть опущено, так как эта опция в операторе **JOIN** действует по умолчанию.

УПРАЖНЕНИЯ

1. Напишите запрос для получения списка предметов вместе с фа- милиями студентов, изучающих их на соответствующем курсе.
2. Напишите запрос, выполняющий вывод имен и фамилий студен- тов, имеющих весь набор положительных (тройки, четверки и пятерки) оценок.
   * 1. **Операции соединения таблиц посредством ссылочной целостности.** Информация в таблицах STUDENT и EXAM\_MARKS уже связана посредством поля STUDENT\_ID. В таблице STUDENT поле STUDENT\_ID является первичным ключом, а в таблице EXAM\_MARKS — ссылающимся на него внешним ключом. Состояние связанных таким образом таблиц называется состоянием ссылочной целостности. В данном случае ссылочная целостность этих таблиц подразумевает, что *каждому* значению поля STUDENT\_ID в таблице EXAM\_MARKS *обязательно* соответствует *такое же значение* поля STUDENT\_ID в таблице STUDENT. Другими словами, в таблице EXAM\_MARKS не может быть записей, имеющих идентификаторы студентов, которых нет в таблице STUDENT. Стандартное применение операции соединения состоит в извлечении данных в терминах этой связи.

Чтобы получить список фамилий студентов с полученными ими оценками и идентификаторами предметов, можно использовать следу- ющий запрос:

**SELECT** SURNAME, MARK, SUBJ\_ID **FROM** STUDENT, EXAM\_MARKS **WHERE** STUDENT.STUDENT\_ID =

EXAM\_MARKS.STUDENT\_ID;

Тот же самый результат может быть получен при использовании в запросе для задания операции соединения таблиц ключевого слова **JOIN**. Запрос с оператором **JOIN** выглядит следующим образом:

**SELECT** SURNAME, MARK

**FROM** STUDENT **JOIN** EXAM\_MARKS

**ON** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID;

Для такого рода запросов, когда соединение таблиц осуществляется по одноименным столбцам, можно использовать так называемое есте- ственное соединение, задаваемое ключевым словом **NATURAL**. В этом случае в запросе не указывается предложение **ON** условия отбора запи- сей. Приведенный выше запрос будет выглядеть следующим образом:

**SELECT** SURNAME, MARK

**FROM** STUDENT **NATURAL JOIN** EXAM\_MARKS;

Хотя выше речь шла о соединении двух таблиц, можно сформиро- вать запросы путем соединения более чем двух таблиц.

Пусть требуется найти фамилии всех студентов, получивших неудо- влетворительную оценку, вместе с названиями предметов обучения, по которым получена эта оценка.

**SELECT** SUBJ\_NAME, SURNAME, MARK

**FROM** STUDENT, SUBJECT, EXAM\_MARKS

**WHERE** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID

**AND** SUBJECT.SUBJ\_ID = EXAM\_MARKS.SUBJ\_ID

**AND** EXAM\_MARKS.MARK = 2;

То же самое с использованием оператора **JOIN**: **SELECT** SUBJ\_NAME, SURNAME, MARK

**FROM** STUDENT **JOIN** EXAM\_MARKS

**ON** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID

**JOIN** SUBJECT

**ON** SUBJECT.SUBJ\_ID = EXAM\_MARKS.SUBJ\_ID

**WHERE** MARK = 2;

УПРАЖНЕНИЯ

1. Напишите запрос для получения списка университетов с указа- нием количества студентов, обучающихся на каждом курсе.
2. Напишите запрос для получения списка преподавателей с указа- нием их учебных предметов.
3. Напишите запрос для получения списка преподавателей с указа- нием нагрузки (суммарного количества часов) в каждом семестре.
4. Напишите запрос для получения списка университетов вместе с названиями преподаваемых в них предметов.
5. Напишите запрос для получения списка университетов с ука- занием суммарного количества аудиторных часов в каждом семестре.
6. Напишите запрос для получения списка университетов с ука- занием суммарного количества часов, отводимых на изучение каждого предмета.
7. Напишите запрос для получения списка преподавателей с ука- занием суммарного количества часов, отведенных для обучения каждому из предметов.
8. Напишите запрос для сортировки списка университетов по зна- чениям максимальной стипендии, выплачиваемой студентам.
9. Напишите запрос для получения списка университетов вместе с фамилиями самых молодых студентов, обучаемых в них.
10. Напишите запрос для получения списка университетов вместе с фамилиями студентов, получающих максимальную для каждого университета стипендию.
11. Напишите запрос для получения списка студентов вместе с на- званиями предметов и оценками, полученными по каждому пред- мету на экзаменах.
12. Напишите запрос для получения списка предметов вместе с фамилиями студентов, получивших по данному предмету макси- мальную оценку.
13. Напишите запрос для получения списка предметов вместе с фамилиями студентов, сдававших экзамен по данному предмету последними.
14. Напишите запрос для получения списка предметов вместе с фамилиями студентов, первыми сдавших экзамен по данному предмету.
15. Напишите запрос для получения списка преподавателей, препо- дающих более одного предмета.
16. Напишите запрос для получения списка преподавателей, препо- дающих только один предмет.
17. Напишите запрос для получения списка студентов, сдававших экзамены по какому-либо из предметов более одного раза.
18. Напишите запрос для получения списка университетов вместе с фамилиями студентов, получивших хотя бы одну неудовлетвори- тельную оценку.
19. Напишите запрос, выполняющий вывод имен и фамилий студен- тов, получивших хотя бы одну отличную оценку.
20. Напишите запрос, выполняющий вывод данных о предметах обу- чения, которые преподает Колесников.
21. Напишите запрос, выполняющий вывод имен и фамилий препо- давателей, проводящих занятия на первом курсе.
22. Напишите запрос, выполняющий вывод имен и фамилий препо- давателей, проводящих занятия в двух и более семестрах.
23. Напишите запрос, выполняющий вывод наименований предметов обучения, читаемых двумя или более преподавателями.
24. Напишите запрос, выполняющий вывод количества часов заня- тий, проводимых преподавателем Лагутиным.
25. Напишите запрос, выполняющий вывод фамилий преподавателей, учебная нагрузка которых (количество учебных часов) превышает нагрузку преподавателя Николаева.
26. Напишите запрос, выполняющий вывод фамилий преподавателей университетов с рейтингом, меньшим 200.
27. Напишите запрос, выполняющий вывод общего количества учеб- ных часов занятий, проводимых для студентов первого курса ВГУ.
28. Напишите запрос, выполняющий вывод среднего количества учебных часов предметов обучения, преподаваемых студентам второго курса ВГУ.
29. Напишите запрос, выполняющий вывод списка фамилий студен- тов, имеющих две или более отличных оценок в каждом семестре.
30. Приведите как можно больше формулировок запроса “Получить фамилии студентов, сдававших экзамен по информатике”.
31. Приведите как можно больше формулировок запроса “Получить фамилии преподавателей, преподающих информатику”.
    * 1. **Внешнее соединение таблиц.** Как отмечалось ранее, при использовании *внутреннего* (**INNER**) соединения таблиц соединяются только те их строки, в которых совпадают значения полей, задаваемые в предложении **WHERE** запроса. Однако во многих случаях это может привести к нежелательной потере информации. Рассмотрим еще раз приведенный выше пример запроса на выборку списка фамилий сту- дентов с полученными ими оценками и идентификаторами предметов. При использовании, как это было сделано в рассматриваемом примере, внутреннего соединения в результат запроса не попадут студенты, которые еще не сдавали экзамены и которые, следовательно, отсут- ствуют в таблице EXAM\_MARKS. Если же необходимо иметь записи об этих студентах в выдаваемом запросом списке, то можно присоединить сведения о студентах, не сдававших экзамен, путем использования оператора **UNION** с соответствующим запросом. Например, следующим образом:

**SELECT** SURNAME, **CAST** MARK **AS** CHAR(1),

**CAST** SUBJ\_ID **AS** CHAR(10)

**FROM** STUDENT, EXAM\_MARKS

**WHERE** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID

UNION

**SELECT** SURNAME, **CAST NULL AS** CHAR(1),

**CAST NULL AS** CHAR(10)

**FROM** STUDENT

WHERE NOT EXIST

(**SELECT** \*

**FROM** EXAM\_MARKS

**WHERE** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID);

(здесь функция преобразования типов **CAST** используется для обеспе- чения совместимости типов полей объединяемых запросов).

Нужный результат, однако, может быть получен и путем исполь- зования оператора *внешнего соединения*, точнее, одной из его разно- видностей — *левого внешнего соединения* **LEFT OUTER JOIN**. В этом случае запрос будет выглядеть следующим образом:

**SELECT** SURNAME, MARK

**FROM** STUDENT **LEFT OUTER JOIN** EXAM\_MARKS

**ON** STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID;

При использовании *левого* соединения расширение выводимой таб- лицы осуществляется за счет записей входной таблицы, имя которой указано *слева* от оператора **JOIN**.

Следует заметить, что в СУБД Oracle для обозначения внешних соединений наряду со стандартной может использоваться и другая но-

тация. Например, приведенный выше запрос может иметь следующий вид:

**SELECT** SURNAME, MARK, SUBJ\_ID **FROM** STUDENT, EXAM\_MARKS **WHERE** STUDENT.STUDENT\_ID =

EXAM\_MARKS.STUDENT\_ID(**+**);

Знак (**+**) ставится у той таблицы, которая дополняется записями с **NULL**-значениями, чтобы при соединении таблиц в выходное отно- шение попали и те записи другой таблицы, для которых в таблице со знаком (**+**) не находится строк с соответствующими значениями атрибутов, используемых для соединения, т. е. для *левого* внешнего соединения в запросе Oracle-SQL указатель (**+**) ставится у *правой* таблицы.

Приведенный выше запрос может быть реализован и с применением

*правого внешнего соединения*. Он будет иметь следующий вид:

**SELECT** SURNAME, MARK

**FROM** EXAM\_MARKS **RIGHT OUTER JOIN** STUDENT

**ON** EXAM\_MARKS.STUDENT\_ID = STUDENT.STUDENT\_ID;

Здесь таблица STUDENT, за счет записей которой осуществляется расширение выводимой таблицы, стоит справа от оператора **JOIN**.

В нотации, допустимой в СУБД Oracle, этот запрос может выгля- деть следующим образом:

**SELECT** SURNAME, MARK, SUBJ\_ID

**FROM** STUDENT, EXAM\_MARKS

**WHERE** EXAM\_MARKS.STUDENT\_ID(**+**) = STUDENT.STUDENT\_ID;

Видно, что использование правого или левого внешнего соединения позволяет существенно упростить запрос, сделать его запись более компактной.

Иногда возникает необходимость включения в результат запроса записей из обеих (правой и левой) соединяемых таблиц, для которых не удовлетворяется условие соединения. Такое соединение называется *полным внешним соединением* и осуществляется указанием в запросе ключевых слов **FULL OUTER JOIN**.

УПРАЖНЕНИЯ

1. Напишите запрос, который выполняет вывод фамилий студентов, *сдававших* экзамены, вместе с идентификаторами каждого сдан- ного ими предмета обучения.
2. Напишите запрос, который выполняет выборку фамилий *всех* студентов, с указанием для студентов, сдававших экзамены, идентификаторов сданных ими предметов обучения.
3. Напишите запрос, который выполняет вывод фамилий студентов, *сдававших* экзамены, вместе с наименованиями каждого сданно- го ими предмета обучения.
4. Напишите запрос на выдачу списка *всех* студентов. Для студен- тов, сдававших экзамены, укажите названия соответствующих предметов обучения.
5. Напишите запрос на выдачу названий всех предметов, по ко- торым студенты получили только хорошие (4 и 5) оценки. В выходных данных должны быть приведены фамилии студентов, названия предметов и оценка.
6. Напишите запрос, который выполняет вывод списка университе- тов с рейтингом, превышающим 300, вместе со значением мак- симального размера стипендии, получаемой студентами в этих университетах.
7. Напишите запрос на выдачу списка студентов (в алфавитном порядке фамилий) вместе со значением рейтинга университета, где каждый из них учится, включив в список и тех студентов, место учебы которых в базе данных не указано.
8. Напишите запрос для получения списка всех студентов вместе с названиями университетов, в которых они учатся. Отдельным запросом получите записи, расширяющие данный список по срав- нению с тем, который был бы получен внутренним соединением.
9. Напишите запрос для получения списка всех университетов вме- сте с фамилиями студентов, в них обучающихся. Отдельным запросом получите записи, расширяющие данный список по срав- нению с тем, который был бы получен внутренним соединением.
10. Напишите запрос для получения списка всех университетов вме- сте с фамилиями преподавателей, в них работающих. Отдельным запросом получите записи, расширяющие данный список по срав- нению с тем, который был бы получен внутренним соединением.
11. Напишите запрос для получения списка всех преподавателей вместе с университетами, в которых они работают. Есть ли отличие списка от того, который был бы получен внутренним соединением? Подтвердите отдельным запросом ваш вывод.
12. Напишите запрос для получения списка всех студентов и оценок, полученных ими на экзаменах. Отдельным запросом получите за- писи, расширяющие данный список по сравнению с тем, который был бы получен внутренним соединением.
13. Напишите запрос для получения списка всех экзаменационных оценок вместе с фамилиями получивших их студентов. Есть ли отличие списка от того, который был бы получен внутренним соединением? Подтвердите отдельным запросом ваш вывод.
14. Напишите запрос для получения полного списка предметов с со- ответствующими экзаменационными оценками. Есть ли отличие списка от того, который был бы получен внутренним соединени- ем? Подтвердите отдельным запросом ваш вывод.
15. Напишите запрос для получения полного списка оценок вместе с названиями предметов, по которым они получены. Есть ли отличие списка от того, который был бы получен внутренним соединением? Подтвердите отдельным запросом ваш вывод.
    * 1. **Использование псевдонимов при соединении копий од- ной таблицы.** Часто при получении информации из таблиц базы данных необходимо осуществлять соединение таблицы с ее же копией. Например, это требуется в случае, когда требуется найти фамилии студентов, имеющих одинаковые имена. При соединении таблицы с ее же копией вводят псевдонимы (алиасы) таблицы. Запрос для поиска фамилий студентов, имеющих одинаковые имена, выглядит следующим образом:

**SELECT** FIRST.SURNAME, SECOND.SURNAME **FROM** STUDENT FIRST, STUDENT SECOND **WHERE** FIRST.NAME = SECOND.NAME;

В этом запросе введены два псевдонима для одной таблицы STUDENT, что позволяет корректно задать выражение, связывающее две копии таблицы. Чтобы исключить повторения строк в выводимом результате запроса из-за повторного сравнения одной и той же пары студентов, необходимо задать порядок следования для двух значений так, чтобы одно значение было меньше, чем другое, что делает преди- кат асимметричным.

**SELECT** FIRST.SURNAME, SECOND.SURNAME **FROM** STUDENT FIRST, STUDENT SECOND **WHERE** FIRST.NAME = SECOND.NAME

**AND** FIRST.SURNAME < SECOND.SURNAME;

УПРАЖНЕНИЯ

1. Напишите запрос, выполняющий вывод списка всех пар фамилий студентов, проживающих в одном городе. При этом не вклю- чать в список комбинации фамилий студентов самих с собой (т. е. комбинации типа “Иванов–Иванов”) и комбинации фами- лий студентов, отличающиеся порядком следования (т. е. из двух комбинаций типа “Иванов–Петров” и “Петров–Иванов” включать только одну).
2. Напишите запрос, выполняющий вывод списка всех пар названий университетов, расположенных в одном городе, не включая в список комбинации названий университетов самих с собой и пары названий университетов, отличающиеся порядком следования.
3. Напишите запрос, который позволяет получить названия универ- ситетов с рейтингом, не меньшим рейтинга ВГУ, и городов, в которых они расположены.
4. И. Ф. Астахова, В. М. Мельников, А. П. Толстобров, В. В. Фертиков
5. Напишите запрос, выполняющий выборку идентификаторов сту- дентов, имеющих такие же оценки, что и студент с идентифика- тором 12.
6. Напишите запрос, выполняющий выборку всех пар идентифика- торов преподавателей, ведущих один и тот же предмет обучения.

2.16. Оператор объединения UNION

Оператор **UNION** используется для объединения выходных данных двух или более SQL-запросов в единое множество строк и столбцов. Например, для того, чтобы получить в одной таблице фамилии и идентификаторы студентов и преподавателей из Москвы, можно ис- пользовать следующий запрос:

**SELECT** ‘Студент’, SURNAME, STUDENT\_ID

**FROM** STUDENT

**WHERE** CITY = ‘Москва’

UNION

**SELECT** ‘Преподаватель’, SURNAME, LECTURER\_ID

**FROM** LECTURER

**WHERE** CITY = ‘Москва’;

Обратите внимание на то, что символом “;” (точка с запятой) оканчивается только последний запрос. Отсутствие этого символа в конце **SELECT**-запроса означает, что этот запрос, как и следующий за ним, является частью общего запроса с **UNION**.

Использование оператора **UNION** возможно только при объединении запросов, соответствующие столбцы которых *совместимы по объеди- нению*. Совместимость по объединению означает, что столбцы, как минимум, должны относиться к одному типу. При этом если говорить о таких конкретных характеристиках типов, как, например, количество символов для полей символьного типа, размер и точность числовых полей, то возможность оператора **UNION** зависит от конкретной реа- лизации СУБД. В одних системах задание оператора **UNION** требует полного совпадения характеристик типов столбцов, а в других воз- можно неявное приведение отличающихся характеристик. В примерах, приводимых в данном пособии, подразумевается, что такое неявное приведение имеет место. Аналогично, в зависимости от реализации СУБД, оператор **UNION** может иметь ограничение, связанное с исполь- зованием **NULL**-значений: если **NULL**-значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапро- сах объединения.

* + 1. **Устранение дублирования в UNION.** В отличие от обыч- ных запросов, **UNION** автоматически исключает из выходных данных дубликаты строк: например, в запросе

**SELECT** CITY

**FROM** STUDENT

**UNION SELECT** CITY

**FROM** LECTURER;

совпадающие наименования городов будут исключены.

Если все же необходимо в каждом запросе вывести все строки независимо от того, имеются ли такие же строки в других объеди- няемых запросах, то следует использовать во множественном запросе конструкцию с оператором **UNION ALL**. Так, в запросе

**SELECT** CITY

**FROM** STUDENT **UNION ALL SELECT** CITY

**FROM** LECTURER;

дубликаты значений городов, выводимые второй частью запроса, не будут исключаться.

Приведем еще один пример использования оператора **UNION**. Пусть необходимо составить отчет, содержащий для каждой даты сдачи эк- заменов сведения по каждому студенту, получившему максимальную или минимальную оценку.

**SELECT** ‘макс оц’, A.STUDENT\_ID, SURNAME, MARK, EXAM\_DATE

**FROM** STUDENT A, EXAM\_MARKS B

**WHERE** (A.STUDENT\_ID = B.STUDENT\_ID

**AND** B.MARK =

(**SELECT MAX**(MARK) **FROM** EXAM\_MARKS C

**WHERE** C.EXAM\_DATE = B.EXAM\_DATE))

UNION ALL

**SELECT** ‘мин оц ’, A.STUDENT\_ID, SURNAME, MARK, EXAM\_DATE

**FROM** STUDENT A, EXAM\_MARKS B

**WHERE** (A.STUDENT\_ID = B.STUDENT\_ID

**AND** B.MARK =

(**SELECT MIN**(MARK) **FROM** EXAM\_MARKS C

**WHERE** C.EXAM\_DATE = B.EXAM\_DATE));

Для различения строк, выводимых первой и второй частями запро- са, в них вставлены текстовые константы ‘макс оц’ и ‘мин оц’.

В приведенном запросе агрегирующие функции используются в подзапросах. Можно предложить альтернативный вариант формулиров-

3\*

ки запроса, возможно, более рациональный с точки зрения времени, затрачиваемого на выполнение запроса:

**SELECT** ‘макс оц’, A.STUDENT\_ID, SURNAME,

E.MARK, E.EXAM\_DATE

**FROM** STUDENT A,

(**SELECT** B.STUDENT\_ID, B.MARK, B.EXAM\_DATE

**FROM** EXAM\_MARKS B,

(**SELECT MAX**(MARK) **AS** MAX\_MARK, C.EXAM\_DATE

**FROM** EXAM\_MARKS C

**GROUP BY** C.EXAM\_DATE) D

**WHERE** B.EXAM\_DATE=D.EXAM\_DATE

**AND** B.MARK=MAX\_MARK) E

**WHERE** A.STUDENT\_ID=E.STUDENT\_ID

UNION ALL

**SELECT** ‘мин оц ’, A.STUDENT\_ID, SURNAME,

E.MARK, E.EXAM\_DATE

**FROM** STUDENT A,

(**SELECT** B.STUDENT\_ID, B.MARK, B.EXAM\_DATE

**FROM** EXAM\_MARKS B,

(**SELECT MIN**(MARK) **AS** MIN\_MARK, C.EXAM\_DATE

**FROM** EXAM\_MARKS C

**GROUP BY** C.EXAM\_DATE) D

**WHERE** B.EXAM\_DATE=D.EXAM\_DATE

**AND** B.MARK=MIN\_MARK) E

**WHERE** A.STUDENT\_ID=E.STUDENT\_ID;

* + 1. **Использование UNION с ORDER BY.** Предложение **ORDER BY** применяется для упорядочения выходных данных объеди- нения запросов так же, как и для отдельных запросов. Последний пример, при необходимости упорядочения выходных данных запроса по фамилиям студентов и датам экзаменов, может выглядеть так:

**SELECT** ‘макс оц’, A.STUDENT\_ID, SURNAME,

E.MARK, E.EXAM\_DATE

**FROM** STUDENT A,

(**SELECT** B.STUDENT\_ID, B.MARK, B.EXAM\_DATE

**FROM** EXAM\_MARKS B,

(**SELECT MAX**(MARK) **AS** MAX\_MARK, C.EXAM\_DATE

**FROM** EXAM\_MARKS C

**GROUP BY** C.EXAM\_DATE) D

**WHERE** B.EXAM\_DATE=D.EXAM\_DATE

**AND** B.MARK=MAX\_MARK) E

**WHERE** A.STUDENT\_ID=E.STUDENT\_ID

UNION ALL

**SELECT** ‘мин оц ’, A.STUDENT\_ID, SURNAME,

E.MARK, E.EXAM\_DATE

**FROM** STUDENT A,

(**SELECT** B.STUDENT\_ID, B.MARK, B.EXAM\_DATE

**FROM** EXAM\_MARKS B,

(**SELECT MIN**(MARK) **AS** MIN\_MARK, C.EXAM\_DATE

**FROM** EXAM\_MARKS C

**GROUP BY** C.EXAM\_DATE) D

**WHERE** B.EXAM\_DATE=D.EXAM\_DATE

**AND** B.MARK=MIN\_MARK) E

**WHERE** A.STUDENT\_ID=E.STUDENT\_ID

**ORDER BY** SURNAME, E.EXAM\_DATE;

Часто полезна операция объединения двух запросов, в которой второй запрос выбирает строки, исключенные первым.

Рассмотрим пример. Пусть в таблице STUDENT имеются записи о студентах без указания идентификатора университета. Требуется составить список студентов с указанием наименования университета для тех студентов, у которых эти данные есть, но при этом не отбра- сывая и студентов, у которых университет не указан. Можно получить желаемые сведения, сформировав объединение двух запросов, один из которых выполняет выборку студентов с названиями их университетов, а второй выбирает студентов с **NULL**-значениями в поле UNIV\_ID. В данном случае оказывается полезной возможность вставки в запрос констант, в нашем случае текстовой константы ‘неизвестен’, чтобы отметить в списке тех студентов, у которых отсутствует информация об университете.

**SELECT** SURNAME, NAME, UNIV\_NAME

**FROM** STUDENT, UNIVERSITY

**WHERE** STUDENT.UNIV\_ID = UNIVERSITY.UNIV\_ID

UNION

**SELECT** SURNAME, NAME, ‘неизвестен’

**FROM** STUDENT

**WHERE** UNIV\_ID **IS NULL ORDER BY** 1;

Для совместимости столбцов объединяемых запросов константу ‘неизвестен’ во втором запросе следует дополнить пробелами так, чтобы ее длина соответствовала длине поля UNIV\_NAME или исполь- зовать для согласования типов функцию **CAST**. В некоторых СУБД согласование типов поля и замещающей его текстовой константы осу- ществляется автоматически.

УПРАЖНЕНИЯ

1. Создайте объединение двух запросов, которые выдают значения полей UNIV\_NAME, CITY, RATING для всех университетов. Те из них, у которых рейтинг равен или выше 300, должны иметь комментарий ‘высокий’, все остальные — ‘низкий’.

70 *Гл. 2. Выборка данных (оператор* ***SELECT****)*



1. Напишите команду, которая выдает список фамилий студентов с комментарием: ‘успевает’ у студентов, имеющих все поло- жительные оценки; ‘не успевает’ для сдававших экзамены, но имеющих хотя бы одну неудовлетворительную оценку; ‘не сда- вал’ — для всех остальных. В выводимом результате фамилии студентов упорядочите по алфавиту.
2. Выведите объединенный список студентов и преподавателей, жи- вущих в Москве, с соответствующими комментариями ‘студент’ или ‘преподаватель’.
3. Выведите объединенный список студентов и преподавателей ВГУ с соответствующими комментариями ‘студент’ или ‘преподаватель’.
4. Для каждого города выведити названия университетов с мини- мальным и максимальным для данного города рейтингом. По- метьте строки списка словами ‘min’ и ‘max’, поместив их в дополнительном столбце.
5. Для каждого курса выведите фамилии студентов, получающих минимальные и максимальные на их курсе стипендии. Пометьте строки списка словами ‘min’ и ‘max’, поместив их в дополни- тельном столбце.
6. Для каждого курса выведите фамилии самого старшего и са- мого младшего студентов. Пометьте строки списка словами ‘младший’ и ‘старший’, поместив их в дополнительном столб- це.
7. Напишите запрос для получения полного списка университетов вместе с фамилиями студентов, которые в них учатся. Для уни- верситетов, не имеющих студентов, поместите в список фразу ‘Студентов нет’.
8. Напишите запрос для получения полного списка университетов вместе с фамилиями преподавателей, в них работающих. Для университетов, не имеющих преподавателей, поместите в список фразу ‘Преподавателей нет’.
9. Выведити полный список студентов вместе с оценками, получен- ными ими на экзаменах. Для студентов, не сдававших экзамены, в поле оценки поместите 0.

Г л а в а 3

**МАНИПУЛИРОВАНИЕ ДАННЫМИ**

3.1. Операторы манипулирования данными

В SQL для выполнения операций ввода данных в таблицу, их изменения и удаления предназначены три оператора языка манипулиро- вания данными (DML). Это операторы **INSERT** (ВСТАВИТЬ), **UPDATE** (ОБНОВИТЬ), **DELETE** (УДАЛИТЬ).

Оператор **INSERT** осуществляет ***вставку*** в таблицу новой строки.

В простейшем случае он имеет следующий вид:

**INSERT INTO** *<имя таблицы>*

**VALUES** (*<значение>*, *<значение>*, ...);

При такой записи указанные в скобках после ключевого слова **VALUES** значения вводятся в поля добавленной в таблицу новой стро- ки в том порядке, в котором соответствующие столбцы указаны при создании таблицы, т. е. в операторе **CREATE TABLE**.

Например, ввод новой строки в таблицу STUDENT может быть осуществлен следующим образом:

**INSERT INTO** STUDENT

**VALUES** (101, ‘Иванов’, ‘Александр’, 200, 3,

‘Москва’, ‘6/10/1979’, 15);

Чтобы такой оператор мог быть выполнен, таблица с указанным в нем именем (STUDENT) должна быть предварительно определена (создана) оператором **CREATE TABLE**. Если в какое-либо поле необхо- димо вставить **NULL**-значение, то оно вводится как обычное значение:

**INSERT INTO** STUDENT

**VALUES** (101, ‘Иванов’, **NULL**, 200, 3,

‘Москва’, ‘6/10/1979’, 15);

В случаях, когда необходимо ввести значения полей в порядке, отличном от порядка столбцов, заданного оператором **CREATE TABLE**,

72 *Гл. 3. Манипулирование данными*



или если требуется ввести значения не во все столбцы, то следует использовать следующую форму оператора **INSERT**:

**INSERT INTO** STUDENT (STUDENT\_ID, CITY, SURNAME, NAME)

**VALUES** (101, ‘Москва’, ‘Иванов’, ‘Саша’);

Столбцам, наименования которых не указаны в списке, при- веденном в скобках, автоматически присваивается значение по умолчанию, если оно назначено при определении таблицы (оператор **CREATE TABLE**), либо значение **NULL**.

С помощью оператора **INSERT** можно извлечь значение из одной таблицы и разместить его в другой, к примеру, запросом следующего вида:

**INSERT INTO** STUDENT1

SELECT \*

**FROM** STUDENT

**WHERE** CITY = ‘Москва’;

При этом таблица STUDENT1 должна быть предварительно создана оператором **CREATE TABLE** (раздел 4.1) и иметь структуру, идентич- ную таблице STUDENT.

***Удаление*** строк из таблицы осуществляется с помощью оператора

DELETE.

Следующее выражение удаляет все строки таблицы EXAM\_MARKS1:

**DELETE FROM** EXAM\_MARKS1;

В результате таблица становится пустой (после этого она может быть удалена командой **DROP TABLE**).

Для удаления из таблицы сразу нескольких строк, удовлетво- ряющих некоторому условию, можно воспользоваться предложением **WHERE**, например,

**DELETE FROM** EXAM\_MARKS1

**WHERE** STUDENT\_ID = 103;

Можно удалить группу строк

**DELETE FROM** STUDENT1

**WHERE** CITY = ‘Москва’;

Оператор **UPDATE** позволяет ***изменять***, т. е. обновлять, значения некоторых или всех полей в существующей строке или строках таб- лицы. Например, чтобы для всех университетов, сведения о которых находятся в таблице UNIVERSITY1, изменить рейтинг на значение 200, можно использовать конструкцию:

**UPDATE** UNIVERSITY1

**SET** RATING = 200;

* 1. *Операторы манипулирования данными* 73



Если требуется изменить значения полей в конкретных строках таблицы, то в операторе **UPDATE** можно использовать предикат, зада- ваемый в предложении **WHERE**.

**UPDATE** UNIVERSITY1

**SET** RATING = 200

**WHERE** CITY = ‘Москва’;

В результате выполнения этого запроса будет изменен рейтинг только у университетов, расположенных в Москве.

Оператор **UPDATE** позволяет изменять не только один, но и множе- ство столбцов. Для указания конкретных столбцов, значения которых должны быть модифицированы, используется предложение **SET**.

Например, пусть наименование предмета обучения ‘Математика’ (для него SUBJ\_ID = 43) должно быть заменено на название ‘Высшая математика’, при этом идентификационный номер необходимо сохра- нить, но в соответствующие поля строки таблицы ввести новые дан- ные об этом предмете обучения. Запрос будет выглядеть следующим образом:

**UPDATE** SUBJECT1

**SET** SUBJ\_NAME = ‘Высшая математика’, HOUR = 36, SEMESTER = 1

**WHERE** SUBJ\_ID = 43;

В предложении **SET** оператора **UPDATE** можно использовать ска- лярные выражения, указывающие способ изменения значений поля, в которые могут входить значения изменяемого и других полей.

**UPDATE** UNIVERSITY1

**SET** RATING = RATING\*2;

Например, чтобы удвоить значения поля STIPEND в таблице

STUDENT1 для студентов из Москвы, можно использовать запрос

**UPDATE** STUDENT1

**SET** STIPEND = STIPEND\*2

**WHERE** CITY = ‘Москва’;

УПРАЖНЕНИЯ

1. Напишите команду, которая вводит в таблицу SUBJECT строку для нового предмета обучения со следующими значениями полей: SEMESTER = 4; SUBJ\_NAME = ‘Алгебра’; HOUR = 72; SUBJ\_ID = 201.
2. Введите запись для нового студента, которого зовут Орлов Нико- лай, обучающегося на первом курсе ВГУ, живущего в Воронеже, сведения о дате рождения и размере стипендии неизвестны.
3. Напишите команду, удаляющую из таблицы EXAM\_MARKS записи обо всех оценках студента, идентификатор которого равен 100.
4. Напишите команду, которая увеличивает на 5 значение рейтинга всех университетов, расположенных в Санкт-Петербурге.
5. Измените в таблице значение города, в котором проживает сту- дент Иванов, на ‘Воронеж’.
   1. Использование подзапросов в INSERT

Применение оператора **INSERT** с подзапросом позволяет загружать сразу несколько строк в одну таблицу, используя информацию из дру- гой таблицы. В то время как оператор **INSERT**, использующий **VALUES**, добавляет только одну строку, **INSERT** с подзапросом добавляет в таб- лицу столько строк, сколько подзапрос извлекает из другой таблицы. При этом количество и тип возвращаемых подзапросом столбцов долж- но соответствовать количеству и типу столбцов таблицы, в которую вставляются данные.

Например, пусть таблица STUDENT1 имеет структуру, полностью совпадающую со структурой таблицы STUDENT. В разделе 3.1 приве- ден запрос, позволяющий заполнить таблицу STUDENT1 записями из таблицы STUDENT обо всех студентах из Москвы:

**INSERT INTO** STUDENT1

SELECT \*

**FROM** STUDENT

**WHERE** CITY = ‘Москва’;

Для того же, чтобы добавить в таблицу STUDENT1 сведения обо всех студентах, которые *учатся* в Москве, можно использовать в предложении **WHERE** соответствующий подзапрос. Например,

**INSERT INTO** STUDENT1

SELECT \*

**FROM** STUDENT

**WHERE** UNIV\_ID **IN**

(**SELECT** UNIV\_ID

**FROM** UNIVERSITY

**WHERE** CITY = ‘Москва’);

* + 1. **Использование подзапросов, основанных на таблицах внешних запросов.** Предположим, существует таблица SSTUD, в которой хранятся сведения о студентах, обучающихся в том же городе, в котором они живут. Можно заполнить эту таблицу данными из таб- лицы STUDENT, используя связанные подзапросы, следующим образом:

**INSERT INTO** SSTUD

SELECT \*

**FROM** STUDENT A

**WHERE** CITY **IN**

(**SELECT** CITY

**FROM** UNIVERSITY B

**WHERE** A.UNIV\_ID = B.UNIV\_ID);

Предположим, что требуется выбрать список студентов, имеющих максимальный балл на каждый день сдачи экзаменов, и разместить его в другой таблице с именем EXAM. Это можно осуществить с помощью запроса

**INSERT INTO** EXAM

**SELECT** EXAM\_ID, STUDENT\_ID, SUBJ\_ID, MARK, EXAM\_DATE

**FROM** EXAM\_MARKS A **WHERE** MARK = (**SELECT MAX**(MARK)

**FROM** EXAM\_MARKS B

**WHERE** A.EXAM\_DATE = B.EXAM\_DATE);

* + 1. **Использование подзапросов с DELETE.** Пусть филиал университета в Нью-Васюках ликвидирован, и требуется удалить из таблицы STUDENT записи о студентах, которые там учились. Эту операцию можно выполнить с помощью запроса

DELETE

**FROM** STUDENT

**WHERE** UNIV\_ID **IN**

(**SELECT** UNIV\_ID

**FROM** UNIVERSITY

**WHERE** CITY = ‘Нью-Васюки’);

В предикате предложения **FROM** (подзапроса) нельзя ссылаться на таблицу, из которой осуществляется удаление. Однако можно ссы- латься на текущую строку из таблицы, являющуюся кандидатом на удаление, т. е. на строку, которая в настоящее время проверяется в основном предикате.

DELETE

**FROM** STUDENT

**WHERE** EXISTS (**SELECT** \*

**FROM** UNIVERSITY

**WHERE** RATING = 401

**AND** STUDENT.UNIV\_ID = UNIVERSITY.UNIV\_ID);

Часть **AND** предиката внутреннего запроса ссылается на таблицу STUDENT. Команда удаляет данные о студентах, которые учатся в уни- верситетах с рейтингом 401. Существуют и другие способы решения этой задачи:

DELETE

**FROM** STUDENT

**WHERE** 401 **IN**

(**SELECT** RATING

**FROM** UNIVERSITY

**WHERE** STUDENT.UNIV\_ID = UNIVERSITY.UNIV\_ID);

Пусть для каждого дня сдачи экзаменов нужно найти наименьшее значение выставленной оценки и удалить из таблицы сведения о сту- денте, который получил эту оценку. Запрос будет иметь вид

DELETE

**FROM** STUDENT

**WHERE** STUDENT\_ID **IN**

(**SELECT** STUDENT\_ID **FROM** EXAM\_MARKS A **WHERE** MARK =

(**SELECT MIN**(MARK) **FROM** EXAM\_MARKS B

**WHERE** A.EXAM\_DATE = B.EXAM\_DATE));

Так как столбец STUDENT\_ID является первичным ключом, то удаляется единственная строка.

Если в какой-то день сдавался только один экзамен (т. е. получена только одна минимальная оценка) и по какой-либо причине запись, в которой находится эта оценка, требуется оставить, то решение будет иметь вид:

DELETE

**FROM** STUDENT

**WHERE** STUDENT\_ID **IN**

(**SELECT** STUDENT\_ID **FROM** EXAM\_MARKS A **WHERE** MARK =

(**SELECT MIN**(MARK) **FROM** EXAM\_MARKS B

**WHERE** A.EXAM\_DATE = B.EXAM\_DATE

**AND** 1 <

(**SELECT COUNT**(SUBJ\_ID) **FROM** EXAM\_MARKS B

**WHERE** A.EXAM\_DATE = B.EXAM\_DATE)));

* + 1. **Использование подзапросов с UPDATE.** С помощью ко- манды **UPDATE** можно применять подзапросы в любой форме, прием- лемой для команды **DELETE**.

Например, используя связанные подзапросы, можно увеличить зна- чение размера стипендии на 20 в записях студентов, сдавших экзамены на 4 и 5:

**UPDATE** STUDENT1

**SET** STIPEND = STIPEND + 20

**WHERE** 4 <= (**SELECT MIN**(MARK)

**FROM** EXAM\_MARKS

**WHERE** EXAM\_MARKS.STUDENT\_ID = STUDENT1.STUDENT\_ID);

Другой запрос — уменьшить величину стипендии на 20 всем сту- дентам, получившим на экзамене минимальную оценку:

**UPDATE** STUDENT1

**SET** STIPEND = STIPEND - 20

**WHERE** STUDENT\_ID **IN**

(**SELECT** STUDENT\_ID **FROM** EXAM\_MARKS A **WHERE** MARK =

(**SELECT MIN**(MARK) **FROM** EXAM\_MARKS B

**WHERE** A.EXAM\_DATE = B.EXAM\_DATE));

УПРАЖНЕНИЯ

1. Пусть существует таблица с именем STUDENT1, определения столбцов которой полностью совпадают с определениями столб- цов таблицы STUDENT. Вставьте в эту таблицу сведения о сту- дентах, успешно сдавших экзамены более чем по пяти предметам обучения.
2. Напишите команду, удаляющую из таблицы SUBJECT1 сведения о предметах обучения, по которым студентами не получено ни одной оценки.
3. Напишите запрос, увеличивающий размер стипендии на 20% всем студентам, у которых общая сумма баллов превышает зна- чение 50.

*4.2. Использование индексации для быстрого доступа к данным* 79



Г л а в а 4

**СОЗДАНИЕ ОБЪЕКТОВ БАЗЫ ДАННЫХ**

* 1. Создание таблиц базы данных

Создание объектов базы данных осуществляется с помощью опера- торов языка определения данных (DDL).

Таблицы базы данных создаются с помощью оператора **CREATE TABLE**. Эта команда создает пустую таблицу, т.е. таблицу, не имеющую строк. Значения в эту таблицу вводятся с помощью оператора **INSERT**. Оператор **CREATE TABLE** определяет имя таблицы и множество по- именованных столбцов в указанном порядке. Для каждого столбца должны быть определены тип и размер. Каждая создаваемая таблица должна иметь по крайней мере один столбец. Упрощенный синтаксис оператора **CREATE TABLE** имеет следующий вид:

**CREATE TABLE** *<имя таблицы>*

(*<имя столбца><тип данных>*[(*<размер>*)], ...);

Следующий пример показывает запрос, создающий таблицу

STUDENT1:

**CREATE TABLE** STUDENT1 (STUDENT\_ID **INTEGER**,

SURNAME **VARCHAR**(60),

NAME **VARCHAR**(60),

STIPEND **DOUBLE**,

KURS **INTEGER**,

CITY **VARCHAR**(60),

BIRTHDAY **DATE**,

UNIV\_ID **INTEGER**);

* 1. Использование индексации для быстрого доступа к данным

Операции поиска-выборки (**SELECT**) данных из таблиц по значени- ям их полей могут быть существенно ускорены путем использования индексации данных. Индекс содержит упорядоченный (в алфавитном или числовом порядке) список содержимого столбца или группы столб- цов в индексируемой таблице с идентификаторами соответствующих строк (**ROWID**). Для пользователей индексирование таблицы по тем или иным столбцам представляет собой способ *логического* упорядо- чения значений индексированных столбцов, позволяющего, в отличие от последовательного перебора строк, существенно повысить скорость доступа к конкретным строкам таблицы при выборках, использующих значения этих столбцов. Индексация позволяет находить содержащий индексированную строку блок данных, выполняя небольшое число обращений к внешнему устройству хранения данных.

При использовании индексации следует, однако, иметь в виду, что управление индексом существенно увеличивает время выполнения операций, связанных с обновлением данных (таких, как **INSERT** и **DELETE**), так как эти операции требуют перестройки индексов.

Индексы можно создавать как по одному, так и по множеству полей. Если указано более одного поля для создания единственного индекса, то данные упорядочиваются по значениям первого поля, по которому осуществляется индексирование. Внутри получившейся груп- пы осуществляется упорядочивание по значениям второго поля, для получившихся в результате групп осуществляется упорядочивание по значениям третьего поля и т.д.

Синтаксис оператора создания индекса имеет следующий вид:

**CREATE INDEX** *<имя индекса>* **ON** *<имя таблицы>*

(*<имя столбца>* [,*<имя столбца>*]...);

При этом таблица должна уже быть созданной и содержать столб- цы, имена которых указаны в команде создания индекса. Имя индекса, определенное в команде, должно быть уникальным в базе данных. Будучи однажды созданным, индекс является невидимым для пользо- вателя, все операции с ним осуществляет СУБД.

Пример.

Если таблица EXAM\_MARKS часто используется для поиска оценки конкретного студента по значению поля STUDENT\_ID, то следует со- здать индекс по этому полю:

**CREATE INDEX** STUDENT\_ID\_1

**ON** EXAM\_MARKS (STUDENT\_ID);

Для удаления индекса (при этом обязательно требуется знать его имя) используется команда **DROP INDEX**, имеющая следующий син- таксис:

**DROP INDEX** *<имя индекса>*;

Удаление индекса не изменяет содержимого поля или полей, индекс которых удаляется.

* 1. Изменение существующей таблицы

Для модификации структуры и параметров существующей таблицы используется оператор **ALTER TABLE**. Упрощенный синтаксис опера- тора **ALTER TABLE** для добавления столбцов в таблицу имеет вид

**ALTER TABLE** *<имя таблицы>*

**ADD** (*<имя столбца><тип данных><размер>*);

При выполнении этого оператора для существующих в таблице строк добавляется новый столбец, в который заносится **NULL**-значение. Этот столбец становится последним в таблице. Можно добавлять несколько столбцов, в этом случае их определения в команде **ALTER TABLE** разделяются запятой.

Возможно изменение описания столбцов. Часто это связано с изме- нением размеров столбцов, добавлением или удалением ограничений, накладываемых на их значения. Синтаксис оператора в этом случае имеет вид

**ALTER TABLE** *<имя таблицы>*

**MODIFY** *<имя столбца><тип данных><размер/точность>*;

Следует иметь в виду, что модификация характеристик столбца может осуществляться не в любом случае, а с учетом следующих ограничений:

изменение типа данных возможно только, если столбец пуст; для незаполненного столбца можно изменять размер/точность; для заполненного столбца размер/точность можно увеличить, но нельзя понизить;

*•*

*•*

ограничение **NOT NULL** может быть установлено, если ни одно значение в столбце не содержит **NULL**; опцию **NOT NULL** всегда можно отменить;

*•*

* разрешается изменять значения, установленные по умолчанию.
  1. Удаление таблицы

Синтаксис оператора, осуществляющего удаление таблицы, имеет следующий вид:

**DROP TABLE** *<имя таблицы>*;

УПРАЖНЕНИЯ

1. Напишите команду **CREATE TABLE** для создания таблицы

LECTURER1.

1. Напишите команду **CREATE TABLE** для создания таблицы

SUBJECT1.

1. Напишите команду **CREATE TABLE** для создания таблицы

UNIVERSITY1.

1. Напишите команду **CREATE TABLE** для создания таблицы

EXAM\_MARKS1.

1. Напишите команду **CREATE TABLE** для создания таблицы

SUBJ\_LECT1.

1. Напишите команду, которая позволит быстро выбрать данные о студентах по курсам, на которых они учатся.
2. Создайте индекс, который позволит для каждого студента быстро осуществить поиск оценок, сгруппированных по датам.
   1. Ограничения на множество допустимых значений данных

До сих пор рассматривалось только следующее ограничение: зна- чения, вводимые в таблицу, должны иметь типы данных и разме- ры, совместимые с типами и размерами данных столбцов, в которые эти значения вводятся (как определено в команде **CREATE TABLE** или **ALTER TABLE**). Описание таблицы может быть дополнено бо- лее сложными ограничениями, накладываемыми на значения, которые могут быть вставлены в столбец или группу столбцов. Ограничения (**CONSTRAINTS**) являются частью определения таблицы.

При создании (изменении) таблицы могут быть определены ограни- чения на вводимые значения. В этом случае SQL будет отвергать любое из вводимых значений, не соответствующее заданному ограничению. Ограничения могут быть статическими, ограничивающими значения или диапазон значений, вставляемых в столбец (**CHECK**, **NOT NULL**). Они могут иметь связь со всеми значениями столбца, ограничивая новые строки значениями, которые не содержатся в столбцах или их наборах (уникальные значения, первичные ключи). Ограничения могут также определяться связью со значениями, находящимися в другой таблице, допуская, например, вставку в столбец только тех значений, которые *в данный момент* содержатся также в другом столбце другой или этой же таблицы (внешний ключ). Эти ограничения носят дина- мический характер.

Существуют два основных типа ограничений — ограничения на столбцы и ограничения на таблицу. Ограничения на столбцы (**COLUMN CONSTRAINTS**) применимы только к отдельным столбцам, а огра- ничения на таблицу (**TABLE CONSTRAINTS**) применимы к группам, состоящим из одного или более столбцов. Ограничения на столбец

добавляются в конце определения столбца после указания типа дан- ных и перед окончанием описания столбца (запятой). Ограничения на таблицу размещаются в конце определения таблицы, после определе- ния последнего столбца. Команда **CREATE TABLE** имеет следующий синтаксис, расширенный включением ограничений:

**CREATE TABLE** *<имя таблицы>*

(*<имя столбца><тип данных><ограничения на столбец>*,

*<имя столбца><тип данных><ограничения на столбец>*,

...

*<ограничения на таблицу>*

(*<имя столбца>*[, *<имя столбца>*...])...);

Поля, заданные в круглых скобках после описания ограничений таблицы, — это поля, на которые эти ограничения распространяются. Ограничения на столбцы применяются к тем столбцам, за именами которых они описаны.

* + 1. **Ограничение NOT NULL.** Чтобы запретить возможность использования в поле **NULL**-значений, можно при создании таблицы командой **CREATE TABLE** указать для соответствующего столбца клю- чевое слово **NOT NULL**. Это ограничение применимо только к столб- цам таблицы. Как уже говорилось выше, **NULL** — это специальный маркер, обозначающий тот факт, что поле пусто. Но он полезен не всегда. Первичные ключи, например, в принципе не должны содер- жать **NULL**-значений (быть пустыми), поскольку это нарушило бы требование уникальности первичного ключа (более строго – функци- ональную зависимость атрибутов таблицы от первичного ключа). Во многих других случаях также необходимо, чтобы поля *обязательно* содержали определенные значения. Если ключевое слово **NOT NULL** размещается непосредственно после типа данных (включая размер) столбца, то любые попытки оставить значение поля пустым (ввести в поле **NULL**-значение) будут отвергнуты системой.

Например, для того, чтобы в определении таблицы STUDENT1 запретить использование **NULL**-значений для столбцов STUDENT\_ID, SURNAME и NAME, можно записать следующее:

**CREATE TABLE** STUDENT1 (STUDENT\_ID INTEGER **NOT NULL**,

SURNAME **CHAR**(25) **NOT NULL**, NAME **CHAR**(10) **NOT NULL**, STIPEND **INTEGER**,

KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**, UNIV\_ID **INTEGER**);

Важно помнить, что если для столбца указано **NOT NULL**, то при использовании оператора **INSERT** обязательно должно быть указано

конкретное значение, вводимое в это поле. При отсутствии ограниче- ния **NOT NULL** в столбце значение может отсутствовать, если только не указано значение столбца по умолчанию (**DEFAULT**). Если при создании таблицы ограничение **NOT NULL** не было указано, то его можно указать позже, используя оператор **ALTER TABLE**. Для вновь вводимого с помощью оператора **ALTER TABLE** столбца можно задать ограничение **NOT NULL**, если таблица, в которую добавляется столбец, пустая, или если для столбца указывается значение по умолчанию.

* + 1. **Уникальность как ограничение на столбец.** Иногда тре- буется, чтобы все значения, введенные в столбец, отличались друг от друга. Например, этого требуют первичные ключи. Если при создании таблицы для столбца указывается ограничение **UNIQUE**, то база дан- ных отвергает любую попытку ввести в это поле какой-либо строки значение, уже содержащееся в том же поле другой строки. Можно предложить следующее определение таблицы STUDENT, использующее ограничение **UNIQUE**:

**CREATE TABLE** STUDENT1

(STUDENT\_ID **INTEGER NOT NULL UNIQUE**, SURNAME **CHAR**(25) **NOT NULL**,

NAME **CHAR**(10) **NOT NULL**, STIPEND **INTEGER**,

KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**, UNIV\_ID **INTEGER**);

Объявляя поле STUDENT\_ID уникальным, можно быть уверенным, что в таблице не появится записей для двух студентов с одинаковы- ми идентификаторами. Столбцы, отличные от первичного ключа, для которых требуется поддержать уникальность значений, называются возможными ключами (**CANDIDATE KEYS**) или уникальными ключами (**UNIQUE KEYS**).

* + 1. **Уникальность как ограничение таблицы.** Можно устано- вить требование уникальности для сочетания значений группы полей. В этом случае ключевое слово **UNIQUE** указывается в качестве огра- ничений *таблицы*. При объединении полей в группу важен порядок, в котором они указываются. Ограничение на таблицу **UNIQUE** полезно, если требуется поддерживать уникальность группы полей. Например, если в нашей базе данных не допускается, чтобы студент сдавал в один день больше одного экзамена, то можно в таблице объявить уникальной комбинацию значений полей STUDENT\_ID и EXAM\_DATE. Для этого следует создать таблицу EXAM\_MARKS таким способом:

**CREATE TABLE** EXAM\_MARKS1 (EXAM\_ID **INTEGER NOT NULL**,

STUDENT\_ID **INTEGER NOT NULL**, SUBJ\_ID **INTEGER NOT NULL**, MARK **INTEGER**,

EXAM\_DATE **DATE NOT NULL**,

**UNIQUE** (STUDENT\_ID, EXAM\_DATE));

Обратите внимание на то, что оба поля в ограничении таблицы **UNIQUE** все еще используют ограничение столбца **NOT NULL**. Если бы использовалось ограничение столбца **UNIQUE** для поля STUDENT\_ID, то такое ограничение таблицы было бы необязательным.

Если значения поля STUDENT\_ID должны быть уникальными для каждой строки в таблице EXAM\_MARKS, то это можно сделать, объ- явив **UNIQUE** как ограничение самого поля STUDENT\_ID. В этом случае будет обеспечена уникальность в комбинации значений полей STUDENT\_ID, EXAM\_DATE. Следовательно, указание **UNIQUE** как огра- ничение таблицы наиболее полезно использовать в случаях, когда не требуется уникальность индивидуальных полей, как это имеет место на самом деле в рассматриваемом примере.

* + 1. **Присвоение имен ограничениям.** Ограничениям таблиц можно присваивать уникальные имена. Преимущество явного задания имени ограничения состоит в том, что в этом случае при выдаче системой сообщения о нарушении установленного ограничения будет указано его имя, что упрощает обнаружение ошибок.

Для присвоения имени ограничению используется несколько изме- ненный синтаксис команд **CREATE TABLE** и **ALTER TABLE**.

Приведенный выше пример запроса изменяется следующим обра- зом:

**CREATE TABLE** EXAM\_MARKS1 (EXAM\_ID **INTEGER NOT NULL**,

STUDENT\_ID **INTEGER NOT NULL**, SUBJ\_ID **INTEGER NOT NULL**, MARK **INTEGER**,

EXAM\_DATE **DATE NOT NULL**, **CONSTRAINT** STUD\_SUBJ

**UNIQUE** (STUDENT\_ID, EXAM\_DATE));

В этом запросе STUD\_SUBJ\_CONSTR — это имя, присвоенное ука- занному ограничению таблицы.

* + 1. **Ограничение первичных ключей.** *Первичные ключи таб- лицы* — это специальные случаи комбинирования ограничений **UNIQUE** и **NOT NULL**. Первичные ключи имеют следующие особенности:

таблица может содержать только один первичный ключ; внешние ключи по умолчанию ссылаются на первичный ключ таблицы;

*•*

*•*

первичный ключ является идентификатором строк таблицы (стро- ки, однако, могут идентифицироваться и другими способами).

*•*

Улучшенный вариант создания таблицы STUDENT1 с объявленным первичным ключом имеет теперь следующий вид:

**CREATE TABLE** STUDENT1

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25) **NOT NULL**,

NAME **CHAR**(10) **NOT NULL**, STIPEND **INTEGER**,

KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**, UNIV\_ID **INTEGER**);

* + 1. **Составные первичные ключи.** Ограничение **PRIMARY KEY** может также быть применено для нескольких полей, составля- ющих уникальную комбинацию значений — *составной* первичный ключ. Рассмотрим таблицу EXAM\_MARKS. Очевидно, что ни к полю идентификатора студента (STUDENT\_ID), ни к полю идентификатора предмета обучения (EXAM\_ID) по отдельности нельзя предъявить тре- бование уникальности. Однако для того, чтобы в таблице не могли появиться разные записи для одинаковых комбинаций значений полей STUDENT\_ID и EXAM\_ID (конкретный студент на конкретном экза- мене не может получить более одной оценки), имеет смысл объявить уникальной комбинацию этих полей. Для этого мы можем приме- нить ограничение таблицы **PRIMARY KEY**, объявив пару EXAM\_ID и STUDENT\_ID первичным ключом таблицы:

**CREATE TABLE** NEW\_EXAM\_MARKS (STUDENT\_ID **INTEGER NOT NULL**,

SUBJ\_ID **INTEGER NOT NULL**, MARK **INTEGER**,

DATA **DATE**, **CONSTRAINT** EX\_PR\_KEY

**PRIMARY KEY** (EXAM\_ID, STUDENT\_ID));

* + 1. **Проверка значений полей.** Ограничение **CHECK** позволяет определять условие, которому должно удовлетворять вводимое в поле таблицы значение, прежде чем оно будет принято. Любая попытка обновить или заменить значение поля таким, для которого предикат, задаваемый ограничением **CHECK**, имеет значение *ложь*, будет отвер- гаться.

Рассмотрим таблицу STUDENT. Значение столбца STIPEND в этой таблице STUDENT выражается десятичным числом. Наложим на зна- чения этого столбца следующее ограничение — величина размера стипендии должна быть меньше 200.

Соответствующий запрос имеет следующий вид:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25) **NOT NULL**,

NAME **CHAR**(10) **NOT NULL**,

STIPEND **INTEGER CHECK** (STIPEND < 200), KURS **INTEGER**,

CITY **CHAR**(15), BIRTHDAY **DATE**, UNIV\_ID **INTEGER**);

* + 1. **Проверка ограничивающих условий с использованием составных полей.** Ограничение **CHECK** можно использовать в каче- стве табличного ограничения, то есть при необходимости включить более одного поля строки в ограничивающее условие.

Предположим, что ограничение на размер стипендии (меньше 200) должно распространяться только на студентов, живущих в Воронеже. Это можно указать в запросе со следующим табличным ограничением **CHECK**:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25) **NOT NULL**,

NAME **CHAR**(10) **NOT NULL**, STIPEND **INTEGER**,

KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**,

UNIV\_ID **INTEGER UNIQUE**,

**CHECK** ((STIPEND < 200 **AND** CITY = ‘Воронеж’) **OR**

**NOT** (CITY = ‘Воронеж’)));

или в несколько иной записи:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25) **NOT NULL**,

NAME **CHAR**(10) **NOT NULL**, STIPEND **INTEGER**,

KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**,

UNIV\_ID **INTEGER UNIQUE**, **CONSTRAINT** STUD\_CHECK

**CHECK**(STIPEND < 200 **AND** CITY = ‘Воронеж’) **OR**

**NOT** (CITY = ‘Воронеж’)));

* + 1. **Установка значений по умолчанию.** В SQL имеется воз- можность при вставке в таблицу строки, не указывая значений неко-

торого поля, определять значение этого поля по умолчанию. Наиболее часто используемым значением по умолчанию является **NULL**. Это значение принимается по умолчанию для любого столбца, для которого не было установлено ограничение **NOT NULL**.

Значение поля по умолчанию указывается в команде **CREATE TABLE** тем же способом, что и ограничение столбца, с помощью ключевого слова

**DEFAULT** *<значение по умолчанию>*;

Опция **DEFAULT** не является ограничением, так как она не ограни- чивает значения, вводимые в поле, а просто конкретизирует значение поля в случае, если оно *не было задано*.

Предположим, что основная масса студентов, информация о кото- рых находится в таблице STUDENT, проживает в Воронеже. Чтобы при задании атрибутов не вводить для большинства студентов название города ‘Воронеж’, можно установить его как значение поля CITY по умолчанию, определив таблицу STUDENT следующим образом:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25) **NOT NULL**,

NAME **CHAR**(10) **NOT NULL**,

STIPEND **INTEGER CHECK** (STIPEND < 200), KURS **INTEGER**,

CITY **CHAR**(15) **DEFAULT** ‘Воронеж’, BIRTHDAY **DATE**,

UNIV\_ID **INTEGER**);

Другая цель практического применения задания значения по умол- чанию — это использование его как альтернативы для **NULL**. Как уже отмечалось выше, присутствие **NULL** в качестве возможных значений поля существенно усложняет интерпретацию операций сравнения, в которых участвуют значения таких полей, поскольку **NULL** представля- ет собой признак того, что фактическое значение поля *неизвестно* или *неопределенно*. Следовательно, строго говоря, сравнение с ним любого конкретного значения в рамках двузначной булевой логики некоррект- но, за исключением специальной операции сравнения **IS NULL**, которая определяет, является ли содержимое поля каким-либо значением или оно отсутствует. Действительно, каким образом в рамках двузначной логики ответить на вопрос, истинно или ложно условие: CITY = ‘Воронеж’, если текущее значение поля CITY неизвестно (содержит **NULL**)?

Во многих случаях использование вместо **NULL** значения, подстав- ляемого в поле по умолчанию, может существенно упростить исполь- зование значений поля в предикатах.

Например, можно установить для столбца опцию **NOT NULL**, а для неопределенных значений числового типа установить значение

по умолчанию “равно нулю”, или для полей типа **CHAR** — пробел, использование которых в операциях сравнения не вызывает никаких проблем.

При использовании значений по умолчанию в принципе допустимо применять ограничения **UNIQUE** или **PRIMARY KEY** в соответствую- щем поле. При этом, однако, следует иметь в виду отсутствие в таком ограничении практического смысла, поскольку только одна строка в таблице сможет принять значение, совпадающее с этим значением по умолчанию.

УПРАЖНЕНИЯ

1. Создайте таблицу EXAM\_MARKS так, чтобы не допускался ввод в таблицу двух записей об оценках одного студента по конкретным экзамену и предмету обучения, а также, чтобы не допускалось проведение двух экзаменов по любым предметам в один день.
2. Создайте таблицу предметов обучения SUBJECT так, чтобы ко- личество отводимых на предмет часов по умолчанию было равно 36, не допускались записи с отсутствующим количеством часов, поле SUBJ\_ID являлось первичным ключом таблицы, а значения семестров (поле SEMESTR) лежали в диапазоне от 1 до 12.
3. Создайте таблицу EXAM\_MARKS таким образом, чтобы значения поля EXAM\_ID были больше значений поля SUBJ\_ID, а значения поля SUBJ\_ID были больше значений поля STUDENT\_ID; пусть также будут запрещены значения **NULL** в любом из этих трех полей.

4.6. Поддержка целостности данных

В таблицах рассматриваемой базы данных значения некоторых по- лей связаны друг с другом. Так, поле STUDENT\_ID в таблице STUDENT и поле STUDENT\_ID в таблице EXAM\_MARKS связаны тем, что описы- вают одни и те же объекты, т.е. содержат идентификаторы студентов, информация о которых хранится в базе. Более того, значения иден- тификаторов студентов, которые допустимы в таблице EXAM\_MARKS, должны выбираться только из списка значений STUDENT\_ID, фак- тически присутствующих в таблице STUDENT, т.е. принадлежащих реально описанным в базе студентам. Аналогично, значения поля UNIV\_ID таблицы STUDENT должны соответствовать идентификато- рам университетов UNIV\_ID, фактически присутствующим в таблице UNIVERSITY, а значения поля SUBJ\_ID таблицы EXAM\_MARKS долж- ны соответствовать идентификаторам предметов обучения, фактически присутствующим в таблице SUBJECT.

Ограничения, накладываемые указанным типом связи, называют- ся *ограничениями ссылочной целостности*. Они составляют важную часть описания характеристик предметной области, обеспечения кор-

ректности данных, хранящихся в таблицах. Команды описания таблиц DML имеют средства, позволяющие описывать ограничения ссылочной целостности и обеспечивать поддержание такой целостности при ма- нипулировании значениями полей базы данных.

* + 1. **Внешние и родительские ключи.** Когда каждое значение, присутствующее в одном поле таблицы, представлено в другом поле другой или этой же таблицы, говорят, что первое поле ссылается на второе. Это указывает на прямую связь между значениями двух полей. Поле, которое ссылается на другое поле, называется ***внешним ключом***, а поле, на которое ссылается другое поле, называется ***родительским ключом***. В качестве родительского ключа может выступать только поле, являющееся возможным (первичным или альтернативным) клю- чом отношения. Например, поле UNIV\_ID таблицы STUDENT — это внешний ключ, ссылающийся на поле UNIV\_ID таблицы UNIVERSITY, являющееся ее первичным ключом и выступающее в данном случае в качестве родительского ключа для этого внешнего ключа.

Хотя в приведенном примере имена внешнего и родительского клю- чей совпадают, они *не обязательно* должны быть одинаковыми, хотя часто их сознательно задают одинаковыми, чтобы соединение было более наглядным.

* + 1. **Составные внешние ключи.** Подобно первичному ключу, внешний ключ может состоять как из одного, так и из нескольких полей. Внешний ключ и родительский ключ, на который он ссылается, конечно же, должны быть определены на одинаковом множестве полей (по количеству полей, типам полей и порядку следования полей). Внешние ключи, состоящие из одного поля — применяемые в типовых таблицах настоящего издания — наиболее часты на практике. Чтобы сохранить простоту обсуждения, будем говорить о внешнем ключе, как об одиночном столбце, хотя все, что будет излагаться о поле, которое является внешним ключом, справедливо и для составных внешних ключей, определенных на группе полей.
    2. **Смысл внешнего и родительского ключей.** Когда поле является внешним ключом, оно определенным образом связано с таб- лицей, на которую этот ключ ссылается. Каждое значение в этом поле (внешнем ключе) непосредственно привязано к конкретному значению в другом поле (родительском ключе). Значения родительского ключа должны быть уникальными, так как он одновременно является ключом отношения. Значения внешнего ключа *не обязательно* должны быть уникальными, т.е. в отношении может быть любое число строк с одинаковым сочетанием значений атрибутов, составляющих внешний ключ. При этом строки, содержащие одинаковые значения внешнего ключа, должны *обязательно* ссылаться на конкретное, присутствую- щее в данный момент в таблице, значение родительского ключа или быть неопределенными (**NULL**). Ни в одной строке таблицы не должно быть значений внешнего ключа, для которых в текущий момент от-

сутствуют соответствующие значения родительского ключа. Другими словами, не должно быть так называемых “висячих” ссылок внешнего ключа. Если указанные требования выполняются в конкретный момент существования базы данных, то говорят, что данные находятся в *со- гласованном* состоянии, а сама база находится в состоянии *ссылочной целостности*.

* + 1. **Ограничение внешнего ключа (FOREIGN KEY).** Для ре- шения вопросов поддержания ссылочной целостности в SQL исполь- зуется ограничение **FOREIGN KEY**. Назначение **FOREIGN KEY** — это ограничение допустимых значений поля множеством значений роди- тельского ключа, ссылка на который указывается при описании данно- го ограничения **FOREIGN KEY**.

Проблемы обеспечения ссылочной целостности возникают как при вводе значений поля, являющегося внешним ключом, так и при модификации/удалении значений родительского ключа, т. е. по- ля, на которое ссылается этот ключ. Одно из действий ограничения **FOREIGN KEY** — это отклонение (блокировка) ввода значений внеш- него ключа, отсутствующих в таблице с родительским ключом. Это ограничение воздействует также на возможность изменять или удалять значения родительского ключа.

Ограничение **FOREIGN KEY** используется в командах **CREATE TABLE** и **ALTER TABLE** при создании или модификации таблицы, содержащей поле, которое требуется объявить внешним ключом. В команде указывается имя родительского ключа, на который имеется ссылка в ограничении **FOREIGN KEY**.

* + 1. **Внешний ключ как ограничение таблицы.** Синтаксис ограничения **FOREIGN KEY** имеет следующий вид:

**FOREIGN KEY** (*<список столбцов>*)

**REFERENCES** *<родительская таблица>*

[(*<родительский ключ>*)];

В этом предложении *<список столбцов>* — это список из одного или более столбцов таблицы, которые будут созданы или изменены ко- мандами **CREATE TABLE** или **ALTER TABLE** (должны быть отделены друг от друга запятыми). Параметр *<родительская таблица>* — это имя таблицы, содержащей родительский ключ. Это, в частности, может быть именем таблицы, которая создается или изменяется текущей командой. Параметр *<родительский ключ>* представляет собой спи- сок столбцов родительской таблицы, которые составляют собственно родительский ключ. Оба списка столбцов, определяющих внешний и родительский ключи, должны быть совместимы, а именно:

списки должны содержать одинаковое число столбцов; последовательность (1-й, 2-й, 3-й и т. д.) столбцов списка внеш- него ключа должны иметь типы данных и размеры, совпадающие

*•*

*•*

с соответствующими (1-м, 2-м, 3-м и т. д.) столбцами списка родительского ключа.

Создадим таблицу STUDENT с полем UNIV\_ID, определенным в качестве внешнего ключа, ссылающегося на таблицу UNIVERSITY:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25),

NAME **CHAR**(10), STIPEND **INTEGER**, KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**, UNIV\_ID **INTEGER**,

**CONSTRAINT** UNIV\_FOR\_KEY **FOREIGN KEY** (UNIV\_ID)

**REFERENCES** UNIVERSITY (UNIV\_ID));

Если ограничение **FOREIGN KEY** устанавливается в уже существу- ющей таблице с помощью оператора **ALTER TABLE**, то имеющиеся в этой таблице значения внешнего ключа и значения родительского клю- ча таблицы, на которую ссылается устанавливаемый внешний ключ, должны находиться в состоянии ссылочной целостности. В противном случае команда будет отклонена.

Синтаксис команды **ALTER TABLE** в этом случае имеет следующий вид:

**ALTER TABLE** *<имя таблицы>*

**ADD CONSTRAINT** *<имя ограничения>*

**FOREIGN KEY** (*<список столбцов внешнего ключа>*)

**REFERENCES** *<имя родительской таблицы>*

[(*<список столбцов родительского ключа>*)];

Например, команда

**ALTER TABLE** STUDENT

**ADD CONSTRAINT** STUD\_UNIV\_FOR\_KEY

**FOREIGN KEY** (UNIV\_ID)

**REFERENCES** UNIVERSITY (UNIV\_ID);

добавляет ограничение внешнего ключа для таблицы STUDENT.

* + 1. **Внешний ключ как ограничение столбцов.** Если опреде- ляемый внешний ключ не является составным, а состоит из единствен- ного столбца, то ограничение внешнего ключа может устанавливаться непосредственно в строке, описывающей этот столбец. При таком ва- рианте, называемом *ссылочным ограничением столбца*, ключевое сло- во **FOREIGN KEY** фактически не используется. Просто используется ключевое слово **REFERENCES**, и далее указывается имя родительского ключа, подобно следующему примеру:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25),

NAME **CHAR**(10), STIPEND **INTEGER**, KURS **INTEGER**, CITY **CHAR**(15), BIRTHDAY **DATE**,

UNIV\_ID **INTEGER REFERENCES** UNIVERSITY(UNIV\_ID));

Команда определяет поле STUDENT.UNIV\_ID как внешний ключ, использующий в качестве родительского ключа поле UNIVERSITY.UNIV\_ID, являющееся ключом таблицы UNIVERSITY.

Эта форма эквивалентна следующему ограничению таблицы

STUDENT:

**FOREIGN KEY** (UNIV\_ID)

**REGERENCES** UNIVERSITY (UNIV\_ID)

или, в другой записи,

**CONSTRAINT** UNIV\_FOR\_KEY **FOREIGN KEY** (UNIV\_ID)

**REFERENCES** UNIVERSITY (UNIV\_ID).

Если в родительской таблице у родительского ключа указано огра- ничение **PRIMARY KEY**, то при указании ограничения **FOREIGN KEY**, накладываемого на таблицу или на столбцы, *можно не указывать список столбцов родительского ключа*. Естественно, в случае исполь- зования ключей со многими полями порядок столбцов в соответству- ющих внешних и первичных ключах должен совпадать, и в любом случае должен быть соблюден принцип совместимости между двумя ключами.

Например, если ограничение **PRIMARY KEY** размещено в поле

UNIV\_ID таблицы UNIVERSITY:

**CREATE TABLE** UNIVERSITY

(UNIV\_ID **INTEGER PRIMARY KEY**, UNIV\_NAME **CHAR**(10),

RATING **INTEGER**, CITY **CHAR**(15));

то в таблице STUDENT поле UNIV\_ID можно использовать в качестве внешнего ключа, не указывая в ссылке имя родительского ключа:

**CREATE TABLE** STUDENT

(STUDENT\_ID **INTEGER PRIMARY KEY**, SURNAME **CHAR**(25),

NAME **CHAR**(10), STIPEND **INTEGER**, KURS **INTEGER**,

CITY **CHAR**(15), BIRTHDAY **DATE**,

UNIV\_ID **INTEGER REFERENCES** UNIVERSITY);

Такая возможность встроена в язык для обеспечения использования первичных ключей в качестве родительских.

* + 1. **Поддержание ссылочной целостности и ограничения значений родительского ключа.** Поддержание ссылочной целостно- сти требует выполнения некоторых ограничений на значения, которые могут быть заданы в полях, объявленных как внешний ключ и роди- тельский ключ. Набор значений родительского ключа должен быть та- ким, чтобы каждому значению внешнего ключа в родительской таблице обязательно соответствовала одна и только одна строка, указанная со- ответствующим родительским ключом. Это означает, что родительский ключ должен быть *уникальным*. Следовательно, при объявлении внеш- него ключа необходимо убедиться, что все поля, которые используются как родительские ключи, имеют или ограничение **PRIMARY KEY**, или ограничения **UNIQUE**.
    2. **Использование первичного ключа в качестве уникаль- ного внешнего ключа.** Ссылка внешних ключей только на первичные ключи считается хорошим стилем программирования SQL-запросов. В этом случае используемые внешние ключи связываются не просто с родительскими ключами, на которые они ссылаются, а с одной конкретной строкой родительской таблицы, в которой будет найдено соответствующее значение родительского ключа. Сам по себе роди- тельский ключ не обеспечивает никакой информации, которая бы не была уже представлена во внешнем ключе. Внешний ключ — это не просто связь между двумя идентичными значениями столбцов двух таблиц, но связь *между двумя строками двух таблиц*.

Так как назначение первичного ключа состоит именно в том, чтобы однозначно идентифицировать строку, то использование ссылки на него в качестве внешнего ключа является более логичным и более однозначным выбором для внешнего ключа. Внешний ключ, который не имеет никакой другой цели, кроме связывания строк, напоминает пер- вичный ключ, используемый исключительно для идентификации строк, и является хорошим средством сохранения наглядности и простоты структуры базы данных.

* + 1. **Ограничения значений внешнего ключа.** Внешний ключ может содержать только те значения, которые фактически представ- лены в родительском ключе или являются пустыми (**NULL**). Попытка ввести другие значения в этот ключ должна быть отклонена, поэтому объявление внешнего ключа как **NOT NULL** не является обязательным.
    2. **Действие ограничений внешнего и родительского клю- чей при использовании команд модификации.** Как уже говори- лось, при использовании команд **INSERT** и **UPDATE** для модификации

значений столбца, объявленного как *внешний ключ*, вновь вводимые значения должны уже быть обязательно представлены в фактически присутствующих значениях столбца, объявленного родительским клю- чом. При этом можно помещать в эти поля пустые (**NULL**) значе- ния, несмотря на то, что значения **NULL** недопустимы в родительских ключах. Можно также удалять (**DELETE**) любые строки с внешними ключами из таблицы, в которой эти ключи объявлены.

При необходимости модификации значений *родительского ключа* дело обстоит иначе. Использование команды **INSERT**, которая осу- ществляет ввод новой записи, не вызывает никаких особенностей, при которых возможно нарушение ссылочной целостности. Однако команда **UPDATE**, изменяющая значение родительского ключа, и ко- манда **DELETE**, удаляющая строку, содержащую такой ключ, содержат возможность нарушения согласованности значений родительского и ссылающихся на него внешних ключей. Например, может возникнуть так называемая “*висячая*” ссылка внешнего ключа на несуществующее значение родительского ключа, что совершенно недопустимо. Чтобы при применении команд **UPDATE** и **DELETE** к полю, являющемуся родительским ключом, не нарушалась целостность ссылки, возможны следующие варианты действий.

Любые изменения значений родительского ключа *запрещают- ся* и при попытке их совершения отвергаются (ограничение **NO ACTION** или **RESTRICT**). Эта спецификация действия применя- ется по умолчанию.

*•*

Изменения значений родительского ключа *разрешаются*, но при этом автоматически осуществляется *коррекция* всех значений внешних ключей, ссылающихся на модифицируемое значение родительского ключа. Это называется *каскадным изменением* (ограничение **CASCADE**).

*•*

Изменения значений родительского ключа *разрешаются*, но при этом соответствующие значения внешнего ключа автоматически *удаляются*, т. е. заменяются значением **NULL** (ограничение **SET NULL**).

*•*

Изменения значений родительского ключа *разрешаются*, но при этом соответствующие значения внешнего ключа автоматиче- ски *заменяются* значением по умолчанию (ограничение **SET DEFAULT**).

*•*

При описании внешнего ключа должно указываться, какой из приве- денных вариантов действий следует применять, причем в общем случае это должно быть указано раздельно для каждой из команд **UPDATE** и **DELETE**. В качестве примера использования ограничений, накладывае- мых на операции модификации родительских ключей, можно привести следующий запрос:

**CREATE TABLE** NEW\_EXAM\_MARKS (STUDENT\_ID **INTEGER NOT NULL**,

SUBJ\_ID **INTEGER NOT NULL**, MARK **INTEGER**,

DATA **DATE**, **CONSTRAINT** EXAM\_PR\_KEY

**PRIMARY KEY**(STUDENT\_ID, SUBJ\_ID),

**CONSTRAINT** SUBJ\_ID\_FOR\_KEY **FOREIGN KEY** (SUBJ\_ID)

**REFERENCES** SUBJECT,

**CONSTRAINT** STUDENT\_ID\_FOR\_KEY **FOREIGN KEY**(STUDENT\_ID) **REFERENCES** STUDENT **ON UPDATE CASCADE**

ON DELETE NO ACTION);

В этом примере при попытке изменения значения поля STUDENT\_ID таблицы STUDENT будет автоматически обеспечиваться каскадная корректировка этих значений в таблице EXAM\_MARKS, т. е. при изменении идентификатора студента STUDENT\_ID в таблице STUDENT сохранятся все ссылки на его оценки. Однако любая попытка удаления (**DELETE**) записи о студенте из таблицы STUDENT будет отвергаться, если в таблице EXAM\_MARKS существуют записи об оценках данного студента.

УПРАЖНЕНИЯ

1. Создайте таблицу с именем SUBJECT\_1, с теми же полями, что и в таблице SUBJECT (предмет обучения). Поле SUBJ\_ID является первичным ключом.
2. Создайте таблицу с именем SUBJ\_LECT\_1 (учебные дисци- плины преподавателей), с полями LECTURER\_ID (идентифика- тор преподавателя) и SUBJ\_ID (идентификатор предмета обу- чения). Первичным ключом (составным) таблицы является па- ра атрибутов LECTURER\_ID и SUBJ\_ID; кроме того, поле LECTURER\_ID является внешним ключом, ссылающимся на таб- лицу LECTURER\_1, аналогичную таблице LECTURER (преподава- тель), а поле SUBJ\_ID является внешним ключом, ссылающимся на таблицу SUBJECT\_1, аналогичную таблице SUBJECT.
3. Создайте таблицу с именем SUBJ\_LECT\_1 как в предыдущем задании, но добавьте для всех ее внешних ключей режим обеспе- чения ссылочной целостности, запрещающий обновление и уда- ление соответствующих родительских ключей.
4. Создайте таблицу с именем LECTURER\_1, с теми же полями, что и в таблице LECTURER. Первичным ключом таблицы является ат- рибут LECTURER\_ID; кроме того, поле UNIV\_ID является внеш- ним ключом, ссылающимся на таблицу UNIVERSITY\_1 (аналог UNIVERSITY). Для этого поля установите каскадные режимы обеспечения целостности для команд **UPDATE** и **DELETE**.
5. Создайте таблицу с именем UNIVERSITY\_1, с теми же полями, что и в таблице UNIVERSITY (университеты). Поле UNIV\_ID является первичным ключом.

96 *Гл. 4. Создание объектов базы данных*



1. Создайте таблицу с именем EXAM\_MARKS\_1. Она должна содер- жать те же поля, что и таблица EXAM\_MARKS (экзаменационные оценки). Комбинация полей EXAM\_ID, STUDENT\_ID и SUBJ\_ID является первичным ключом. Кроме того, поля STUDENT\_ID и SUBJ\_ID являются внешним ключами, ссылающимися соответ- ственно на таблицы STUDENT\_1 и SUBJECT\_1. Для этих полей установите режим каскадного обеспечения ссылочной целостно- сти при операции обновления соответствующих первичных клю- чей и режим блокировки операции удаления родительского ключа при наличии ссылки на него.
2. Создайте таблицу с именем STUDENT\_1. Она должна содер- жать те же поля, что и таблица STUDENT, и новое поле SENIOR\_STUDENT (староста), значением которого должен быть идентификатор студента, являющегося старостой группы, в ко- торой учится данный студент. Укажите необходимые для этого ограничения ссылочной целостности.
3. Создайте таблицу STUDENT\_2, аналогичную таблице STUDENT, в которой поле UNIV\_ID (идентификатор университета) являет- ся внешним ключом, ссылающимся на таблицу UNIVERSITY\_1, таким образом, чтобы при удалении из таблицы UNIVERSITY\_1 строки с информацией о каком-либо университете в соответству- ющих записях таблицы STUDENT\_2 поле UNIV\_ID очищалось (замещалось на **NULL**).
4. С помощью команды **CREATE TABLE** создайте запросы для фор- мирования таблиц учебной базы данных, представленной в разде- ле 1.7, с указанием первичных ключей, но без указания ограниче- ний внешних ключей. Затем с помощью команды **ALTER TABLE** укажите для сформированных таблиц все ограничения, в том числе и ограничения ссылочной целостности.

Г л а в а 5

**ПРЕДСТАВЛЕНИЯ (VIEW)**

* 1. Представления — именованные запросы

До сих пор речь шла о таблицах, обычно называемых базовыми таблицами. Это — таблицы, которые содержат данные. Однако имеет- ся и другой вид таблиц, называемый **VIEW** (ПРЕДСТАВЛЕНИЯ). Таб- лицы-представления не содержат никаких собственных данных. Фак- тически ***представление*** — *это* ***именованная*** *таблица*, *содержимое которой является результатом запроса*, *заданного при описании представления*, причем данный запрос выполняется всякий раз, когда таблица-представление становится объектом команды SQL. При этом в каждый момент вывод запроса становится содержанием представления. Представления позволяют:

ограничивать число столбцов, из которых пользователь выбирает или в которые вводит данные;

*•*

ограничивать число строк, из которых пользователь выбирает или в которые вводит данные;

*•*

выводить дополнительные столбцы, преобразованные из других столбцов базовой таблицы;

*•*

выводить группы строк таблицы.

*•*

Благодаря этому представления дают возможность гибкой настройки выводимой из таблиц информации в соответствии с требованиями кон- кретных пользователей, позволяют обеспечивать защиту информации на уровне строк и столбцов, упрощают формирование сложных отчетов и выходных форм.

Представление определяется с помощью команды **CREATE VIEW**

(СОЗДАТЬ ПРЕДСТАВЛЕНИЕ). Например:

**CREATE VIEW** MOSC\_STUD **AS SELECT** \*

**FROM** STUDENT

**WHERE** CITY = ‘Москва’;

4 И. Ф. Астахова, В. М. Мельников, А. П. Толстобров, В. В. Фертиков

98 *Гл. 5. Представления (****VIEW****)*



Данные из базовой таблицы, предъявляемые пользователю в представлении, зависят от условия (предиката), описанного в **SELECT**-запросе при определении представления.

В созданную в результате приведенного выше запроса таблицу- представление MOSC\_STUD передаются данные из базовой таблицы STUDENT, но не все, а только записи о студентах, для которых значение поля CITY равно ‘Москва’. К таблице MOSC\_STUD можно теперь обращаться с помощью запросов так же, как и к любой другой таблице базы данных. Например, запрос для просмотра представления MOSC\_STUD имеет вид:

SELECT \*

**FROM** MOSC\_STUD;

Представление, в котором выбираются *все* строки и столбцы базо- вой таблицы, например,

**CREATE VIEW** NEW\_STUD\_TAB **AS SELECT** \*

**FROM** STUDENT;

по сути эквивалентно применению синонима, но менее эффективно, поэтому применяется редко.

В следующем примере представление выбирает все строки и столб- цы; кроме того, в качестве имен столбцов применяются псевдонимы:

**CREATE VIEW** NEW\_STUDENT

(NEW\_STUDENT\_ID, NEW\_SURNAME, NEW\_NAME, NEW\_STIPEND, NEW\_KURS, NEW\_CITY, NEW\_BIRTHDAY, NEW\_UNIV\_ID)

**AS SELECT** STUDENT\_ID, SURNAME, NAME, STIPEND, KURS, CITY, BIRTHDAY, UNIV\_ID

**FROM** STUDENT;

Такое представление является простым способом организации об- щей таблицы для группы пользователей или прикладных задач, кото- рые используют собственные имена полей и таблицы.

* 1. Модификация представлений

Данные, предъявляемые пользователю через представление, могут изменяться с помощью команд модификации DML, но при этом факти- ческая модификация данных будет осуществляться не в само´й вирту- альной таблице-представлении, а будет перенаправлена к соответству- ющей базовой таблице. Например, запрос на обновление представления NEW\_STUDENT

**UPDATE** NEW\_STUDENT

**SET** CITY = ‘Москва’

* 1. *Маскирующие представления* 99



**WHERE** STUDENT\_ID = 1004;

эквивалентен выполнению команды **UPDATE** над базовой таблицей STUDENT. Следует, однако, обратить внимание на то, что в общем случае, из-за того, что обычно в представлении данные из базовой таб- лицы отображаются в *преобразованном* или *усеченном* виде, примене- ние команд модификации к таблицам-представлениям имеет некоторые особенности, рассматриваемые ниже.

5.3. Маскирующие представления

* + 1. **Представления, маскирующие столбцы.** Данный вид представлений ограничивает число столбцов базовой таблицы, к кото- рым возможен доступ. Например, представление

**CREATE VIEW** STUD **AS**

**SELECT** STUDENT\_ID, SURNAME, CITY

**FROM** STUDENT;

дает пользователю доступ к полям STUDENT\_ID, SURNAME, CITY ба- зовой таблицы STUDENT, полностью скрывая от него как содержимое, так и сам факт наличия в базовой таблице полей NAME, STIPEND, KURS, BIRTHDAY и UNIV\_ID.

* + 1. **Операции модификации в представлениях, маскирую- щих столбцы.** Представления, как уже отмечалось выше, могут изме- няться с помощью команд модификации DML, но при этом модифика- ция данных будет осуществляться не в самой таблице-представлении, а в соответствующей базовой таблице. Поэтому с представлениями, мас- кирующими столбцы, функции вставки и удаления работают несколько иначе, чем с обычными таблицами. Оператор **INSERT**, примененный к представлению, фактически осуществляет вставку строки в соот- ветствующую базовую таблицу, причем *во все* столбцы этой таблицы независимо от того, видны они пользователю через представление или скрыты от него. В связи с этим в столбцах, не включенных в представление, устанавливается **NULL**-значение или значение по умол- чанию. Если для не включенного в представление столбца действует ограничение **NOT NULL** а значение по умолчанию не определено, то генерируется сообщение об ошибке.

Любое применение оператора **DELETE** удаляет строки базовой таб- лицы независимо от их значений.

* + 1. **Представления, маскирующие строки.** Представления могут также ограничивать доступ к строкам. Охватываемые представ- лением строки базовой таблицы задаются условием (предикатом) в конструкции **WHERE** при описании представления. Доступ через пред- ставление возможен только к строкам, удовлетворяющим условию.

Например, представление

4\*

100 *Гл. 5. Представления (****VIEW****)*



**CREATE VIEW** MOSC\_STUD **AS SELECT** \*

**FROM** STUDENT

**WHERE** CITY = ‘Москва’;

показывает пользователю только те строки таблицы STUDENT, для которых значение поля CITY равно ‘Москва’.

* + 1. **Операции модификации в представлениях, маскирую- щих строки.** Каждая включенная в представление строка доступна для вывода, обновления и удаления. Любая допустимая для базовой таблицы строка вставляется в базовую таблицу независимо от ее вклю- чения в представление. При этом может возникнуть проблема, состоя- щая в том, что значения, введенные пользователем в базовую таблицу через представление значений, будут отсутствовать в представлении, оставаясь при этом в базовой таблице. Рассмотрим такой случай:

**CREATE VIEW** HIGH\_RATING **AS SELECT** \*

**FROM** UNIVERSITY

**WHERE** RATING = 300;

Это представление является обновляемым. Оно просто ограничива- ет доступ пользователя к определенным столбцам и строкам в таблице UNIVERSITY. Предположим, необходимо вставить с помощью команды **INSERT** следующую строку:

**INSERT INTO** HIGH\_RATING

**VALUES** (180, ‘Новый университет’, 200, ‘Воронеж’);

Команда **INSERT** допустима в этом представлении. С помощью представления HIGH\_RATING строка будет вставлена в базовую таб- лицу UNIVERSITY. Однако после появления этой строки в базовой таблице *из самого представления она исчезнет*, поскольку значение поля RATING не равно 300, и, следовательно, эта строка не удовлетво- ряет условию предложения **WHERE** для отбора строк в представление. Для пользователя такое исчезновение только что введенной строки является неожиданным. Действительно, непонятно, почему после ввода строки в таблицу ее нельзя увидеть и, например, тут же удалить. Тем более, что пользователь может вообще не знать, работает ли он в данный момент с базовой таблицей или с таблицей-представлением.

Аналогичная ситуация возникнет, если в какой-либо существующей записи представления HIGH\_RATING изменить значение поля RATING на значение, отличное от 300.

Подобные проблемы можно устранить путем включения в определе- ние представления опции **WITH CHECK OPTION**. Эта опция распростра- няет условие **WHERE** для запроса на операции обновления и вставки в описание представления. Например:

*5.3. Маскирующие представления* 101



**CREATE VIEW** HIGH\_RATUNG **AS SELECT** \*

**FROM** UNIVERSITY

**WHERE** RATING = 300

WITH CHECK OPTION;

В этом случае вышеупомянутые операции вставки строки или кор- рекции поля RATING будет отклонены.

Опция **WITH CHECK OPTION** помещается в определение представле- ния, а не в команду **DML**, так что *все* команды модификации в представ- лении будут проверяться. Рекомендуется использовать эту опцию во всех случаях, когда нет причины разрешать представлению помещать в таблицу значения, которые в нем самом не могут быть видны.

* + 1. **Операции модификации в представлениях, маскирую- щих строки и столбцы.** Рассмотренная выше проблема возникает и при вставке строк в представление с предикатом, использующим поля базовой таблицы, не присутствующие в самом представлении. Например, рассмотрим представление

**CREATE VIEW** MOSC\_STUD **AS**

**SELECT** STUDENT\_ID, SURNAME, STIPEND

**FROM** STUDENT

**WHERE** CITY = ‘Москва’;

Видно, что в данное представление не включено поле CITY таблицы

STUDENT.

При попытке вставки строки в это представление операция не будет выполнена, если столбец CITY в таблице STUDENT определен с огра- ничением **NOT NULL** и без значения по умолчанию. Если для столбца CITY определено значение по умолчанию, отличное от **NULL**, то это значение и попадет в столбец CITY. Так как в этом случае значение поля CITY базовой таблицы STUDENT не будет равняться значению ‘Москва’, вставляемая строка будет исключена из самого представле- ния и поэтому не будет видна пользователю. Это будет происходить для *любой* вставляемой в представление MOSC\_STUD строки. Другими словами, пользователь вообще не сможет видеть строки, вводимые им в это представление. Данная проблема не решается и в том случае, когда в определение представления добавляется опция **WITH CHECK OPTION**, если только для столбца CITY не определено значение по умолчанию ‘Москва’:

**CREATE VIEW** MOSC\_STUD **AS**

**SELECT** STUDENT\_ID, SURNAME, STIPEND

**FROM** STUDENT

**WHERE** CITY = ‘Москва’

WITH CHECK OPTION;

102 *Гл. 5. Представления (****VIEW****)*



Таким образом, в определенном указанными способами представ- лении можно модифицировать значения полей или удалять строки, но нельзя *вставлять* строки. Исходя из этого, рекомендуется даже в тех случаях, когда этого не требуется по соображениям полезности (и даже безопасности) информации, при определении представления включать в него *все* поля, на которые имеется ссылка в предикате. Если эти поля не должны отображаться в выводе таблицы, всегда можно исключить их уже в запросе к представлению. Другими словами, можно было бы определить представление MOSC\_STUD, например, так:

**CREATE VIEW** MOSC\_STUD **AS SELECT** \*

**FROM** STUDENT

**WHERE** CITY = ‘Москва’

WITH CHECK OPTION;

Эта команда заполнит в представлении поле CITY одинаковыми значениями, которые можно просто исключить из вывода с помощью другого запроса уже к этому сформированному представлению, указав в запросе только поля, необходимые для вывода:

**SELECT** STUDENT\_ID, SURNAME, STIPEND

**FROM** MOSC\_STUD;

* 1. Агрегированные представления

Создание представлений с использованием агрегированных функ- ций и предложения **GROUP BY** является удобным инструментом для непрерывной обработки и интерпретации извлекаемой информации. Предположим, необходимо следить за количеством студентов, сдаю- щих экзамены, количеством сданных экзаменов, количеством сданных предметов, средним баллом по каждому предмету. Для этого можно сформировать следующее представление:

**CREATE VIEW** TOTALDAY **AS**

**SELECT** EXAM\_DATE, **COUNT**(**DISTINCT** SUBJ\_ID)

**AS** SUBJ\_CNT, **COUNT**(STUDENT\_ID) **AS** STUD\_CNT, **COUNT**(MARK) **AS** MARK\_CNT,

**AVG**(MARK) **AS** MARK\_AVG, **SUM**(MARK) **AS** MARK\_SUM

**FROM** EXAM\_MARKS

**GROUP BY** EXAM\_DATE;

Теперь требуемую информацию можно увидеть с помощью простого запроса к представлению:

**SELECT** \* **FROM** TOTALDAY;

*5.6. Представления и подзапросы* 103



* 1. Представления, основанные на нескольких таблицах

Представления часто используются для объединения нескольких таблиц (базовых и/или других представлений) в одну большую вирту- альную таблицу. Такое решение имеет ряд преимуществ:

представление, объединяющее несколько таблиц, при формирова- нии сложных отчетов может использоваться как промежуточный макет, скрывающий детали объединения большого количества исходных таблиц;

*•*

предварительно объединенные поисковые и базовые таблицы обеспечивают наилучшие условия для транзакций, позволяют ис- пользовать компактные схемы кодов, устраняя необходимость на- писания длинных объединяющих процедур для каждого отчета; позволяет использовать при формировании отчетов более надеж- ный модульный подход;

*•*

*•*

предварительно объединенные и проверенные представления уменьшают вероятность ошибок, связанных с неполным выполнением условий объединения.

*•*

Можно, например, создать представление, которое показывает име- на и названия сданных предметов для каждого студента:

**CREATE VIEW** STUD\_SUBJ **AS**

**SELECT** A.STUDENT\_ID, C.SUBJ\_ID, A.SURNAME, C.SUBJ\_NAME

**FROM** STUDENT A, EXAM\_MARKS B, SUBJECT C

**WHERE** A.STUDENT\_ID = B.STUDENT\_ID

**AND** B.SUBJ\_ID = C.SUBJ\_ID;

Теперь все предметы, сданные студентом, или всех студентов, сдав- ших данный предмет, можно выбрать с помощью простого запроса. Например, чтобы увидеть все предметы, сданные студентом Ивановым, подается запрос:

**SELECT** SUBJ\_NAME

**FROM** STUD\_SUBJ

**WHERE** SURNAME = ‘Иванов’;

* 1. Представления и подзапросы

При создании представлений могут также использоваться подзапро- сы, включая связанные подзапросы. Предположим, предусматривается премия для тех студентов, которые имеют самый высокий балл на любую заданную дату. Получить такую информацию можно с помощью представления:

104 *Гл. 5. Представления (****VIEW****)*



**CREATE VIEW** ELITE\_STUD

**AS SELECT** B.EXAM\_DATE, A.STUDENT\_ID, A.SURNAME

**FROM** STUDENT A, EXAM\_MARKS B **WHERE**A.STUDENT\_ID = B.STUDENT\_ID **AND** B.MARK =

(**SELECT MAX (**MARK**) FROM** EXAM\_MARKS C

**WHERE** C.EXAM\_DATE = B.EXAM\_DATE);

Если, с другой стороны, премия будет назначаться только студенту, который имел самый высокий балл, причем не меньше 10 раз, то необходимо использовать другое представление, основанное на первом:

**CREATE VIEW** BONUS

**AS SELECT DISTINCT** STUDENT\_ID, SURNAME

**FROM** ELITE\_STUD A

**WHERE** 10 <=

(**SELECT COUNT**(\*) **FROM** ELITE\_STUD B

**WHERE** A.STUDENT\_ID = B.STUDENT\_ID);

Извлечение из этой таблицы записей о студентах, которые будут получать премию, выполняется простым запросом:

SELECT \* FROM BONUS;

* 1. Удаление представлений

Синтаксис удаления представления из базы данных подобен син- таксису удаления базовых таблиц:

**DROP VIEW** *<имя представления>*

УПРАЖНЕНИЯ

1. Создайте представление для получения сведений обо всех сту- дентах, имеющих только отличные оценки.
2. Создайте представление для получения сведений о количестве студентов в каждом городе.
3. Создайте представление для получения следующих сведений по каждому студенту: его идентификатор, фамилия, имя, средний и общий баллы.
4. Создайте представление для получения сведений о количестве экзаменов, которые сдавал каждый студент.
   1. *Изменение значений в представлениях* 105



* 1. Изменение значений в представлениях

Как уже говорилось, использование команд модификации язы- ка SQL — **INSERT** (ВСТАВИТЬ), **UPDATE** (ЗАМЕНИТЬ), и **DELETE**

(УДАЛИТЬ) — применительно к представлениям имеет ряд особенно- стей. В дополнение к аспектам, рассмотренным выше, следует отме- тить, что не все представления могут модифицироваться.

Если в представлении могут выполняться команды модификации, то представление является обновляемым (модифицируемым); в против- ном случае оно предназначено только для чтения при запросе. Каким образом можно определить, является ли представление модифицируе- мым? Критерии обновляемости представления можно сформулировать следующим образом.

Представление строится на основе одной и только одной базовой таблицы.

*•*

Представление должно содержать первичный ключ базовой таб- лицы.

*•*

Представление не должно иметь никаких полей, которые пред- ставляют собой агрегирующие функции.

*•*

Представление не должно содержать **DISTINCT** в своем опреде- лении.

*•*

Представление не должно использовать **GROUP BY** или **HAVING**

*•*

в своем определении.

Представление не должно использовать подзапросы. Представление *может быть* использовано в другом представле- нии, но это представление должно быть также модифицируемым. Представление не должно использовать в качестве полей вывода константы или выражения значений.

*•*

*•*

*•*

Суть этих ограничений в том, что обновляемые представления фактически подобны окнам в базовых таблицах. Они показывают ин- формацию из базовой таблицы, ограничивая определенные ее строки (использованием соответствующих предикатов) или специально име- нованные столбцы (с исключениями). Но при этом представления выводят значения *без их обработки* с использованием агрегирующих функций и группировки. Они также не сравнивают строки таблиц друг с другом (как это имеет место в объединениях и подзапросах или при использовании **DISTINCT**).

Различия между модифицируемыми (обновляемыми) представлени- ями и представлениями “*только для чтения*” не случайны. Обновляе- мые представления в основном используются аналогично базовым таб- лицам. Пользователи могут даже не знать, является ли запрашиваемый ими объект базовой таблицей или представлением. Это превосходный механизм защиты для скрытия частей таблицы, которые конфиденци- альны или не предназначены данному пользователю.

106 *Гл. 5. Представления (****VIEW****)*



Немодифицируемые представления, с другой стороны, позволяют более рационально получать и переформатировать данные. С их помо- щью формируются библиотеки сложных запросов, которые могут затем использоваться в запросах для получения информации самостоятельно (например, в объединениях). Эти представления могут также иметь значение при решении задач защиты и безопасности данных. Напри- мер, можно предоставить некоторым пользователям возможность по- лучения агрегатных данных (таких, как усредненное значение оценки студента), не показывая конкретных значений оценок и тем более не позволяя их модифицировать.

* 1. Примеры обновляемых и необновляемых представлений

Пример 1.

**CREATE VIEW** DATEEXAM (EXAM\_DATE, QUANTITY)

**AS SELECT** EXAM\_DATE, **COUNT** (\*)

**FROM** EXAM\_MARKS

**GROUP BY** EXAM\_DATE;

Данное представление — *необновляемое* из-за присутствия в нем агрегирующей функции и **GROUP BY**.

Пример 2.

**CREATE VIEW** LCUSTT

AS SELECT \*

**FROM** UNIVERSITY

**WHERE** CITY = ‘Москва’;

Это — *обновляемое* представление.

Пример 3.

**CREATE VIEW** SSTUD (SURNAME1, NUMB, KUR)

**AS SELECT** SURNAME, STUDENT\_ID, KURS\*2

**FROM** STUDENT

**WHERE** CITY = ‘Москва’;

Это представление — *немодифицируемое* из-за наличия выражения “KURS\*2”.

Пример 4.

**CREATE VIEW** STUD3

AS SELECT \*

**FROM** STUDENT

**WHERE** STUDENT\_ID **IN**

*5.9. Примеры обновляемых и необновляемых представлений* 107



(**SELECT** MARK

**FROM** EXAM\_MARKS

**WHERE** EXAM\_DATE = ‘10/02/1999’);

Из-за наличия подзапроса представление лучше отнести к *немоди- фицируемым*, хотя некоторые СУБД могут допускать его обновление.

Пример 5.

**CREATE VIEW** SOMEMARK

**AS SELECT** STUDENT\_ID, SUBJ\_ID, MARK

**FROM** EXAM\_MARKS

**WHERE** EXAM\_DATE

**IN** (‘10/02/1999’, ‘10/06/1999’);

Это — *обновляемое* представление.

УПРАЖНЕНИЯ

1. Какие из представленных ниже представлений являются обнов- ляемыми?

а) **CREATE VIEW** DAILYEXAM **AS**

**SELECT DISTINCT** STUDENT\_ID, SUBJ\_ID, MARK, EXAM\_DATE

**FROM** EXAM\_MARKS;

б) **CREATE VIEW** CUSTALS **AS**

**SELECT** SUBJECT.SUBJ\_ID, **SUM** (MARK) **AS** MARK1

**FROM** SUBJECT, EXAM\_MARKS

**WHERE** SUBJECT.SUBJ\_ID = EXAM\_MARKS.SUBJ\_ID

**GROUP BY** SUBJECT.SUBJ\_ID;

в) **CREATE VIEW** THIRDEXAM **AS SELECT** \*

**FROM** DAILYEXAM

**WHERE** EXAM\_DATE = ‘10/02/1999’;

г) **CREATE VIEW** NULLCITIES

**AS SELECT** STUDENT\_ID, SURNAME, CITY

**FROM** STUDENT

**WHERE** CITY **IS NULL**

**OR** SURNAME **BETWEEN** ‘А’ **AND** ‘Д’;

1. Создайте представление таблицы STUDENT с именем STIP, вклю- чающее поля STIPEND и STUDENT\_ID и позволяющее вводить или изменять значение поля STIPEND (стипендия), но только в пределах от 100 до 200.

*6.3. Команда* ***GRANT*** 109



Г л а в а 6 **ОПРЕДЕЛЕНИЕ ПРАВ ДОСТУПА ПОЛЬЗОВАТЕЛЕЙ К ДАННЫМ**

* 1. Пользователи и привилегии

Каждый, кто имеет доступ к базе данных, называется *пользовате- лем*. SQL используется обычно в многопользовательских средах, кото- рые требуют разграничения прав пользователей с точки зрения доступа к данным и прав на выполнение с ними тех или иных манипуляций. Для этих целей в SQL реализованы средства, позволяющие устанавли- вать и контролировать привилегии пользователей базы данных.

Каждый пользователь в среде SQL имеет специальное имя, или идентификатор, с помощью которого осуществляется идентификация пользователя для установки и определения его прав с точки зрения доступа к данным. Каждая посланная к СУБД команда SQL-запроса ассоциируется СУБД с идентификатором и правами доступа соответ- ствующего пользователя.

Пользователь может быть определен с помощью следующего опера- тора:

**CREATE USER** *<имя\_пользователя>*

**IDENTIFIED BY** *<пароль>*;

После выполнения этого оператора пользователь становится изве- стен базе данных, но пока не может выполнять никаких операций.

Для удаления пользователя используется оператор

**DROP USER** *<имя\_пользователя>*;

Назначаемые пользователю привилегии — это то, что определяет, может ли указанный пользователь выполнить данную команду над определенным объектом базы данных или нет. Имеется несколько типов привилегий, соответствующих нескольким типам операций. При- вилегии даются и отменяются двумя командами SQL, соответственно:

**GRANT** — установка привилегий,

**REVOKE** — отмена привилегий.

* 1. Стандартные привилегии

Привилегии, определенные SQL, — это привилегии объекта. Это означает, что пользователь имеет привилегию (право) на выполнение данной команды только на определенном объекте в базе данных. Привилегии объекта связаны одновременно и с пользователями, и с таблицами базы данных, т.е. привилегия дается определенному поль- зователю в указанной таблице. Это может быть как базовая таблица, так и представление.

Пользователь, создавший таблицу (любого вида), является *владель- цем* этой таблицы. Это означает, что этот пользователь имеет *все* при- вилегии, относящиеся к этой таблице, в том числе он может передавать привилегии на работу с этой таблицей другим пользователям.

Пользователю могут быть назначены следующие привилегии: **SELECT** — пользователь может выполнять запросы к таблице; **INSERT** — пользователь может выполнять в таблице команду

*•*

*•*

INSERT;

**UPDATE** — пользователь может выполнять в таблице команду **UPDATE**. Эта привилегия может быть ограничена для определен- ных столбцов таблицы;

*•*

**DELETE** — пользователь может выполнять в таблице команду

*•*

DELETE;

**REFERENCES** — пользователь может определить внешний ключ, который использует в качестве родительского ключа один или более столбцов этой таблицы. Возможно ограничение этой при- вилегии для определенных столбцов.

*•*

Кроме того, могут быть нестандартные привилегии объекта, такие, как: **INDEX** — пользователь имеет право создавать индекс в таблице; **SYNONYM** — пользователь имеет право создавать синоним для

*•*

*•*

объекта;

**ALTER** — пользователь имеет право выполнять команду **ALTER TABLE** в таблице;

*•*

**EXECUTE** — позволяет выполнять процедуру.

*•*

Назначение пользователям этих привилегий осуществляется с по- мощью команды **GRANT**.

* 1. Команда GRANT

Пользователь, являющийся владельцем таблицы STUDENT, может передать другому пользователю (пусть это будет пользователь с име- нем IVANOV) привилегию **SELECT** с помощью следующей команды.

**GRANT SELECT ON** STUDENT **TO** IVANOV;

Теперь пользователь с именем IVANOV может выполнять

**SELECT**-запросы к таблице STUDENT. При отсутствии других приви-

легий он может только *выбирать* значения, но не может выполнять никаких действий, которые воздействовали бы на значения в таблице STUDENT, включая ее использование в качестве родительской таблицы внешнего ключа. Когда SQL получает команду **GRANT**, проверяются привилегии пользователя, давшего эту команду, чтобы определить допустимость команды **GRANT** для этого пользователя. Пользователь IVANOV самостоятельно не может задать эту команду. Он также не может предоставить право **SELECT** другому пользователю, так как таблица принадлежит не ему (ниже будет показано, как владелец таблицы может передать другому пользователю право предоставления привилегий).

Команда

**GRANT INSERT ON** EXAM\_MARKS **TO** IVANOV;

предоставляет пользователю IVANOV право вводить в таблицу

EXAM\_MARKS новые строки.

В команде **GRANT** допустимо указывать через запятые список предо- ставляемых привилегий и список пользователей, которым они предо- ставляются. Например:

**GRANT SELECT**, **INSERT ON** SUBJECT **TO** IVANOV, PETROV;

При этом все указанные в списке привилегии предоставляются всем указанным пользователям. В строгой ANSI-интерпретации невозмож- но предоставить привилегии для нескольких таблиц одной командой **GRANT**.

* 1. Использование аргументов ALL и PUBLIC

Аргумент **ALL PRIVILEGES** (все привилегии) или просто **ALL** ис- пользуется вместо имен привилегий в команде **GRANT**, чтобы предоста- вить все привилегии в таблице. Например, команда

**GRANT ALL PRIVILEGES ON** STUDENT **TO** IVANOV;

или более коротко:

**GRANT ALL ON** STUDENT **TO** IVANOV;

передает пользователю IVANOV ***весь*** набор привилегий в таблице

STUDENT.

Аргумент **PUBLIC** используется для передачи указанных в команде привилегий *всем* остальным пользователям. Наиболее часто это при- меняется для привилегии **SELECT** в определенных базовых таблицах или представлениях, которые необходимо сделать доступными для лю- бого пользователя. Например, чтобы позволить любому пользователю получать информацию из таблицы EXAM\_MARKS, можно использовать команду

**GRANT SELECT ON** EXAM\_MARKS **TO PUBLIC**;

Предоставление *всех* привилегий к таблице *всем* пользователям обычно является нежелательным. Все привилегии, за исключением **SELECT**, позволяют пользователю изменять содержание таблицы, по- этому разрешение всем пользователям изменять содержание таблиц может вызвать определенные проблемы обеспечения безопасности и защиты данных. Тем более, что привилегия **PUBLIC** не ограничена в передаче прав только текущим пользователям. Любой новый пользова- тель, добавляемый к системе, автоматически получает в этом случае полный набор привилегий, назначенный ранее всем пользователям. Поэтому для ограничения доступа к таблице всем и всегда лучше всего предоставить привилегии, отличные от **SELECT**, только индивидуаль- ным пользователям.

* 1. Отмена привилегий

Отмена привилегии осуществляется с помощью команды **REVOKE** , которая имеет синтаксис, аналогичный команде **GRANT**.

Например, команда

**REVOKE INSERT ON** STUDENT **FROM** PETROV;

отменяет привилегию **INSERT** в таблице STUDENT для пользователя PETROV. Возможно использование в команде **REVOKE** списков приви- легий и пользователей. Например:

**REVOKE INSERT**, **DELETE ON** STUDENT

**FROM** PETROV, SIDOROV;

Следует иметь в виду, что привилегии отменяются тем пользо- вателем, который их предоставил, и при этом отмена автоматически распространяется на всех пользователей, получивших от него эту при- вилегию.

* 1. Использование представлений для фильтрации привилегий

Действия привилегий можно сделать более точными, используя представления. Привилегия, передаваемая пользователю в базовой таб- лице, автоматически распространяется на все строки, а при исполь- зовании возможных исключений **UPDATE** и **REFERENCES** — и на все столбцы таблицы. Создавая представление, которое ссылается на базо- вую таблицу, и затем передавая привилегию уже на это представление, можно ограничить эти привилегии любыми выражениями в запросе, содержащемся в представлении. Такой метод расширяет возможности команды **GRANT**.

Для создания представлений пользователь должен обладать приви- легией **SELECT** во всех таблицах, на которые он ссылается в представ- лении. Если представление модифицируемое, то любая из привилегий **INSERT**, **UPDATE** и **DELETE** , которая предоставлена пользователю в базовой таблице, будет автоматически распространяться на представ- ление. Если привилегии на обновление отсутствуют, то их невозможно получить и в созданных представлениях, даже если сами эти пред- ставления обновляемые. Так как внешние ключи не применяются в представлениях, то и привилегия **REFERENCES** никогда не использует- ся при создании представлений.

* + 1. **Ограничение привилегии SELECT для определенных столбцов.** Предположим, необходимо обеспечить пользователю PETROV возможность доступа только к столбцам STUDENT\_ID и SURNAME таблицы STUDENT. Это можно сделать, поместив имена этих столбцов в представление

**CREATE VIEW** STUDENT\_VIEW **AS SELECT** STUDENT\_ID, SURNAME **FROM** STUDENT;

и предоставив пользователю PETROV привилегию **SELECT** в созданном представлении, а не в самой таблице STUDENT:

**GRANT SELECT ON** STUDENT\_VIEW **TO** PETROV;

Для столбцов можно создать различные привилегии, однако следует иметь в виду, что для команды **INSERT** это будет означать вставку значений по умолчанию, а для команды **DELETE** ограничение столбца вообще не будет иметь значения.

* + 1. **Ограничение привилегий для определенных строк.** Представления позволяют ограничивать (фильтровать) привилегии для определенных строк таблицы. Для этого естественно использовать в представлении предикат, который определит, какие строки включены в представление. Чтобы предоставить пользователю PETROV привилегию вида **UPDATE** в таблице UNIVERSITY для всех записей о московских университетах, можно создать следующее представление:

**CREATE VIEW** MOSC\_UNIVERSITY **AS SELECT** \* **FROM** UNIVERSITY **WHERE** CITY = ‘Москва’

WITH CHECK OPTION;

Затем можно передать привилегию **UPDATE** в этой таблице пользо- вателю PETROV:

**GRANT UPDATE ON** MOSC\_UNIVERSITY **TO** PETROV;

В отличие от привилегии **UPDATE** для определенных столбцов, которая распространена на все строки таблицы UNIVERSITY, данная

привилегия относится только к строкам, для которых значение поля CITY равно ‘Москва’. Предложение **WITH CHECK OPTION** предохра- няет пользователя PETROV от замены значения поля CITY на любое значение, кроме значения ‘Москва’.

* + 1. **Предоставление доступа только к извлеченным данным.** Другая возможность состоит в том, чтобы устанавливать пользова- телям привилегии на доступ к уже извлеченным данным, а не к значениям в таблице. Для этого удобно использовать агрегирующие функции. Например, создадим представление, которое дает информа- цию о количестве оценок, среднем и общем баллах для студентов на каждый день:

**CREATE VIEW** DATETOTALS **AS**

**SELECT** EXAM\_DATE, **COUNT** (\*) **AS** KOL,

**SUM** (MARK) **AS** SUMMA, **AVG** (MARK) **AS** TOT

**FROM** EXAM\_MARKS

**GROUP BY** EXAM\_DATE;

Теперь можно передать пользователю PETROV привилегию **SELECT**

в созданном представлении DATETOTALS с помощью запроса:

**GRANT SELECT ON** DATETOTALS **TO** PETROV;

* + 1. **Использование представлений в качестве альтернативы ограничениям.** Представления с **WITH CHECK OPTION** могут исполь- зоваться в качестве альтернативы ограничениям. Например, необходи- мо удостовериться, что все значения поля CITY в таблице STUDENT равны названиям конкретных городов. Для этого можно установить ограничение **CHECK** непосредственно на столбец CITY. Однако позже его изменение будет затруднено. В качестве альтернативы можно со- здать представление, исключающее неправильные значения CITY:

**CREATE VIEW** CURCITYES **AS SELECT** \*

**FROM** STUDENT

**WHERE** CITY **IN** (‘Москва’, ‘Воронеж’)

WITH CHECK OPTION;

Теперь вместо того, чтобы предоставлять пользователям привиле- гии обновления в таблице STUDENT, можно предоставить соответству- ющие привилегии в представлении CURCITYES. Преимущество такого подхода состоит в том, что при необходимости изменения можно уда- лить это представление, создать новое и предоставить в этом новом представлении привилегии пользователям. Такая операция выполняет- ся проще, чем изменение ограничений в таблице. Недостатком этого метода является то, что владелец таблицы STUDENT тоже должен использовать это новое представление, иначе его собственные команды также не будут приняты.

114 *Гл. 6. Определение прав доступа пользователей к данным*



* 1. Другие типы привилегий

До сих пор не рассмотрены вопросы установки целого ряда других привилегий, а именно:

Кто имеет право создавать таблицы?

*•*

Кто имеет право изменять, удалять или ограничивать таблицы? Должны ли права создания базовых таблиц отличаться от прав создания представлений?

*•*

*•*

Должен ли существовать *суперпользователь*, т.е. пользователь, отвечающий за поддержание базы данных и, следовательно, име- ющий наибольшие, или полные привилегии, которые не предо- ставляются обычному пользователю?

*•*

Привилегии, которые не определяются в терминах специальных объектов данных, называются *привилегиями системы*, или *права- ми базы данных*. Эти привилегии включают в себя право создавать объекты данных, отличающиеся от базовых таблиц (обычно создавае- мых несколькими пользователями) и представлений (обычно создавае- мых большинством пользователей). Привилегии системы для создания представлений должны дополнять, а не заменять привилегии объек- та, которые стандарт требует от создателей представлений (описаны ранее). Кроме того, в любой системе всегда имеются некоторые типы суперпользователей, т.е. пользователей, которые имеют большинство или все привилегии и могут передать свой статус суперпользователя кому-либо с помощью привилегии или группы привилегий. Такого рода пользователем является так называемый *администратор базы данных*, или DBA (Database Administrator).

* 1. Типичные привилегии системы

При общем подходе имеются три базовых привилегии системы:

* **CONNECT** (ПОДКЛЮЧИТЬ),
* **RESOURCE** (РЕСУРС),
* **DBA** (АДМИНИСТРАТОР БАЗЫ ДАННЫХ).

Привилегия **CONNECT** состоит из права зарегистрироваться и пра- ва создавать представления и синонимы, если переданы привилегии объекта.

Привилегия **RESOURCE** состоит из права создавать базовые таблицы.

Привилегия **DBA** — это привилегия администратора базы данных, т.е. суперпользователя, которому предоставляются самые высокие пол- номочия при работе с базой данных. Эту привилегию могут иметь пользователи с функциями администратора базы данных. Команда **GRANT** (в измененной форме) может применяться как с привилегиями объекта, так и с системными привилегиями.

* 1. *Создание и удаление пользователей* 115



* 1. Создание и удаление пользователей

В большинстве реализаций SQL нового пользователя создает пользователь с привилегией **DBA**, т.е. администратор базы данных, который автоматически предоставляет новому пользователю приви- легию **CONNECT**. В этом случае обычно добавляется предложение **IDENTIFIED BY**, указывающее пароль для этого пользователя. Напри- мер, команда

**GRANT CONNECT TO** PETROV

**IDENTIFIED BY** ‘PETROVPASSWORD’;

приведет к созданию пользователя с именем PETROV, предоставит ему право регистрироваться в базе данных, и назначит ему пароль ‘PETROVPASSWORD’. После этого, так как PETROV уже является зарегистрированным пользователем, он (или пользователь **DBA**) мо- жет использовать эту же команду для изменения данного пароля ‘PETROVPASSWORD’.

Когда пользователь A предоставляет привилегию **CONNECT** другому пользователю B, говорят, что пользователь A “*создает*” пользовате- ля B. При этом пользователь A обязательно должен иметь привиле- гию **DBA**. Если пользователь B будет создавать базовые таблицы (а не только представления), то ему также должна быть предоставлена привилегия **RESOURCE**. Но при этом возникает другая проблема. При попытке удаления пользователем A привилегии **CONNECT** пользователя B, который уже имеет созданные им таблицы, эта команда удаления привилегии будет отклонена, поскольку ее действие оставит эти табли- цы без владельца, что не допускается. Поэтому, прежде чем удалить привилегию **CONNECT** у какого-либо пользователя, сначала необходимо удалить из базы данных все созданные этим пользователем таблицы. Привилегию **RESOURCE** удалять отдельно не требуется, достаточно удалить **CONNECT**, чтобы удалить пользователя.

УПРАЖНЕНИЯ

1. Передайте пользователю PETROV право на изменение оценок студентов в базе данных.
2. Передайте пользователю SIDOROV право передавать другим пользователям права на осуществление запросов к таблице EXAM\_MARKS.
3. Отмените привилегию **INSERT** по отношению к таблице STUDENT у пользователя IVANOV и у всех других пользователей, которым привилегия, в свою очередь, была предоставлена этим пользова- телем IVANOV.
4. Передайте пользователю SIDOROV право выполнять операции вставки или обновления в таблице UNIVERSITY, но только для записей об университетах, значения рейтингов которых лежат в диапазоне от 300 до 400.

116 *Гл. 6. Определение прав доступа пользователей к данным*



1. Разрешите пользователю PETROV делать запросы к таблице EXAM\_MARKS, но запретите ему изменять в этой таблице зна- чения оценок студентам, имеющим неудовлетворительные (=2) оценки.
   1. Создание синонимов (SYNONYM)

Каждый раз при ссылке к базовой таблице или представлению, не являющимся собственностью пользователя, требуется установить в качестве префикса к имени этой таблицы имя ее владельца, так как у разных пользователей могут оказаться таблицы с одинаковыми именами и в этом случае система не сможет определить местонахож- дение таблицы. Использование длинных имен с префиксами может оказаться неудобным. Поэтому большинство реализаций SQL позво- ляют создавать для таблиц синонимы. *Синоним*— это альтернатив- ное имя таблицы. При создании синонима пользователь становится его собственником, поэтому необходимость использования префикса к имени таблицы для него отпадает. Пользователь имеет право создавать синоним для таблицы, если он имеет по крайней мере одну привилегию в одном или более столбцах этой таблицы.

С помощью команды **CREATE SYNONYM** пользователь IVANOV мо- жет для таблицы с именем PETROV.STUDENT создать синоним с име- нем CLIENTS следующим образом:

**CREATE SYNONYM** CLIENTS **FOR** PETROV.STUDENT;

Теперь пользователь IVANOV может использовать таблицу с именем

CLIENTS в команде точно так же, как имя PETROV.STUDENT.

Как уже говорилось, префикс пользователя — это фактически часть имени любой таблицы. Всякий раз, когда пользователь не указывает собственное имя вместе с именем своей таблицы, SQL по умолчанию подставляет идентификатор пользователя в качестве префикса имени таблицы. Следовательно, два одинаковых имени таблицы, но связанные с различными владельцами, становятся неидентичными и, следователь- но, не приводят к какой-либо путанице в запросах. Таким образом, два пользователя могут создавать две полностью независимые таблицы с одинаковыми именами, но это также означает, что один пользователь может создать представление, основанное на имени, стоящем после имени таблицы и используемом другим пользователем. Это иногда де- лается в случаях, когда представление используется как замена самой исходной таблицы, например, если представление просто использует **CHECK OPTION** как заменитель ограничения **CHECK** в базовой таблице. Можно также создавать собственные синонимы пользователя, имена которых будут такими же, как и первоначальные имена таблиц. Напри- мер, пользователь PETROV может определить имя STUDENT как свой синоним для таблицы IVANOV.STUDENT с помощью запроса:

*6.12. Удаление синонимов* 117



**CREATE SYNONYM** STUDENT **FOR** IVANOV.STUDENT;

С точки зрения SQL, теперь имеются два разных имени одной таблицы: IVANOV.STUDENT и PETROV.STUDENT. Однако каждый из соответствующих пользователей может обращаться к данной таблице, используя имя STUDENT. SQL, как говорилось выше, сам добавит к этому имени недостающие имена пользователей в качестве префиксов.

* 1. Синонимы общего пользования (PUBLIC)

Если планируется использовать таблицу STUDENT большим числом пользователей, то удобнее, чтобы все пользователи ссылались к ней с помощью одного и того же имени. Это даст возможность, например, ис- пользовать указанное имя без ограничений в прикладных программах. Чтобы создать единое имя для всех пользователей, создается общий синоним.

Например, если все пользователи будут вызывать таблицу STUDENT с данными о студентах, можно присвоить ей *общий* синоним STUDENT следующим образом:

**CREATE PUBLIC SYNONYM** STUDENT **FOR** STUDENT;

Общие синонимы в основном создаются владельцами объектов или пользователями с привилегиями администратора базы данных (поль- зователь **DBA**). Другим пользователям при этом должны быть предо- ставлены соответствующие привилегии в таблице STUDENT, чтобы она была им доступна. Даже если имя является общим, сама таблица общей не является.

* 1. Удаление синонимов

Общие и другие синонимы могут удаляться командой **DROP SYNONYM**. Синонимы могут удаляться только их владельцами, кроме общих синонимов, которые могут удаляться соответствующими привилегированными пользователями (обычно это пользователи **DBA**). Чтобы удалить, например, синоним CLIENTS, когда вместо него уже появился общий синоним STUDENT, пользователь может ввести команду

**DROP SYNONYM** CLIENTS;

УПРАЖНЕНИЯ

1. Пользователь IVANOV передал Вам право **SELECT** в таблице EXAM\_MARKS. Запишите команду, позволяющую Вам обращаться к этой таблице, используя имя EXAM\_MARKS без префикса.
2. Вы передали право **SELECT** в таблице EXAM\_MARKS пользовате- лю IVANOV. Запишите команду, позволяющую ему обращаться к этой таблице, используя имя EXAM\_MARKS без префикса.

Г л а в а 7

**УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ**

В процессе выполнения последовательности команд SQL таблицы базы данных не всегда могут находиться в согласованном состоянии. В случае возникновения каких-либо сбоев, когда логически связан- ная последовательность запросов не доведена до конца, возможно нарушение целостности данных в базе. Для обеспечения целостности данных логически связанные последовательности запросов, неделимые с точки зрения воздействия на базу данных, объединяют в так называ- емые *транзакции*. При этом либо запросы, составляющие транзакцию, должны выполняться все полностью — с первого до последнего, и тогда транзакция завершается командой **COMMIT**, либо, если в силу каких-либо внешних причин это оказывается невозможным, внесенные запросами транзакции изменения в базе данных должны аннулировать- ся командой **ROLLBACK**. Во втором случае база данных возвращается в целостное состояние на момент, предшествующий началу транзакции. Это называют *откатом* транзакции.

Новая транзакция начинается после каждой команды **COMMIT** или

ROLLBACK.

В большинстве реализаций можно установить параметр, называе- мый **AUTOCOMMIT**. Он будет автоматически запоминать все выполняе- мые действия над данными. Действия, которые приведут к ошибке при незавершенной транзакции, всегда будут автоматически “откатаны” обратно.

Имеется возможность установки режима **AUTOCOMMIT** автоматиче- ски при регистрации. Если сеанс пользователя завершается аварий- но, например, произошел сбой системы или выполнена перезагрузка пользователя, то текущая транзакция выполнит автоматический откат изменений. Это — одна из возможностей управления выполнением диалоговой обработки запросов путем разделения команд на большое количество различных транзакций. Одиночная транзакция не должна содержать слишком много несвязанных команд, на практике она часто состоит из единственной команды. Хорошее правило, которому можно

следовать — это создавать транзакции из одной команды или несколь- ких близко связанных команд.

Например, требуется удалить сведения о студенте по фамилии Ива- нов из базы данных. Прежде, чем сведения будут удалены из таблицы STUDENT, требуется осуществить определенные действия с данными об этом студенте в других таблицах, в частности, с данными о его оценках. Необходимо установить в **NULL** соответствующее этому сту- денту поле STUDENT\_ID в таблице EXAM\_MARKS. После этого можно удалить запись об этом студенте из таблицы STUDENT. Эти действия выполняются с помощью двух запросов:

**UPDATE** EXAM\_MARKS

**SET** STUDENT\_ID = **NULL WHERE** STUDENT\_ID = 1004;

**DELETE FROM** STUDENT

**WHERE** STUDENT\_ID = 1004;

Если возникает проблема с удалением записи о студенте с фамили- ей Иванов (возможно, имеется другой внешний ключ, ссылающийся на него, о котором не было известно и который, соответственно, не учтен при удалении), то можно было бы отменить все сделанные изменения, по крайней мере до тех пор, пока проблема не будет решена. Для этого приведенную группу команд следует обрабатывать как одиноч- ную транзакцию, предусматривая ее завершение с помощью команды **COMMIT** или **ROLLBACK** — в зависимости от результата.

П р и л о ж е н и е

**ЗАДАЧИ ПО ПРОЕКТИРОВАНИЮ БД**

В приложении приводятся тексты задач по проектированию баз данных, относящихся к различным предметным областям. Требуется в соответствии с условиями задач:

сформировать структуру таблиц баз данных; подобрать подходящие имена таблицам и их полям;

*•*

*•*

обеспечить требования нормализации таблиц баз данных (т. е. приведение к пятой нормальной форме);

*•*

сформировать SQL-запросы для создания таблиц баз данных с указанием первичных и внешних ключей и необходимых огра- ничений, SQL-запросы для добавления, изменения и выборки необходимых данных.

*•*

При решении задач предполагается использование средств, позво- ляющих разрабатывать схемы баз данных, и приложений, работающих с базами данных (Power Designer, Oracle Developer, ERWin, Power Builder, Borland Delphi, C++ Builder и др.)

**Задача 1.** *Летопись острова Санта Белинда*.

Где-то в великом океане лежит воображаемый остров Санта Белин- да. Вот уже триста лет ведется подробная летопись острова. В летопись заносятся и данные обо всех людях, хоть какое-то время проживавших на острове. Для каждого из островитян записываются его имя, пол, даты рождения и смерти. Хранятся там и имена их родителей, если известно, кто они. У некоторых отсутствуют сведения об отце, у неко- торых — о матери, а часть людей, судя по записям, — круглые сироты. Из летописи можно узнать, когда был построен каждый дом, стоящий на острове (а если сейчас его уже нет, то когда он был снесен), точный адрес и подробный план этого дома, кто и когда в нем жил.

Точно так же, как и столетия назад, на острове действуют пред- приниматели, занимающиеся, в частности, ловлей рыбы, заготовкой сахарного тростника и табака. Большинство из них все делают сами, а некоторые нанимают работников, заключая с ними контракты разной

*Приложение. Задачи по проектированию БД* 151



продолжительности. Имеются записи и о том, кто кого нанимал, на какую работу, когда начался и закончился контракт. Собственно, круг занятий жителей острова крайне узок и не меняется веками. Неуди- вительно поэтому, что в летописи подробно описывается каждое дело, будь то рыбная ловля или выпечка хлеба. Все предприниматели — уро- женцы острова. Некоторые объединяются в кооперативы, и по записям можно установить, кто участвовал в деле, когда вступил и когда вышел из него, каким паем владел. Имеются краткие описания деятельности каждого предпринимателя или кооператива, сообщающие, в частности, когда было начато дело, когда и почему прекращено.

Предлагается сформировать систему нормализованных таблиц, в которых можно было бы хранить всю эту многообразную информацию. Подыщите выразительные имена для таблиц и полей, снабдив их при необходимости соответствующими пояснениями.

**Задача 2.** *База данных “Скачки”.*

В информационной системе клуба любителей скачек должна быть представлена информация об участвующих в скачках лошадях (кличка, пол, дата рождения), их владельцах (имя, адрес, телефон) и жокеях (имя, адрес, дата рождения, рейтинг). Необходимо сформировать таб- лицы для хранения информации по каждому состязанию: дата, время и место проведения скачек (ипподром), название состязаний (если тако- вое имеется), номера заездов, клички участвующих в заездах лошадей и имена жокеев с указанием занятого места и показанного в заезде времени.

**Задача 3.** *База данных “Хроники восхождений” в альпинистском клубе.*

В базе данных должны записываться даты начала и завершения каждого восхождения, имена и адреса участвовавших в нем альпи- нистов, название и высота горы, страна и район, где эта гора рас- положена. Присвойте выразительные имена таблицам и полям для хранения указанной информации. Написать запросы, осуществляющие следующие операции.

1. Для введенного пользователем интервала дат показать список гор с указанием даты последнего восхождения. Для каждой горы сфор- мировать в хронологическом порядке список групп, осуществлявших восхождение.
2. Предоставить возможность добавления новой вершины с указа- нием ее названия, высоты и страны местоположения.
3. Предоставить возможность изменения данных о вершине, если на нее не было восхождения.
4. Показать список альпинистов, осуществлявших восхождение в указанный интервал дат. Для каждого альпиниста вывести список гор, на которые он осуществлял восхождения в этот период, с указанием названия группы и даты восхождения.
5. Предоставить возможность добавления нового альпиниста в со- став указанной группы.
6. Показать информацию о количестве восхождений каждого аль- пиниста на каждую гору. При выводе список отсортировать по количе- ству восхождений.
7. Показать список восхождений (групп), которые осуществлялись в указанный пользователем период времени. Для каждой группы по- казать ее состав.
8. Предоставить возможность добавления новой группы, указав ее название, вершину, время начала восхождения.
9. Предоставить информацию о том, сколько альпинистов побывало на каждой горе. Список отсортировать в алфавитном порядке по на- званию вершин.

**Задача 4.** *База данных медицинского кооператива.*

Базу данных использует для работы коллектив врачей. В таблицы должны быть занесены имя, пол, дата рождения и домашний адрес каждого их пациента. Всякий раз, когда врач осматривает больного (пришедшего на прием или на дому), фиксируется дата и место прове- дения осмотра, симптомы, диагноз и предписания больному, простав- ляется имя пациента и имя врача. Если врач прописывает больному какое-либо лекарство, в таблицу заносится название лекарства, способ его приема, словесное описание предполагаемого действия и возмож- ных побочных эффектов.

**Задача 5.** *База данных “Городская Дума”.*

В базе хранятся имена, адреса, домашние и служебные телефоны всех членов Думы. В Думе работает около сорока комиссий, все участ- ники которых являются членами Думы. Каждая комиссия имеет свой профиль, например, вопросы образования, проблемы, связанные с жи- льем, и т. п. Данные по каждой из комиссий включают: председатель и состав, прежние (за 10 предыдущих лет) председатели и члены этой комиссии, даты включения и выхода из состава комиссии, избрания ее председателей. Члены Думы могут заседать в нескольких комиссиях. В базу заносятся время и место проведения каждого заседания комиссии с указанием депутатов и служащих Думы, которые участвуют в его организации.

1. Показать список комиссий, для каждой — ее состав с указанием председателя.
2. Предоставить возможность добавления нового члена комиссии.
3. Для введенного пользователем интервала дат и названия комис- сии показать в хронологическом порядке всех ее председателей.
4. Показать список членов Думы, для каждого из них — список комиссий, в которых он участвовал и/или был председателем.
5. Предоставить возможность добавления новой комиссии, с указа- нием председателя.
6. Для указанного интервала дат и комиссии выдать список членов с указанием количества пропущенных заседаний.
7. Вывести список заседаний в указанный интервал дат в хроноло- гическом порядке, для каждого заседания — список присутствующих.
8. Предоставить возможность добавления нового заседания, с ука- занием присутствующих.
9. По каждой комиссии показать количество проведенных заседа- ний в указанный период времени.

**Задача 6.** *База данных рыболовной фирмы.*

Фирме принадлежит небольшая флотилия рыболовных катеров. Каждый катер имеет паспорт, куда занесены его название, тип, во- доизмещение и дата постройки. Фирма регистрирует каждый выход на лов, записывая название катера, имена и адреса членов команды с указанием их должностей (капитан, боцман и т. п.), даты выхода и возвращения, а также улов (массу пойманной рыбы) отдельно по сортам (например, трески). За время одного рейса катер может посе- тить несколько рыболовных мест (банок). Фиксируется дата прихода на каждую банку и дата отплытия, качество выловленной рыбы (отличное, хорошее, плохое). На борту улов не взвешивается. Написать запросы, осуществляющие следующие операции.

1. По указанному типу и интервалу дат вывести все катера, осу- ществлявшие выход в море, указав для каждого в хронологическом порядке записи о выходе в море и значениях улова.
2. Предоставить возможность добавления выхода катера в море с указанием команды.
3. Для указанного интервала дат вывести для каждого сорта рыбы список катеров с наибольшим уловом.
4. Для указанного интервала дат вывести список банок, с указани- ем среднего улова за этот период. Для каждой банки вывести список катеров, осуществлявших лов.
5. Предоставить возможность добавления новой банки с указанием данных о ней.
6. Для заданной банки вывести список катеров, которые получили улов выше среднего.
7. Вывести список сортов рыбы и для каждого сорта — список рейсов с указанием даты выхода и возвращения, величины улова. При этом список показанных рейсов должен быть ограничен интерва- лом дат.
8. Для выбранного пользователем рейса и банки добавить данные о сорте и количестве пойманной рыбы.
9. Предоставить возможность пользователю изменять характери- стики выбранного катера.
10. Для указанного интервала дат вывести в хронологическом по- рядке список рейсов за этот период времени, с указанием для каждого рейса пойманного количества каждого сорта рыбы.
11. Предоставить возможность добавления нового катера.
12. Для указанных сорта рыбы и банки вывести список рейсов с указанием количества пойманной рыбы. Список должен быть отсорти- рован в порядке уменьшения количества пойманной рыбы.

**Задача 7.** *База данных фирмы, проводящей аукционы.*

Фирма занимается продажей с аукциона антикварных изделий и произведений искусства. Владельцы вещей, выставляемых на прово- димых фирмой аукционах, юридически являются продавцами. Лица, приобретающие эти вещи, именуются покупателями. Получив от про- давцов партию предметов, фирма решает, на котором из аукционов выгоднее представить конкретный предмет. Перед проведением очеред- ного аукциона каждой из выставляемых на нем вещей присваивается отдельный номер лота. Две вещи, продаваемые на различных аукцио- нах, могут иметь одинаковые номера лотов.

В книгах фирмы делается запись о каждом аукционе. Там от- мечаются дата, место и время его проведения, а также специфика (например, выставляются картины, написанные маслом и не ранее 1900 г.). Заносятся также сведения о каждом продаваемом предмете: аукцион, на который он заявлен, номер лота, продавец, отправная цена и краткое словесное описание. Продавцу разрешается выставлять любое количество вещей, а покупатель имеет право приобретать любое количество вещей. Одно и то же лицо или фирма может выступать и как продавец, и как покупатель. После аукциона служащие фирмы, проводящей аукционы, записывают фактическую цену, уплаченную за проданный предмет, и фиксируют данные покупателя.

Написать запросы, осуществляющие следующие операции.

1. Для указанного интервала дат вывести список аукционов в хронологическом порядке с указанием наименования, даты и места проведения. Для каждого из них показать список выставленных вещей.
2. Добавить для продажи на указанный пользователем аукцион предмет искусства с указанием начальной цены.
3. Вывести список аукционов с указанием отсортированных по величине суммарных доходов от продажи.
4. Для указанного интервала дат вывести список проданных на аук- ционах предметов. Для каждого из предметов дать список аукционов, где он выставлялся.
5. Предоставить возможность добавления факта продажи заданного предмета на указанном аукционе.
6. Для указанного интервала дат вывести список продавцов в по- рядке убывания общей суммы, полученной ими от продажи предметов в этот промежуток времени.
7. Вывести список покупателей и для каждого из них — список аукционов, где были сделаны приобретения в указанный интервал дат.
8. Предоставить возможность добавления записи о проводимом аукционе (место, время).
9. Для указанного места вывести список аукционов, отсортирован- ных по количеству выставленных вещей.
10. Для указанного интервала дат вывести список продавцов, кото- рые принимали участие в аукционах, с указанием для каждого из них списка выставленных предметов.
11. Предоставить возможность добавления и изменения информа- ции о продавцах и покупателях.
12. Вывести список покупателей с указанием количества приобре- тенных предметов в указанный период времени.

**Задача 8.** *База данных музыкального магазина.*

Таблицы базы данных содержат информацию о музыкантах, музы- кальных произведениях и обстоятельствах их исполнения. Несколько музыкантов, образующих единый коллектив, называются ансамблем. Это может быть классический оркестр, джазовая группа, квартет, квинтет и т. п. К музыкантам причисляют исполнителей (играющих на одном или нескольких инструментах), композиторов, дирижеров и руководителей ансамблей.

Кроме того, в базе данных хранится информация о компакт-дис- ках, которыми торгует магазин. Каждый компакт-диск, а точнее, его наклейка, идентифицируется отдельным номером, так что всем его копиям, созданным в разное время, присвоены одинаковые номера. На компакт-диске может быть записано несколько вариантов исполнения одного и того же произведения — для каждого из них в базе заведена отдельная запись. Когда выходит новый компакт-диск, регистрируется название выпустившей его компании (например, EMI), а также адрес оптовой фирмы, у которой магазин может приобрести этот компакт- диск. Не исключено, что компания-производитель занимается и опто- вой продажей компакт-дисков. Магазин фиксирует текущие оптовые и розничные цены на каждый компакт-диск, дату его выпуска, количе- ство экземпляров, проданных за прошлый год и в нынешнем году, а также число еще не проданных компакт-дисков.

**Задача 9.** *База данных кегельной лиги.*

Ставится задача спроектировать базу данных для секретаря ке- гельной лиги небольшого городка, расположенного на Среднем Западе США. В ней секретарь будет хранить всю информацию, относящуюся к кегельной лиге, а средствами СУБД — формировать еженедельные отчеты о состоянии лиги. Специальный отчет предполагается форми- ровать в конце сезона.

Секретарю понадобятся фамилии и имена членов лиги, их телефон- ные номера и адреса. Так как в лигу могут входить только жители городка, нет необходимости хранения для каждого игрока названия города и почтового индекса. Интерес представляют число очков, на- бранных каждым игроком в еженедельной серии из трех встреч, в которых он принял участие, и его текущая результативность (среднее

число набираемых очков в одной встрече). Секретарю необходимо знать для каждого игрока название команды, за которую он выступает, и фамилию (и имя) капитана каждой команды. Помимо названия, секре- тарь планирует назначить каждой команде уникальный номер.

Исходные значения результативности каждого игрока необходимы как в конце сезона, при определении игрока, достигшего наибольшего прогресса в лиге, так и при вычислении гандикапа для каждого игрока на первую неделю нового сезона. Лучшая игра каждого игрока и лучшие серии потребуются при распределении призов в конце сезона.

На каждую неделю каждой команде требуется назначать площадку, на которой она будет выступать. Эту информацию хранить в БД не нужно (соперники выступают на смежных площадках).

Наконец, в БД должна содержаться вся информация, необходимая для расчета положения команд. Команде засчитывается одна победа за каждую игру, в которой ей удалось набрать больше очков (выбить больше кеглей), чем команде соперников. Точно так же команде засчи- тывается одно поражение за каждую встречу, в которой эта команда выбила меньшее количество кеглей, чем команда соперников. Коман- де также засчитывается одна победа (поражение) в случае, если по сравнению с командой соперников ею набрано больше (меньше) очков за три встречи, состоявшиеся на неделе. Таким образом, на каждой неделе разыгрывается 4 командных очка (побед или поражений). В случае ничейного результата каждая команда получает 1/2 победы и 1/2 поражения. В случае неявки более чем двух членов команде автоматически засчитывается 4 поражения, а команде соперников — 4 победы. В общий результат команде, которой засчитана неявка, очки не прибавляются, даже если явившиеся игроки в этой встрече высту- пили, однако в индивидуальные показатели — число набранных очков и проведенных встреч — будут внесены соответствующие изменения. Написать запросы, осуществляющие следующие операции.

1. Для указанного интервала дат показать список выступающих команд. Для каждой из них вывести состав и капитана команды.
2. Предоставить возможность добавления новой команды.
3. Вывести список игровых площадок с указанием количества про- веденных игр на каждой из них.
4. Для указанного интервала дат вывести список игровых площа- док, с указанием списка игравших на них команд.
5. Предоставить возможность заполнения результатов игры двух команд на указанной площадке.
6. Вывести список площадок с указанием суммарной результатив- ности игроков на каждой из них.

**Задача 10.** *База данных библиотеки.*

Разработать информационную систему обслуживания библиотеки, которая содержит следующую информацию: название книги, ФИО авторов, наименование издательства, год издания, количество страниц,

количество иллюстраций, цена, название филиала библиотеки или кни- гохранилища, в которых находится книга, количество имеющихся в библиотеке экземпляров конкретной книги, количество студентов, ко- торым выдавалась конкретная книга, названия факультетов, в учебном процессе которых используется указанная книга.

**Задача 11.** *База данных по учету успеваемости студентов.*

База данных должна содержать данные:

* контингенте студентов — фамилия, имя, отчество, год поступ- ления, форма обучения (дневная/вечерняя/заочная), номер или название группы;

*•*

об учебном плане — название специальности, дисциплина, се- местр, количество отводимых на дисциплину часов, форма отчет- ности (экзамен/зачет);

*•*

* журнале успеваемости студентов — год/семестр, студент, дис- циплина, оценка.

*•*

**Задача 12.** *База данных для учета аудиторного фонда универ- ситета.*

База данных должна содержать следующую информацию об ауди- торном фонде университета: наименование корпуса, в котором распо- ложено помещение, номер комнаты, расположение комнаты в корпусе, ширина и длина комнаты в метрах, назначение и вид помещения, под- разделение университета, за которым закреплено помещение. В базе данных также должна быть информация о высоте потолков в поме- щениях (в зависимости от места расположения помещений в корпусе). Следует также учитывать, что структура подразделений университета имеет иерархический вид, когда одни подразделения входят в состав других (факультет, кафедра, лаборатория, . . .).

Помимо SQL-запросов для создания таблиц базы данных, составьте запрос на создание представления (**VIEW**), в котором помимо приве- денной выше информации присутствовали бы данные о площадях и объемах каждого помещения.

**Задача 13.** *База данных регистрации происшествий.*

Необходимо создать базу данных регистрации происшествий. База должна содержать:

данные для регистрации сообщений о происшествиях (регистра- ционный номер сообщения, дата регистрации, краткая фабула (тип происшествия));

*•*

информацию о принятом по происшествию решении (отказано в возбуждении дела, удовлетворено ходатайство о возбуждении уголовного дела с указанием регистрационного номера заведен- ного дела, отправлено по территориальной принадлежности); информацию о лицах, причастных к происшествию (регистраци- онный номер лица, фамилия, имя, отчество, адрес, количество

*•*

*•*

судимостей, отношение конкретного лица к данному происше- ствию: виновник, потерпевший, подозреваемый, свидетель, . . .).

**Задача 14.** *База данных для подготовки работы конференции.*

База данных должна содержать справочник персоналий участников конференции (фамилия, имя, отчество, ученая степень, ученое зва- ние, научное направление, место работы, кафедра (отдел), должность, страна, город, почтовый индекс, адрес, рабочий телефон, домашний телефон, электронный адрес), и информацию, связанную с участием в конференции (докладчик или участник, дата рассылки первого пригла- шения, дата поступления заявки, тема доклада, отметка о поступлении тезисов, дата рассылки второго приглашения, дата поступления орг- взноса, размер поступившего оргвзноса, дата приезда, дата отъезда, потребность в гостинице).

**Задача 15.** *База данных для обслуживания склада.*

База данных должна обеспечить автоматизацию складского учета.

В ней должны содержаться следующие данные:

информация о “единицах хранения” — номер ордера, дата, код поставщика, балансный счет, код сопроводительного документа по справочнику документов, номер сопроводительного докумен- та, код материала по справочнику материалов, счет материала, код единицы измерения, количество пришедшего материала, цена единицы измерения;

*•*

информация о хранящихся на складе материалах (справочник материалов) — код класса материала, код группы материала, наименование материала;

*•*

информация о единицах измерения конкретных видов материа- лов — код материала, единица измерения (метры, килограммы, литры и т. п.).

*•*

информация о поставщиках материалов — код поставщика, его наименование, ИНН, юридический адрес (индекс, город, улица, дом), адрес банка (индекс, город, улица, дом), номер банковского счета.

*•*

**Задача 16.** *База данных фирмы.*

Фирма отказалась от приобретения некоторых товаров у своих поставщиков, решив самостоятельно наладить их производство. С этой целью она организовала сеть специализированных цехов, каждый из которых принимает определенное участие в технологическом процессе. Каждому виду выпускаемой продукции присваивается, как обычно, свой шифр товара, под которым он значится в файле товарных запасов. Этот же номер служит и шифром продукта. В записи с этим шифром указывается, когда была изготовлена последняя партия этого продукта,

какова ее стоимость, сколько операций потребовалось.

Операцией считается законченная часть процесса производства, которая целиком выполняется силами одного цеха в соответствии с техническими требованиями, перечисленными на отдельном чертеже. Для каждого продукта и для каждой операции в базе данных фирмы заведена запись, содержащая описание операции, ее среднюю продол- жительность и номер требуемого чертежа, по которому его можно отыскать. Кроме того, указывается номер цеха, обычно производящего данную операцию.

В запись, связанную с конкретной операцией, заносятся потребные количества расходных материалов, а также присвоенные им шифры товара. Расходными называют такие материалы, как, например, элек- трический кабель, который нельзя использовать повторно. При выдаче расходного материала со склада в процессе подготовки к выполнению операции, регистрируется фактически выданное количество, соответ- ствующий шифр товара, номер служащего, ответственного за выдачу, дата и время выдачи, номер операции и номер наряда на проведение работ (о котором несколько ниже). Реально затраченное количество материала может не совпадать с расчетным (например, из-за брака).

Каждый из цехов располагает требуемым инструментарием и обо- рудованием. При выполнении некоторых операций их иногда недоста- точно, и цех вынужден обращаться в центральную инструментальную за недостающими инструментами. Каждый тип инструмента снабжен отдельным номером, и на него заведена запись со словесным описани- ем. Кроме того, отмечается, какое количество инструментов этого типа выделено цехам и какое осталось в инструментальной. Экземпляры инструмента конкретного типа, например, гаечные ключи одного раз- мера, различаются по своим индивидуальным номерам. На фирме для каждого типа инструмента имеется запись, содержащая перечень всех индивидуальных номеров. Кроме того, указаны даты их поступления на склад.

По каждой операции в фирме отмечают типы и соответствую- щие количества инструментов, которые должны использоваться при ее выполнении. Когда инструменты действительно берутся со склада, фиксируется индивидуальный номер каждого экземпляра, указываются номер заказавшего их цеха и номер наряда на проведение работ. В этом случае реально полученное количество также не всегда совпадает с заказанным.

Наряд на проведение работ по форме напоминает заказ на приобре- тение товаров, но, в отличие от последнего, направляется не поставщи- ку, а в один из цехов. Оформляется наряд после того, как руководство фирмы сочтет необходимым выпустить партию некоторого продукта. В наряд заносятся шифр продукта, дата оформления наряда, срок, к которому должен быть выполнен заказ, а также требуемое количество продукта.

Разработайте структуру таблиц базы данных, подберите имена таб- лиц и полей, в которых могла бы разместиться вся эта информация.

160 *Приложение. Задачи по проектированию БД*



Напишите SQL-запросы, осуществляющие следующие операции.

1. Для выбранного цеха выдать список выполняемых им операций. Для каждой операции показать список расходных материалов с указа- нием количества.
2. Показать список инструментов и предоставить возможность до- бавления нового.
3. Выдать список используемых инструментов, отсортированных по количеству их использования в различных нарядах.
4. Для указанного интервала дат вывести в хронологическом по- рядке список нарядов, для каждого из которых показать список ис- пользуемых инструментов.
5. Показать список операций и предоставить возможность добавле- ния новой операции.
6. Выдать список расходуемых материалов, отсортированных по количеству их использования в различных нарядах.
7. Выдать список товаров, с указанием используемых при их изго- товлении инструментов.
8. Показать список нарядов в хронологическом порядке и предоста- вить возможность добавления нового.
9. Выдать отчет о производстве товаров различными цехами, указав

**DBA** 114

DDL 13, 78

DML 13

**e**scape-символ 27

**S**QL

* встроенный 13
* интерактивный 12

Предметный указатель

* + возможный 83
  + первичный (**PRIMARY KEY**) 11, 83–85, 88, 92, 93
  + родительский 89, 90, 93, 94
  + уникальный 83

ключевые слова 16

команды 16

— **ALTER TABLE** 80, 83, 90, 91

* + — **ADD** 80
  + — **MODIFY** 80
  + — добавление столбца 80

наименование цеха, название товара и его количество.

**а**дминистратор базы данных 114

**б**аза данных учебная 16

* таблица EXAM\_MARKS 19
* таблица LECTURER 17
* таблица STUDENT 17
* таблица SUBJ\_LECT 19
* таблица SUBJECT 18
* таблица UNIVERSITY 18

**в**ставка

* столбцов 80
* строк 71, 74

**д**екартово произведение 58

**и**зменение таблицы 80, 83

индексация 79

* создание индекса 79
* удаление индекса 80 использование символа \* 21

**к**люч

* внешний (**FOREIGN KEY**) 12, 59,

83, 89–91, 93, 95, 96

* — изменение описания столбцов 80
* — синтаксис 80, 84, 91
* **COMMIT** 118
* **CREATE INDEX** 79

— **CREATE TABLE** 71, 78, 82, 87, 90

* — синтаксис 84
* **CREATE USER** 108, 115
* **CREATE VIEW** 97

— **DELETE** 71, 72, 75, 76, 94, 99, 112

* — ограничение удаления роди- тельского ключа
* — — **CASCADE** 94
* — — **NO ACTION** 94
* — — **RESTRICT** 94
* — — **SET DEFAULT** 94
* — — **SET NULL** 94
* **DROP INDEX** 80
* **DROP TABLE** 80
* **DROP USER** 108
* **DROP VIEW** 104

— **GRANT** 108, 109, 111, 114

— **INSERT** 71, 72, 74, 78, 82, 93, 94,

100, 112

* — **VALUES** 71, 74
* — вставить **NULL**-значение 71

— **REVOKE** 108, 111

* **ROLLBACK** 118

6 И. Ф. Астахова, В. М. Мельников, А. П. Толстобров, В. В. Фертиков

* **SELECT** 20, 79
* — аргументы

— — — **ALL** 38

— — — **DISTINCT** 12, 22, 38

* — использование символа \* 21
* — оператор **JOIN** 58, 59
* — оператор объединения таблиц

**UNION** 20

* — предложения
* — — **FROM** 20
* — — **GROUP BY** 20, 39

— — — **HAVING** 20, 39, 46, 49

— — — **ORDER BY** 20, 43, 44, 68

— — — — **ASC** 43

— — — — **DESC** 43

— — — **WHERE** 20, 23, 25, 73, 74

* — синтаксис 20

— **UPDATE** 71–73, 76, 93, 94, 99

* — ограничение модификации ро- дительского ключа
* — — **CASCADE** 94
* — — **NO ACTION** 94
* — — **RESTRICT** 94
* — — **SET NULL** 94
* — предложения **SET** 72, 73
* — синтаксис 72
* манипулирование данными 71
* оператор соединения таблиц

— — **JOIN** 58, 60, 62, 63

* — — **CROSS** 58

— — — **INNER** 58, 62

* — — **LEFT OUTER JOIN** 62
* — — **RIGHT OUTER JOIN** 63

**л**огика трехзначная 16, 42

**м**анипулирование данными 71

модель данных 10

**о**бновление 72

обозначения при описании синтак- сиса команд 16

ограничения 81

* **ALTER TABLE** 90

— **CHECK** 85, 86, 113, 116

* **CONSTRAINT** 81, 84
* **CREATE TABLE** 90

— **DEFAULT** 83, 86, 87

* **DELETE** 94
* **FOREIGN KEY** 90, 91, 93
* **INSERT** 93, 94
* **NOT NULL** 81, 82

— **PRIMARY KEY** 85, 88, 92, 93

— **UNIQUE** 83, 84, 88

* **UPDATE** 93, 94
* **WITH CHECK OPTION** 113, 116
* альтернативы для **NULL** 87
* в командах
* — **ALTER TABLE** 83, 84
* — **CREATE TABLE** 78, 82, 84
* — **INSERT** 82
* ввод значений поля 90
* значения по умолчанию 86, 87
* ключ
* — внешний (**FOREIGN KEY**) 90, 91, 93
* — первичный (**PRIMARY KEY**) 81, 83–85, 88, 92, 93
* — родительский 89, 90, 93, 94
* — — модификация 93, 94
* — составной 85, 89
* модификация значений поля 90
* присвоение имен 84
* проверка значений полей 85, 86
* ссылочная целостность 88, 90, 93,

94

* столбца 81, 91

— таблицы 81–83, 85, 90, 91

* удаление значений поля 90
* удаления и модификации роди- тельского ключа **ON DELETE** и **ON UPDATE**
* — **CASCADE** 94
* — **NO ACTION** 94
* — **RESTRICT** 94
* — **SET DEFAULT** 94
* — **SET NULL** 94
* уникальность 83, 84, 88 операторы
* (вычитание) 29

*−*

* (конкатенация строк) 29

*"*

* \* (умножение) 29

— + (сложение) 29

* / (деление) 29
* **ALL** 55
* **ANY** 55

— **BETWEEN** 25–27, 107

* **COUNT** 57

— **EXISTS** 50, 55, 57

— **IN** 25, 45–47, 52, 53, 74–77, 106,

107, 113

* **IS NOT NULL** 16
* **IS NULL** 16

— **LIKE** 25–27

* **NOT IN** 25, 52
* **UNION** 66, 68
* сравнение 11, 23, 87

отмена привилегий 111

отношение 10

* атрибут 11
* домен 11
* заголовок 10
* кардинальное число 11
* ключ
* — внешний (**FOREIGN KEY**) 12
* — первичный (**PRIMARY KEY**) 11
* кортеж 11
* свойства 11
* степень 11

**п**ароль 108, 115

* **IDENTIFIED BY** 108, 115

подзапросы 74

* в командах
* — **DELETE** 75
* — **UPDATE** 76
* в предложениях
* — **FROM** 75
* — **HAVING** 46
* в представлениях 103
* вложенные 45
* связанные 46, 50
* — в предложении **HAVING** 49 пользователи 108
* создание 108, 115
* удаление 108

права доступа 108; см. привилегии представление (**VIEW**) 97, 98

* агрегированное 102
* вставка строки 102
* защита данных 105, 106
* использование команды
* — **DELETE** 99
* — **GROUP BY** 102

— — **INSERT** 72, 102, 112

* маскирующее 99
* — столбцы 99, 101
* — — модификация 99
* — строки 99
* — — модификация 100, 101
* многих таблиц 103
* модификация 73, 76, 98, 99
* — значений 105
* — использование
* — — **DISTINCT** 105
* — — **GROUP BY** 105
* — — **HAVING** 105
* — подзапросы 105
* не обновляемое 105, 106
* обновляемое 105, 112
* подзапросы 103
* создание 97
* удаление 104

префикс 116

привилегии 108, 110, 113

* аргументы

— — **ALL** 110

* — **ALL PRIVILEGES** 110

— — **PUBLIC** 110, 111

* базы данных 114
* в базовых таблицах 110
* в представлениях 110

— виды 108, 109

* — **ALTER** 109

— — **DELETE** 109, 112

* — **EXECUTE** 109
* — **INDEX** 109

— — **INSERT** 109, 112

* — **REFERENCES** 109, 112

— — **SELECT** 109, 112, 113

* — **SYNONYM** 109

— — **UPDATE** 109, 112

* использование представлений 111
* ограничение для строк 112
* отмена 108
* регистрации 114
* системы 114

— — **CONNECT** 114, 115

— — **DBA** 114, 115

* — **RESOURCE** 114, 115

6\*

* — Администратор Базы Данных 114
* — подключить 114
* — ресурс 114
* создавать
* — базовые таблицы 114
* — представления 114
* — синонимы 114

— установка 108, 109, 114

* фильтрация 111

псевдонимы 65

**р**еляционная модель данных 10

**с**бои 118

символьные константы 28

синонимы 116

* **CREATE SYNONYM** 116
* **DROP SYNONYM** 117
* общего пользования (**PUBLIC**) 117
* создание 116
* удаление 117

соединение 59

* внешнее 62

— — левое 62, 63

* — полное 63
* — правое 63
* — синтаксис ORACLE 63
* внутреннее (**INNER**) 58, 62
* использование псевдонимов 65
* полное (**CROSS**) 58
* эквисоединение 58 создание
* индексов 79
* объектов базы данных 78
* пользователей 108
* представлений 97
* синонимов 116, 117
* таблиц базы данных 78 сравнение 11, 23, 87

ссылочная целостность 12, 59,

88–91, 93–96

столбец

* добавление 80
* изменение описания 80 строка
* вставка 71
* идентификаторы строк **ROWID** 79

— удаление 72, 76, 112

суперпользователь 114

**т**аблица 10

* базовая 97
* виртуальная 98, 103
* изменение 80, 83
* именованная 97
* родительская 90
* удаление 80 типы данных
* пропущенные данные (**NULL**) 15, 16, 41, 55, 71, 81, 82, 84, 87
* строка символов
* — **CHAR** 13
* — **CHAR VARYING** 14
* — **CHARACTER** 13
* — **CHARACTER VARYING** 14
* — **VARCHAR** 14
* числовые типы 14
* — **DECIMAL** 14
* — **DOUBLE PRECISION** 15
* — **FLOAT** 14
* — **INTEGER** 14
* — **NUMBER** 15
* — **NUMERIC** 14
* — **REAL** 14
* — **SMALLINT** 14

типы данных SQL 13 транзакция 118

* **AUTOCOMMIT** 118
* завершение 118
* — **COMMIT** 118
* — **ROLLBACK** 118
* — нормальное 118
* — откат 118

**у**даление

* индексов 80
* пользователей 108
* представлений 104
* синонимов 117

— строк 72, 76, 112

* таблиц базы данных 72, 80

**ф**ункции

* агрегирующие 38

— — **AVG** 38, 42

— — **COUNT** 38, 41

— — **COUNT**(\*) 38

* — **MAX** 38
* — **MIN** 38
* — **SUM** 38
* встроенные 28
* — **ABS** 33
* — **CAST** 36
* — **CEIL** 33
* — **COS** 33
* — **COSH** 33
* — **EXP** 34
* — **FLOOR** 33
* — **INITCAP** 30
* — **INSTR** 32
* — **LENGTH** 32
* — **LOWER** 30
* — **LPAD** 30
* — **LTRIM** 31
* — **POWER** 34
* — **ROUND** 33
* — **RPAD** 31
* — **RTRIM** 31
* — **SIGN** 34
* — **SIN** 33
* — **SINH** 33
* — **SQRT** 34
* — **SUBSTR** 31
* — **TAN** 33
* — **TANH** 33
* — **TO\_CHAR** 34
* — **TO\_DATE** 35
* — **TO\_NUMBER** 35
* — **TRUNC** 33
* — **UPPER** 30
* — преобразование букв 30
* — работы с числами 33
* — символьные строковые 30

**ц**елостность данных 118

**э**квисоединение 58

**я**зык

* обработки данных (DML) 13
* определения данных (DDL) 13, 78