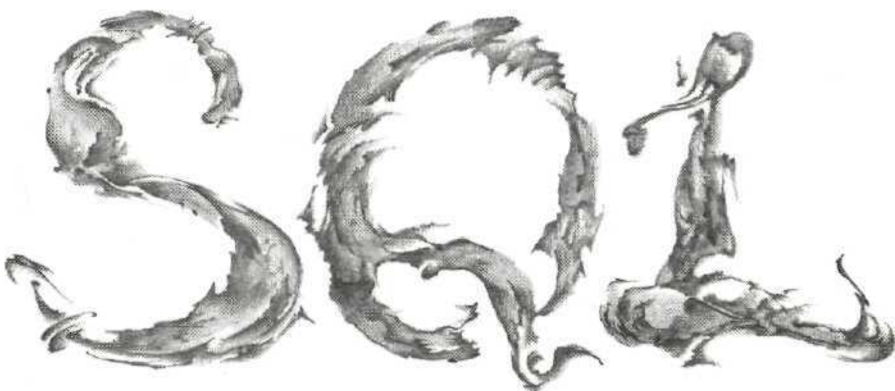
И.Ф. Астахова
А.П. Толстобров
В.М. Мельников



SQL в примерах и задачах

УДК 004.655.3(075.8)
ББК 32.973.26-018.1я73
А91

Рецензенты:

доцент кафедры АСИТ Московского государственного университета *Н.Д. Васюкова;*

Воронежское научно-производственное предприятие «РЕЛЭКС»;

кафедра информатики и МПМ Воронежского государственного педагогического университета;

доктор технических наук, профессор, зав. кафедрой математического моделирования Воронежской государственной технологической академии *В.В. Сысоев;*

доктор физико-математических наук, профессор, декан факультета компьютерных наук Воронежского государственного университета *Э.К. Алгазинов*

Астахова И.Ф.

А91 SQL в примерах и задачах; Учеб. пособие / И.Ф. Астахова, А.П. Толстобров, В.М. Мельников.— Мн.: Новое знание, 2002. — 176 с.

ISBN 985-475-004-3.

Изложены основные понятия и способы применения SQL — популярного языка запросов к реляционным базам данных. Описаны приемы манипулирования данными и формирования запросов различной степени сложности. Каждая глава пособия сопровождается упражнениями, которые позволяют закрепить на практике теоретические знания.

Книга является учебным пособием для студентов, обучающихся по направлению «Прикладная математика и информатика», а также может быть использована для самостоятельного изучения языка SQL.

УДК 004.655.3(075.8)
ББК 32.973.26-018.1я73

ISBN 985-475-004-3

Астахова И.Ф., Толстобров А.П., Мельников В.М., 2001
Оформление. ООО «Новое знание», 2002

Оглавление

Введение . . . **8**

1. Основные понятия и определения . . . **10**
- 1.1. Основные понятия реляционных баз данных . . . 10
- 1.2. Отличие SQL от процедурных языков программирования . . . 12
- 1.3. Интерактивный и встроенный SQL . . . 13
- 1.4. Составные части SQL . . . 13
- 1.5. Типы данных SQL . . . 14
  - 1.5.1. Тип данных «строка символов» . . . 14
  - 1.5.2. Числовые типы данных . . . 15
  - 1.5.3. Дата и время . . . 16
  - 1.5.4. Неопределенные или пропущенные данные (NULL) . . . 17
- 1.6. Используемые термины и обозначения . . . 18
- 1.7. Учебная база данных . . . 18

2. Выборка данных (оператор SELECT) . . . **23**
- 2.1. Простейшие SELECT-запросы . . . 23
- 2.2. Операторы IN, BETWEEN, LIKE, is NULL . . . 28
- 2.3. Преобразование вывода и встроенные функции . . . 31
  - 2.3.1. Числовые, символьные и строковые константы . . . 31
  - 2.3.2. Арифметические операции для преобразования числовых данных . . . 32
  - 2.3.3. Операция конкатенации строк . . . 33
  - 2.3.4. Функции преобразования символов в строке . . . 34
  - 2.3.5. Строковые функции . . . 34
  - 2.3.6. Функции работы с числами . . . 37
  - 2.3.7. Функции преобразования значений . . . 39
- 2.4. Агрегирование и групповые функции . . . 43
- 2.5. Пустые значения (NULL) в агрегирующих функциях . . . 46
  - 2.5.1. Влияние NULL-значений в функции COUNT . . . 46
  - 2.5.2. Влияние NULL-значений в функции AVG . . . 47

2.6. Результат действия трехзначных условных операторов ... 47
2.7. Упорядочение выходных полей (ORDER BY) ... 49
2.8. Вложенные подзапросы ... 51
2.9. Формирование связанных подзапросов ... 53
2.10. Связанные подзапросы в HAVING ... 54
2.11. Использование оператора EXISTS ... 56
2.12. Операторы сравнения с множеством значений IN, ANY, All ... 58
2.13. Особенности применения операторов ANY, ALL, EXISTS при обработке пустых значений (NULL) ... 60
2.14. Использование COUNT вместо EXISTS ... 62
2.15. Оператор объединения UNION ... 63
2.16. Устранение дублирования в UNION ... 64
2.17. Использование UNION с ORDER BY ... 66
2.18. Внешнее объединение ... 67
2.19. Соединение таблиц с использованием оператора JOIN ... 69
2.19.1. Операции соединения таблиц посредством ссылочной целостности ... 70
2.19.2. Внешнее соединение таблиц ... 72
2.19.3. Использование псевдонимов при соединении таблиц ... 74

3. Манипулирование данными ... 78
3.1. Команды манипулирования данными ... 78
3.2. Использование подзапросов в INSERT ... 81
3.2.1. Использование подзапросов, основанных на таблицах внешних запросов ... 82
3.2.2. Использование подзапросов с DELETE ... 83
3.2.3. Использование подзапросов с UPDATE ... 85

4. Создание объектов базы данных ... 87
4.1. Создание таблиц базы данных ... 87
4.2. Использование индексации для быстрого доступа к данным ... 88
4.3. Изменение существующей таблицы ...

4.4. Удаление таблицы ... 90
4.5. Офаничения на множество допустимых значений данных ... 91
4.5.1. Ограничение NOT NULL ... 92
4.5.2. Уникальность как ограничение на столбец ... 93
4.5.3. Уникальность как ограничение таблицы ... 94
4.5.4. Присвоение имен ограничениям ... 95
4.5.5. Ограничение первичных ключей ... 96
4.5.6. Составные первичные ключи ... 96
4.5.7. Проверка значений полей ... 97
4.5.8. Проверка ограничивающих условий с использованием составных полей ... 98
4.5.9. Установка значений по умолчанию ... 99
4.6. Поддержка целостности данных ... 101
4.6.1. Внешние и родительские ключи ... 101
4.6.2. Составные внешние ключи ... 102
4.6.3. Смысл внешнего и родительского ключей ... 102
4.6.4. Ограничение FOREIGN KEY (внешнего ключа) ... 103
4.6.5. Внешний ключ как ограничение таблицы ... 103
4.6.6. Внешний ключ как ограничение столбца ... 105
4.6.7. Поддержание ссылочной целостности и ограничения значений родительского ключа ... 107
4.6.8. Использование первичного ключа в качестве уникального внешнего ключа ... 107
4.6.9. Ограничения значений внешнего ключа ... 108
4.6.10. Действие ограничений внешнего и родительского ключей при использовании команд модификации ... 108

5. Представления (VIEW) ... 112
5.1. Представления — именованные запросы ... 112
5.2. Представления таблиц ... 113
5.3. Представления столбцов ... 113
5.4. Модифицирование представлений ... 114
5.5. Маскирующие представления ... 114
5.5.1. Представления, маскирующие столбцы ... 114
5.5.2. Операции модификации в представлениях, маскирующих столбцы ... 115

5.5.3. Представления, маскирующие строки . . . 115
5.5.4. Операции модификации в представлениях, маскирующих строки . . . 116
5.5.5. Операции модификации в представлениях, маскирующих строки и столбцы . . . 117
5.6. Агрегированные представления . . . 119
5.7. Представления, основанные на нескольких таблицах ... 119
5.8. Представления и подзапросы . . . 120
5.9. Ограничения применения оператора SELECT для создания представлений . . . 121
5.10. Удаление представлений . . . 121
5.11. Изменение значений в представлениях . . . 122
5.12. Примеры обновляемых и необновляемых представлений . . . 123
5.13. Представления, базирующиеся на других представлениях . . . 125

6. Определение прав доступа пользователей к данным . . . 128

6.1. Пользователи и привилегии . . . 128
6.2. Стандартные привилегии . . . 129
6.3. Команда GRANT . . . 130
6.4. Использование аргументов ALL и PUBLIC . . . 131
6.5. Отмена привилегий . . . 131
6.6. Использование представлений для фильтрации привилегий . . . 132
6.6.1. Ограничение привилегии SELECT для определенных столбцов . . . 133
6.6.2. Ограничение привилегий для определенных строк . . . 133
6.6.3. Предоставление доступа только к извлеченным данным . . . 134
6.6.4. Использование представлений в качестве альтернативы ограничениям . . . 134
6.7. Другие типы привилегий . . . 135
6.8. Типичные привилегии системы . . . 136

6.9. Создание и удаление пользователей.... 136
6.10. Создание синонимов (SYNONYM) . . . 138
6.11. Синонимы общего пользования (PUBLIC) 139
6.12. Удаление синонимов . . . 139

7. Управление транзакциями . . . 141

Приложение 1. Ответы к упражнениям . . . 143
Приложение 2. Задачи по проектированию БД 159
Предметный указатель . . . 171

Введение

В настоящее время информационные системы, применяющие базы данных, представляют собой одну из важнейших областей современных компьютерных технологий. С этой сферой связана большая часть современного рынка программных продуктов. Среди общих тенденций в развитии таких систем выделяются процессы интеграции и стандартизации, затрагивающие структуры данных и способы их обработки и интерпретации, системное и прикладное программное обеспечение, средства взаимодействия компонентов баз данных и многое другое. Современные системы управления базами данных (СУБД) основаны на реляционной модели представления данных — в большой степени благодаря простоте и четкости ее концептуальных понятий и строгого математического обоснования.

Неотъемлемая и важнейшая часть любой системы, применяющей базы данных, — языковые средства, обеспечивающие возможность доступа и действий над данными, определения их структур, способов использования и интерпретации. Язык SQL появился в 1970-е годы как одно из таких средств. Его прототип 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 исчисления, 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. Этот стандарт поддерживается всеми ведущими мировыми фирмами, действующими в сфере технологий баз данных. Использование выразительного и эффективного стандартного языка позволило обеспечить высокую степень независимости разрабатываемых прикладных программных систем от конкретного типа используемой СУБД, существенно поднять уровень и унификацию инструментальных средств разработки приложений, работающих с реляционными базами данных.

Говоря о стандарте языка SQL, следует заметить, что большинство его коммерческих реализаций имеют некоторые, большие или меньшие, отличия от стандарта. Это, конечно, ухудшает совместимость систем, использующих различные «диалекты» SQL. Но, с другой стороны, полезные расширения реализаций языка обеспечивают его развитие и со временем включаются в новые редакции стандарта. Учитывая место, занимаемое SQL в современных информационных технологиях, его знание необходимо любому специалисту, работающему в этой области.

Данное пособие в первую очередь предназначено преподавателям и студентам и ориентировано на обучение основам применения языка SQL по учебным курсам, связанным с изучением информационных систем, базирующихся на базах данных. В настоящее время такие курсы входят в учебные планы ряда университетских специальностей. С этой целью в пособии большое внимание уделялось подбору материала для примеров, а также задач и упражнений, необходимых для получения практических навыков составления SQL-запросов к базе данных. При этом в определениях и примерах приоритет отдавался простоте и доходчивости материала, возможно, с некоторым ущербом строгости его изложения. По этой же причине в пособие не вошли особенности языка, требующие более глубоких знаний о функционировании современных СУБД и информационных систем, изучение и использование которых имеет смысл только при условии получения навыков практического использования базовых конструкций языка. При изложении материала авторы по возможности старались, кроме специально оговоренных случаев, не отступать от стандарта языка SQL.

В приложении 1 пособия содержатся ответы на большинство приведенных в нем упражнений. Примеры и задачи упражнений протестированы с использованием СУБД Oracle и отечественной СУБД ЛИНТЕР. ЛИНТЕР представляет собой полномасштабный кросс-платформенный SQL-сервер, соответствующий основным мировым стандартам, предъявляемым к системам такого класса. Для некоммерческого использования учебным заведениям он предоставляется бесплатно. Более подробную информацию о системе можно получить на сайте компании РЕЛЭКС по адресу www.relex.ru.

В приложении 2 приведены тексты дополнительных задач по проектированию баз данных. Эти задачи могут использоваться в качестве тем курсовых работ и для самостоятельной работы студентов.

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

Основные понятия и определения

1.1. Основные понятия реляционных баз данных

Основой современных систем, применяющих базы данных, является *реляционная модель данных.* В этой модели данные, представляющие информацию о предметной области, организованы в виде двухмерных таблиц, называемых *отношениями.* На рисунке 1 приведен пример такой таблицы-отношения и поясняются основные термины реляционной модели.

Рис. 1. Пример таблицы-отношения реляционной базы данных

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

- Строки таблицы-отношения называются *кортежами,* или *записями.* Столбцы называются *атрибутами.* Термины — атрибут, столбец, колонка, поле — обычно используются как синонимы. Каждый атрибут имеет имя, которое должно быть уникальным в конкретной таблице-отношении, однако в разных таблицах имена атрибутов могут совпадать.
- Количество кортежей в таблице-отношении называется *кардинальным числом* отношения, а количество атрибутов — *степенью* отношения.
- *Ключ,* или *первичный ключ* отношения — это уникальный идентификатор строк (кортежей), то есть такой атрибут (набор атрибутов), для которого в любой момент времени в отношении не существует строк с одинаковыми значениями этого атрибута (набора атрибутов). На приведенном рисунке таблицы ячейка с именем ключевого атрибута имеет нижнюю границу в виде двойной черты.
- *Домен* отношения — это совокупность значений, из которых могут выбираться значения конкретного атрибута. То есть конкретный набор имеющихся в таблице значений атрибута в любой момент времени должен быть подмножеством множества значений домена, на котором определен этот атрибут. В общем случае на одном и том же домене могут быть определены значения разных атрибутов. Важным является то, что домены вводят ограничения на операции сравнения значений различных атрибутов. Эти ограничения состоят в том, что корректным образом можно сравнивать между собой только значения атрибутов, определенных на одном и том же домене.

Отношения реляционной базы данных обладают следующими свойствами:

- в отношениях не должно быть кортежей-дубликатов,
- кортежи отношений не упорядочены,
- атрибуты отношений также не упорядочены.

Из этих свойств отношения вытекают важные следствия.

- Уникальность кортежей определяет, что в отношении *всегда* имеется атрибут или набор атрибутов, позволяющих *идентифицировать* кортеж, другими словами, в отношении ***всегда*** есть первичный ключ.

- Неупорядоченность кортежей приводит к тому, что, во-первых, в отношении не существует другого способа адресации кортежей, кроме адресации *по ключу,* а во-вторых — в отношении не существует таких понятий, как первый кортеж, последний, предыдущий, следующий и т.д.
- Неупорядоченность атрибутов определяет, что единственным способом их адресации в запросах является использование наименования атрибута.

Относительно свойства реляционного отношения, касающегося отсутствия кортежей-дубликатов, следует сделать важное замечание. В этом пункте SQL не полностью соответствует реляционной модели. А именно: 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 нескольких таблиц взаимосвязанной информации появляется возможность выполнения более сложных операций с данными, то есть более сложной их обработки. Для работы со связанными данными из нескольких таблиц важным является понятие так называемых *внешних ключей.*

Внешним ключом таблицы называется атрибут (набор атрибутов) этой таблицы, каждое значение которого в текущем состоянии таблицы всегда совпадает со значением атрибутов, являющихся ключом, в другой таблице. Внешние ключи используются для связывания значений атрибутов из разных таблиц. С помощью внешних ключей обеспечивается так называемая ссылочная целостность базы данных, то есть согласованность данных, описывающих одни и те же объекты, но хранящихся в разных таблицах.

1.2. Отличие SQL от процедурных языков программирования

SQL относится к классу непроцедурных языков программирования. В отличие от универсальных процедурных языков, которые также могут быть использованы для работы с базами данных, 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-отношений. 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, то есть последовательность действий, необходимых для получения результата, а условия, которым должны удовлетворять кортежи результирующего отношения, сформулированные в терминах входного (или входных) отношения.

1.3. Интерактивный и встроенный SQL

Существуют и используются две формы языка SQL: *интерактивный SQL* и *встроенный SQL.*

Интерактивный SQL используется для задания SQL-запросов пользователем и получения результата в интерактивном режиме.

Встроенный SQL состоит из команд SQL, встроенных внутрь программ, обычно написанных на каком-то другом языке (Паскаль, С, С++ и др.). Это делает программы, использующие такие языки, более мощными, гибкими и эффективными, обеспечивая их применение для работы с данными, хранящимися в реляционных базах. При этом, однако, требуются дополнительные средства интерфейса SQL с языком, в который он встраивается.

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

1.4. Составные части SQL

И интерактивный, и встроенный SQL подразделяются на следующие 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.д.).

В число дополнительных функций DDL могут быть включены и средства ограничения целостности данных, определения порядка структур их хранения, описания элементов физического уровня хранения данных.

Язык обработки Даннь^^х — 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.

1.5. Типы данных SQL

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 согласуются с требованиями официального стандарта SQL. Это объясняется, в частности, желанием обеспечить совместимость SQL с другими языками программирования.

1.5.1. Тип данных «строка символов»

Стандарт поддерживает только один тип представления текста — CHARACTER (CHAR). Этот тип данных представляет собой символьные строки фиксированной длины. Его синтаксис имеет вид:

CHARACTER [(длина)] или

CHAR [(длына)].

Текстовые значения поля таблицы, определенного как тип CHAR, имеют фиксмрованную длину, которая определяется параметром длина. Этот параметр может принимать значения от 1 до 255, то есть строка может содержать до 255 символов. Если во вводимой в поле текстовой константе фактическое число символов меньше числа, определенного параметром длмна, то эта константа автоматически дополняется справа пробелами до заданного числа символов.

Некоторые реализации языка SQL поддерживают в качестве типа данных строки переменной длины. Этот тип может обозначаться ключевыми словами VARCHAR (j, CHARACTER VARYING или CHAR VARYING (j . Он описывает текстовую строку, которая может иметь произвольную длину до определенного конкретной реализацией SQL максимума (в Oracle — до 2000 символов). В отличие от типа CHAR в этом случае при вводе текстовой константы, фактическая длина которой мены^е заданной, не производится ее дополнение пробелами до заданного максимального значения.

Константы, имеющие тип CHARACTER и VARCHAR, в выражениях SQL заключаются в одиночные кавычки, например, 'текст'.

Следующие предложения эквивалентны:

VARCHAR [(длмня)], CHAR VARYING [(длмнд)],

CHARACTER VARYING [(длмна)].

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 использоватъ память, выделяемую для хранения текстовых значений, и оказывается более удобным при выполнении операций, связанных со сравнением текстовых констант.

1.5.2. Числовые типы данных

Стандартными числовыми типами данных SQL являются:

- INTEGER -- используется для представления целых чисел в диапазоне от -2^{31} до $+2^{31}$.
- SMOLLINT -- используется для представления целых чисел в меньшем, чем для INTEGER, диапазоне, а именно — от -2^{15} до $+2^{15}$.
- DECIMAL (точность[,масштаб]) — десятичное число с фиксированной точкой, точность определяет количество значащих цифр в числе. Масштаб указывает максимальное число цифр справа от точкм.
- NUmeRIC (точность[,масштаб|) — десятичное число с фиксированной точкой, такое же, как и DECIMAL.

- FLOAT [(точность)] — число с плавающей точкой и указанной минимальной точностью.
- REAL — число такое же, как при типе FLOAT, за исключением определения точности по умолчанию (в зависимости от конкретной реализации SQL).
- DOUBLE PRECISION — число аналогично REAL, но точность в два раза выше точности 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 полагается равным 38. Значение параметра масштаб по умолчанию предполагается равным 0. Значение параметра *точность* может изменяться от 1 до 38; значение параметра *масштаб* может изменяться от —84 до 128. Использование отрицательных значений масштаба означает сдвиг десятичной точки в сторону старших разрядов. Например, определение NUMBER (7,—3) означает округление до тысяч.

Типы DECIMAL (иногда обозначаемый DEC) и NUMERIC полностью эквивалентны типу NUMBER.

Синтаксис: DECIMAL [(точнос/иь[,л«асштао])],

DEC *[(точностъ[,масштаб])],*

NUMERIC *[(точность[,масштаб\)].*

1.5.3. **Дата** и время

Тип данных, предназначенный для представления даты и *времени,* также является нестандартным, хотя и чрезвычайно полезным. Для точного определения типов данных, поддерживаемых конкретной СУБД, следует обращаться к ее документации.

В СУБД Oracle имеется тип DATE, используемый для хранения даты и времени. Поддерживаются даты, начиная от 1 января 4712 года до н.э. и до 31 декабря 4712 года. По умолчанию при

определении даты без уточнения времени принимается время полуночи.

Наличие типа данных для хранения даты и времени позволяет поддерживать специальную арифметику дат и времен. 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 формата, принятого в операционной системе. Например, '03.05.1999', или '12/06/1989', или '03-nov-1999', или 'ОЗ-арг-99'.

1.5.4. Неопределенные или пропущенные данные (NOLL)

Для обозначения отсутствующих, пропущенных или неизвестных значений атрибута в SQL используется ключевое слово NULL. Довольно часто можно встретить словосочетание *«атрибут имеет значение* NULL». Строго говоря, NULL не является значением в обычном понимании, а используется именно для обозначения того факта, что действительное значение атрибута на самом деле пропущено или неизвестно. Это приводит к ряду особенностей, что следует учитывать при использовании значений атрибутов, которые могут находиться в состоянии NULL.

- В агрегирующих функциях, позволяющих получать сводную информацию по множеству значений атрибута, например суммарное или среднее значение, для обеспечения точности и однозначности толкования результатов отсутствующие или NULL-значения атрибутов игнорируются.
- Условные операторы от булевой двузначной логики TRUE/FALSE расширяются до трехзначной логики TRUE/FALSE/UNKNOWN.
- Все операторы, за исключением оператора конкатенации строк «||», возвращают пустое значение (NULL), если значение любого из операндов отсутствует (имеет «значение NULL»).
- Для проверки на пустое значение следует использовать операторы is NULL и is NOT NULL (использование с этой целью оператора сравнения «=» является ошибкой).
- Функции преобразования типов, имеющие NULL в качестве аргумента, возвращают пустое значение (NULL).

1.6. **Используемые** термины и обозначения

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

Команды, или *предложения,* являются инструкциями, с помощью которых SQL обращается к базе данных. Команды состоят из 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: ТАБЛ^ ЦА1, АТРИБУТ_2.

В описании синтаксиса команд SQL:

- оператор определения «::=» разделяет определяемый элемент (слева от оператора) и собственно его определение (справа от оператора);
- квадратные скобки «[ ]» указывают *необязательный* элемент синтаксической конструкции;
- многоточие «… » определяет, что выражение, предшествующее ему, может повторяться любое число раз;
- фигурные скобки «{ }» объединяют последовательность элементов в *логическую группу,* один из элементов которой должен быть обязательно использован;
- вертикальная черта «|» указывает, что часть определения, следующая за этим символом, является одним из возможных вариантов;
- в угловые скобки «< >» заключаются элементы, объясняемые по мере того, как они вводятся.

1.7. Учебная **база** данных

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

STUDENT (Студент)

STUDENT ID	SURNAME	NAME	STIPEND	KURS	CITY	BIRTHDAY	UNIV ID
1	Иванов	Иван	150	1	Орел	3/12/1982	10
3	Петров	Петр	200	3	Курск	1/12/1980	10
6	Сидоров	Вадим	150	4	Москва	7/06/1979	22
10	Кузнецов	Борис	0	2	Брянск	8/12/1981	10
12	Зайцева	Ольга	250	2	Липецк	1/05/1981	10
265	Павлов	Андрей	0	3	Воронеж	5/11/1979	10
32	Котов	Павел	150	5	Белгород	NULL	14
654	Лукин	Артем	200	3	Воронеж	1/12/1981	10
276	Петров	Антон	200	4	NULL	5/08/1981	22
55	Белкин	Вадим	250	5	Воронеж	7/01/1980	10

STUDENT_ID — числовой код, идентифицирующий 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 — числовой код, идентифицирующий университет, в котором учится студент.

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 — числовой код, идентифицирующий преподавателя,
SURNAME — фамилия преподавателя,
NAME — имя преподавателя,
CITY — город, в котором живет преподаватель,
UNIV_ID — идентификатор университета, в котором работает преподаватель.

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.

UNIVERSITY (Университеты)

UNIV_ID	UNIV NAME	RATING	CITY
22	МГУ	606	Москва
10	ВГУ	296	Воронеж
11	НГУ	345	Новосибирск
32	РГУ	416	Ростов
14	БГУ	326	Белгород
15	ТГУ	368	Томск
18	ВГМА	327	Воронеж

UNIV_ID — идентификатор университета,
UNIV_NAME — название университета,
RATING — рейтинг университета,
CITY — город, в котором расположен университет.

EXAM_MARKS (Экзаменационные оценки)

EXAM_ID	STUDENT ID	SUBJ ID	MARK	EXAM_DATE
145	12	10	5	12/01/2000
34	32	10	4	23/01/2000
75	55	10	5	05/01/2000
238	12	22	3	17/06/1999
639	55	22	NULL	22/06/1999
43	6	22	4	18/01/2000

EXAM_ID — идентификатор экзамена,
STUDENT_ID — идентификатор студента,
SUBJ_ID — идентификатор предмета обучения,
MARK — экзаменационная оценка,
EXAM DATE — дата экзамена.

SUBJ LECT (Учебные дисциплины преподавателей)

LECTURER _ID	SUBJJCD
24	24
46	46
74	74
108	108
276	276
328	328

LECTURER_ID — идентификатор преподавателя,
SUBJ ID — идентификатор предмета обучения.

Вопросы

1. Какие поля приведенных таблиц являются первичными ключами?
2. Какие данные хранятся в столбце 2 таблицы «Предмет обучения»?
3. Как по-другому называется строка? Столбец?
4. Почему нельзя запросить для просмотра первые пять строк?

2 Выборка данных (оператор SELECT)

2.1. Простейшие SELECT-запросы

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

```
SELECT [DISTINCT] <список атрибутов>
    FROM <список таблиц>
    [WHERE <условие выборки>]
    [ORDER BY <список атрибутов>]
    [GROUP BY <список атрибутов>]
    [HAVING <условие>]
    [UNION <выражение с оператором SELECT>J;
```

В квадратных скобках указаны элементы, которые могут отсутствовать в запросе.

Ключевое слово SELECT сообщает базе данных, что данное предложение является запросом на ***извлечение*** 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,

```
SELECTNAME,SURNAME
    FROM STUDENT;
```

Любой SQL-запрос должен заканчиваться символом «;» ***(точ ка с запятой).***

Приведенный запрос осуществляет выборку всех значений полей NAME и SURNAME из таблицы STUDENT.

Его результатом является таблица следующего вида:

NAME	SURNAME
Иван	Иванов
Петр	Петров
Вадим	Сидоров
Борис	Кузнецов
Ольга	Зайцева
Андрей	Павлов
Павел	Котов
Артем	Лукин
Антон	Петров
Вадим	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 можно вместо перечисления их имен использовать символ «*» (звездочка).

```
SELECT *
   FROM STUDENT;
```

В данном случае результатом выполнения запроса будет вся таблица STUDENT.

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

Например, запрос «Получить список названий городов, где проживают студенты, сведения о которых находятся в таблице STUDENT», можно записать в следующем виде.

```
SELECT CITY FROM STUDENT;
```

Его результатом будет таблица:

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 идентичны.

Предыдущий запрос можно записать в следующем виде.

```
SELECT DISTINCT CITY
   FROM STUDENT;
```

В результате получим таблицу, в которой дубликаты строк исключены.

CITY
Орел
Курск
Москва
Брянск
Липецк
Воронеж
Белгород
NULL

ч

Ключевое слово ALL (все), в отличие от DISTINCT, оказывает противоположное действие, то есть 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.

Использование в операторе SELECT предложения, определяемого ключевым словом WHERE (где), позволяет задавать выражение условия (предикат), принимающее значение *истина* или *ложь* для значений полей строк таблиц, 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 студентов 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	NAME
Петров	Петр
Петров	Антон

В задаваемых в предложении WHERE условиях могут использоваться операции сравнения, определяемые операторами = (равно), > (больше), < (меньше), >= (больше или равно), <— (меньше или равно), <> (не равно), а также логические операторы AND, OR и NOT.

Например, запрос для получения *имен* и *фамилий* студентов, обучающихся на *третьем* курсе и получающих стипендию (размер стипендии *больше нуля),* будет выглядеть таким образом:

```
SELECT NAME, SURNAME
  FROM STUDENT
  WHERE KURS = 3 AND STIPEND > 0;
```

Результат выполнения этого запроса имеет вид:

SURNAME	NAME
Петров	Петр
Лукин	Артем

Упражнения

1. Напишите запрос для вывода идентификатора (номера) предмета обучения, его наименования, семестра, в котором он читается, и количества отводимых на этот предмет часов для всех строк таблицы SUBJECT.
2. Напишите запрос, позволяющий вывести все строки таблицы 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 выводит наименование предмета обучения (SUBJJSIAME) и количество часов (HOUR) для каждого предмета (SUBJECT) в 4-м семестре (SEMESTER).
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 выводит список фамилий студентов, обучающихся на третьем и последующих курсах.
7. Напишите запрос, выбирающий данные о фамилии, имени и номере 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 университетов, рейтинг которых превышает 300 баллов.
10. Напишите запрос к таблице STUDENT для вывода списка фамилий (SURNAME), имен (NAME)- и номера курса (KURS) всех студентов со стипендией, большей или равной 100, и живущих в Воронеже.
11. Какие данные будут получены в результате выполнения запроса?

```
SELECT *
   FROM STUDENT
   WHERE (STIPEND < 100 OR
      NOT (BIRTHDAY >= '10/03/1980'
      AND STODENT_ID > 1003));
```

12. Какие данные будут получены в результате выполнения запроса?

```
SELECT *
   FROM STUDENT
   WHERE NOT ((BIRTHDAY = '10/03/1980' OR STIPEND > 100)
      AND STUDENT_ID >= 1003);
```

2.2. 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 равен ни одному из списка) используются для сравнения проверяемого значения поля с заданным списком. Этот список значений указывается в скобках справа от оператора IN.

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

Предикат, построенный с использованием NOT IN, считается истинным, если значение поля, имя которого указано слева от NOT IN, *не совпадает* ни с одним из значений, перечисленных в списке, указанном в скобках справа от NOT IN.

Примеры

Получить из таблицы EXAM_MARKS сведения о студентах, *имеющих* экзаменационные оценки только 4 и 5.

```
SELECT *
   FROM EXAM_MARKS
   WHERE MARK IN (4, 5 );
```

Получить сведения о студентах, *не имеющих* ни одной экзаменационной оценки, равной 4 и 5.

```
SELECT *
   FROM EXAM_MARKS
   WHERE MARK NOT IN (4, 5 );
```

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

Например, запрос на вывод записей о предметах, на изучение которых отводится количество часов, находящееся 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 символьных типов полей.

Оператор LIKE применим только к символьным полям типа CHAR или VARCHAR (см. раздел 1.5 «Типы данных SQL»).

Этот оператор просматривает строковые значения полей с целью определения, входит ли заданная в операторе LIKE подстрока (образец поиска) в символьную строку-значение проверяемого поля.

Для выборки строковых значений по заданному образцу подстроки можно применять шаблон искомого образца строки, использующий следующие символы:

- символ подчеркивания «_», указанный в шаблоне, определяет возможность наличия в указанном месте *одного любого* символа;
- символ «%» допускает присутствие в указанном месте проверяемой строки последовательности любых символов произвольной длины.

Пример

Написать запрос, выбирающий из таблицы STUDENT сведения о студентах, фамилии которых начинаются на букву «Р».

```
SELECT *
   FROM STUDENT
   WHERE SURNAME LIKE 'P%';
```

В случае необходимости включения в образец самих символов «_» и «%» применяют так называемые *escape-символы*. Если escape-символ предшествует знаку «_» и «%», то эти знаки будут восприниматься буквально. Например, можно задать образец поиска с помощью следующего выражения:

LIKE '__P' ESCAPE 'V.

В этом выражении символ 'V с помощью ключевого слова ESCAPE объявляется escape-символом. Первый символ «_» в заданном шаблоне поиска '__P' будет соответствовать, как и ранее, любому символу в проверяемой строке. Однако второй символ «_», следующий после символа 'V, объявленного escape-символом, уже будет интерпретироваться буквально как обычный символ, так же как и символ 'Р' в заданном шаблоне.

Обращаем внимание на то, что рассмотренные выше операторы сравнения «=, <, >, <=, >=, <>» и операторы IN, BETWEEN и LIKE ни в коем случае нельзя использовать для про-

верки содержимого поля на наличие в нем пустого значения NULL (см. раздел 1.5 «Типы данных SQL»). Для этих целей предназначены специальные операторы is NULL (является пустым) и IS NOT NULL (является не пустым).

Упражнения

1. Напишите запрос на вывод находящихся в таблице EXAM_MARKS номеров предметов обучения, экзамены по которым сдавались между 10 и 20 января 1999 года.
2. Напишите запрос, выбирающий данные обо всех предметах обучения, экзамены по которым сданы студентами, имеющими идентификаторы 12 и 32.
3. Напишите запрос на вывод названий предметов обучения, начинающихся на букву «И».
4. Напишите запрос, выбирающий сведения о студентах, у которых имена начинаются на буквы «И» или «С».
5. Напишите запрос для выбора из таблицы EXAM_MARKS записей, в которых отсутствуют значения оценок (поле MARK).
6. Напишите запрос на вывод из таблицы EXAM_MARKS записей, имеющих в поле MARK значения оценок.

2.3. Преобразование вывода и встроенные функции

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

2.3.1. Числовые, символьные и строковые константы

Несмотря на то, что SQL работает с данными в понятиях строк и столбцов таблиц, имеется возможность применения значений выражений, построенных с использованием встроенных функций, констант, имен столбцов, определяемых как своего рода виртуальные столбцы. Они помещаются в списке столбцов и могут сопровождаться псевдонимами.

Если в запросе вместо спецификации столбца SQL обнаруживает *число,* то оно интерпретируется как *числовая константа.*

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

Например, результатом выполнения запроса

```
SELECT 'Фамилия', SURNAME, 'Имя', NAME, 100
  FROM STUDENT;
```

является таблица следующего вида:

	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

2.3.2. Арифметические операции для преобразования числовых данных

- Унарный (одиночный) оператор «—» (знак минус) изменяет знак числового значения, перед которым он указан, на противоположный.

- Бинарные операторы «+», «—», «*» и «/» предоставляют возможность выполнения арифметических операций сложения, вычитания, умножения и деления.

  Например, результат запроса

```
SELECT SURNAME, NAME, STIPEND, -(STIPEND*KURS)/2
  FROM STUDENT
  WHERE KURS = 4 AND STIPEND > 0;
```

выглядит следующим образом:

SURNAME	NAME	STIPEND	KURS	
Сидоров	Вадим	150	4	-300
Петров	Антон	200	4	-400

2.3.3. Операция конкатенации строк

Операция конкатенации «||» позволяет соединять («склеивать») значения двух или более столбцов символьного типа или символьных констант в одну строку.

Эта операция имеет синтаксис

<значимое символьное выражение > {||} <значимое символьное выражение>.

Например:

```
SELECT SURNAME ||'_'|| NAME, STIPEND
  FROM STUDENT
  WHERE KURS = 4 AND STIPEND > 0;
```

Результат запроса будет выглядеть следующим образом:

	STIPEND
Сидоров_Вадим	150
Петров_Антон	200

2.3.4. Функции преобразования символов 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 прописную (заглавную)

  INITCAP (<строка>)

  Например:

```
SELECT  LOWER(SURNAME),  UPPER(NAME)
        FROM STUDENT
        WHERE KURS = 4 AND STIPEND > 0;
```

Результат запроса будет выглядеть следующим образом:

- SURNAME	NAME
сидоров	ВАДИМ
петров	АНТОН

2.3.5. Строковые функции

- LPAD — дополнение строки слева

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

  - <строка> дополняется **слева** заданной в <подстроке> последовательностью символов до указанной <длины> (возможно, с повторением последовательности);
  - если <подстрока> не указана, то по умолчанию <строка> дополняется пробелами;
  - если <длина> меньше длины <строки>, то исходная <строка> усекается слева до заданной <длины>.
- RPAD — дополнение строки справа

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

- <строка> дополняется **справа** заданной в <подстроке> последовательностью символов до указанной <длины> (возможно, с повторением последовательности);
- если <подстрока> не указана, то по умолчанию <строка> дополняется пробелами;
- если <длина> меньше длины <строки>, то исходная <строка> усекается справа до заданной <длины >.

- LTRIM — удаление левых граничных символов

  LTRIM (<строка>[,<подстрока>])

  - из <строки> удаляются слева символы, указанные в <подстроке>;
  - если <подстрока> не указана, по умолчанию удаляются пробелы;
  - в <строку> справа добавляется столько пробелов, сколько символов слева было удалено, то есть длина <строки> остается неизменной.

- RTRIM — удаление правых граничных символов

  RTRIM (<строка>[,<подстрока>])

  - из <строки> удаляются справа символы, указанные в <подстроке>;
  - если <подстрока> не указана, по умолчанию удаляются пробелы;
  - в <строку> слева добавляется столько пробелов, сколько символов справа было удалено, то есть длина <строки> остается неизменной.

Функции LTRIM и RTRIM рекомендуется использовать при написании условных выражений, в которых сравниваются текстовые строки. Дело в том, что наличие начальных или конечных пробелов в сравниваемых операндах может исказить результат сравнения.

Например, константы **' AAA'** и **'AAA '** не равны друг другу.

- SUBSTR — выделение подстроки

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

  - из <строки> выбирается заданное <количество> символов, начиная с указанной параметром <начало> позиции в строке;

- если <количество> не задано, символы выбираются с <начала> и до конца <строки>;
- возвращается подстрока, содержащая число символов, заданное параметром <количество>, либо число символов от позиции, заданной параметром <начало> до конца *строки;*
- 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 подстроки. Если не задан, то по умолчанию принимается значение 1;
- значимые выражения в <начале поиска> или в <номере вхождения> должны иметь беззнаковый целый тип или приводиться к этому типу;
- тип возвращаемого значения — INT;

функция возвращает позицию найденной подстроки.

• LENGTH — определение длины строки

```
ыагстн(<строка>)
```

- длина <строки>, тип возвращаемого значения — ЮТ;
- функция возвращает NULL, если <строка> имеет NULL-значение.

Примеры запросов, использующих строковые функции

Результат запроса

```
SELECT LPAD (SURNAME, 10, '@'), RPAD (NAME, 10, '$')
   FROM STUDENT
   WHERE KURS = 3 AND STIPEND > 0;
```

будет выглядеть следующим образом:

@@@@Петров	Петр$$$$$$
@@@@Павлов	Андрей$$$$
@@@@@Дукин	Артем$$$$$

А запрос

```
SELECT SUBSTR(NAME, 1, 1) Ц'.' || 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

2.3.6. Функции работы с числами

- ABS — абсолютное значение

  ABS (<значимое числовое выражение>)

- FLOOR — урезает значение числа с плавающей точкой до наибольшего целого, не превосходящего заданное число

  FLOOR (<значимое числовое выражение>)

- CEIL — самое малое целое, равное или большее 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 (оначимое 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 >)

- Экспоненциальная функция — EXP

  EXP (<значимое числовое выражение>)

- Логарифмические функции — LN, LOG

  ш (<значимое числовое выражение>)

  LOG (<значимое числовое выражение>)

- Функция возведения в степень — POWER

  POWER (<значимое числовое выражение>,<экспонента>)

- Определение знака числа — SIGN

  SIGN (<значимое числовое выражение>)

- Вычисление квадратного корня — SQRT

  SQRT (<значимое числовое выражение>)

Пример

Запрос

```
SELECT UNIVJffiME, RATING, ROUND (RATING, -1), TRDNC (RATING,
   FROM UNIVERSITY;
```

вернет результат:

UNIV NAME	RATING		
МГУ	606	610	600
ВГУ	296	300	290
НГУ	345	350	340
РГУ	416	420	410
БГУ	326	330	320
ТГУ	368	370	360
ВГМА	327	330	320

2.3.7. Функции преобразования значений

- Преобразование в символьную строку — TO_CHAR

TO_CHAR (<значимое выражение>[,<символьный формат>])

- <значимое выражение> — числовое значение 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 отображения знака), 9 — представление цифр-знаков числового значения (для каждого знакоместа). Символьный формат определяет вид отображения чисел. По умолчанию для числовых значений используется формат '999999.99';
- для значений типа дата-время <символьный формат> имеет вид (то есть вид отображения значений даты и времени):

— в части даты

'DD-Mon-YY'
'DD-Mon-YYYY'
'MM/DD/YY'
'MM/DD/YYYY'[1]
'DD.MM.YY'
'DD.MM.YYYY'

— в части времени

'HH24'
ИН24.Ш'
'HH24:MI:SS'
'HH24:MI:SS.FF'

ще: HH24 — часы в диапазоне от 0 до 24
R/FI — минуты
SS — секунды
FF — тики (сотые доли секунды)

При выводе времени в качестве разделителя по умолчанию используется 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

TCMDATE (<значимое символьное выражение >[,<символьный формат>])

- <значимое символьное выражение> должно задавать символьное значение типа **дата-время;**
- <символьный формат> должен описывать представление значения типа дата-время в <значимом символьном выражении>. 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:

- бинарная операция сложения;
- бинарная операция вычитания.

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

Например:

- при добавлении к дате '22.05.1998' пяти лет получится дата '22.05.2003';
- при добавлении к этой же дате девяти месяцев получится дата '22.02.1998';
- при добавлении 10 дней получим '01.06.1998'.

При сложении двух полных дат, например, '22.05.1998' и '01.12.2000', результат непредсказуем.

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	3/12/1982	3-дек-1982	3.12.82
Петров	Петр	1/12/1980	1-дек-1980	1.12.80
Сидоров	Вадим	7/06/1979	7-июн-1979	7.06.79
Кузнецов	Борис	8/12/1981	8-дек-1981	8.12.81
Зайцева	Ольга	1/05/1981	1-МЗЙ-1981	1.05.81
Павлов	Андрей	5/11/1979	5-ноя-1979	5.11.79
Котов	Павел	NULL	NULL	NULL
Лукин	Артем	1/12/1981	1-дек-1981	1.12.81
Петров	Антон	5/08/1981	5-авг-1981	5.08.81
Белкин	Вадим	7/01/1980	7-янв-1980	7.01.80

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

сАзКзначимое выражение> AS <тип данных>

- <значимое выражение> должно иметь числовой или символьный тип языка SQL (возможно, с указанием длины, точности и масштаба) или быть NULL-значением;
- любое числовое выражение может быть явно преобразовано в любой другой числовой тип;
- символьное выражение может быть преобразовано в любой числовой тип. При этом в результате символьного выражения отсекаются начальные и конечные пробелы, а остальные символы преобразуются в числовое значение по правилам языка SQL;
- если явно заданная длина символьного типа недостаточна и преобразованное значение не размещается в нем, то результативное значение усекается справа;
- возможно явное преобразование символьного типа в символьный с другой длиной. Если длина результата больше длины аргумента, то значение дополняется пробелами; если меньше, то усекается;
- NULL-значение преобразуется в NULL-значение соответствующего типа;
- числовое выражение может быть преобразовано в символьный тип.

Пример

```
SELECT CAST STUDENT_JD AS CHAR(10)
   FROM STUDENT;
```

Упражнения

1. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала один столбец, содержащий последовательность разделенных символом «;» (точка с запятой) значений всех столбцов этой таблицы, и при этом текстовые значения должны отображаться прописными символами (верхний регистр), то есть быть представленными в следующем виде: 10;КУЗНЕЦОВ;БОРИС;0;БРЯНСК;8/12/1981;10.

2. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Б.КУЗНЕЦОВ;местожительства-БРЯНСК;родился-8.12.81.
3. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: б.кузнецов;место жительства-брянск;родился:8-дек-1981.
4. Составьте запрос для таблицы STUDENT таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Борис Кузнецов родился в 1981 году.
5. Вывести фамилии, имена студентов и величину получаемых ими стипендий, при этом значения стипендий должны быть увеличены в 100 раз.
6. То же, что и в задаче 4, но только для студентов 1, 2 и 4-го курсов и таким образом, чтобы фамилии и имена были выведены прописными буквами.
7. Составьте запрос для таблицы UNIVERSITY таким образом, чтобы выходная таблица содержала всего один столбец в следующем виде: Код-10;ВГУ-г. ВОРОНЕЖ;Рейтинг=296.
8. То же, что и в задаче 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 используются следующие ключевые слова:

- COUNT определяет количество строк или значений поля, выбранных 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 аналогично 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:

```
SELKCT AVERAGE (MARK)
   PROM EXAM_MARKS;
```

Для подсчета общего количества строк в таблице следует использовать функцию COUNT со звездочкой.

```
SELECT COUNT(*)
   FROM EXAM_MARKS;
```

Аргументы DISTINCT и ALL позволяют, соответственно, исключать и включать дубликаты обрабатываемых функцией COUNT значений, при этом необходимо учитывать, что при использовании опции ALL значения NULL все равно не войдут в число подсчитываемых значений.

```
SELECT COUNT(DISTINCT SUBJ_IDj
   FROM SUBJECT;
```

Предложение GROUP BYGROUP BY (группировать по) позволяет группировать 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, полученной каждым студентом. Запрос будет выглядеть следующим образом:

```
SELECT STUDENT_ID, MAX(MARKj
   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 зна-

чение поля MARK. Предложение GROUP BY позволяет применять агрегирующие функции к каждой группе, определяемой общим значением поля (или полей), указанных в этом предложении. В приведенном запросе рассматриваются группы записей, сгруппированные по идентификаторам студентов.

В конструкции GROUP BY для группирования может быть использовано более одного столбца. Например:

```
SELECT STUDENT_ID, SUBJ_ID, MAX (MARKj
   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 группирование, но и указывает порядок разбиения столбцов на группы.

Следует иметь в виду, что в предложении GROUP BY должны быть указаны все выбираемые столбцы, приведенные после ключевого слова SELECT, кроме столбцов, указанных в качестве аргумента в агрегирующей функции.

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(HOURj
   FROM SUBJECT
   GROUP BY SUBJ_NAME
   HAVING MAX (HOURj >= 72;
```

В условии, задаваемом предложением HAVING, указывают только поля или выражения, которые на выходе имеют единственное значение для каждой выводимой группы.

Упражнения

1. Напишите запрос для подсчета количества студентов, сдававших экзамен по предмету обучения с идентификатором, равным 20.

2. Напишите запрос, который позволяет подсчитать в таблице EXAM_MARKS количество различных предметов обучения.
3. Напишите запрос, который выполняет выборку для каждого студента значения его идентификатора и минимальной из полученных им оценок.
4. Напишите запрос, осуществляющий выборку для каждого студента значения его идентификатора и максимальной из полученных им оценок.
5. Напишите запрос, выполняющий вывод фамилии первого в алфавитном порядке (по фамилии) студента, фамилия которого начинается на букву «И».
6. Напишите запрос, который выполняет вывод (для каждого предмета обучения) наименования предмета и максимального значения номера семестра, в котором этот предмет преподается.
7. Напишите запрос, который выполняет вывод данных для каждого конкретного дня сдачи экзамена о количестве студентов, сдававших экзамен в этот день.
8. Напишите запрос для получения среднего балла для каждого курса по каждому предмету.
9. Напишите запрос для получения среднего балла для каждого студента.
10. Напишите запрос для получения среднего балла для каждого экзамена.
11. Напишите запрос для определения количества студентов, сдававших каждый экзамен.
12. Напишите запрос для определения количества изучаемых предметов на каждом курсе.

2.5. Пустые значения (NULL) в агрегирующих функциях

Наличие пустых (NULL) значений в полях таблицы определяет особенности выполнения агрегирующих операций над данными, которые следует учитывать в SQL-запросах.

2.5.1. Влияние **NULL-значений** в функции COUNT

Если аргумент функции COUNT является константой или столбцом без пустых значений, то функция возвращает количество строк, к которым применимо 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 в COUNT условие или группирование.

Если бы механизм NULL не был доступен, то неприменимые и отсутствующие значения пришлось бы исключать с помощью конструкции WHERE.

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

2.5.2. Влияние NULL-значений в функции AVG

Среднее значение множества чисел равно сумме чисел, деленной на число элементов множества. Однако если некоторые элементы пусты (то есть их значения неизвестны или не существуют), деление на количество всех элементов множества приведет к неправильному результату.

Функция AVG вычисляет среднее значение всех *известных* значений множества элементов, то есть 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.

2.6. Результат действия трехзначных условных операторов

Условные операторы при отсутствии пустых значений возвращают 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, условие WHERE A = 2, где A — 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 оказывают влияние на использование логических операторов NOT, AND и OR.

Оператор NOT

Обычный унарный оператор NOT обращает оценку TRUE в FALSE и наоборот. Однако NOT NULL по прежнему будет возвращать пустое значение NULL. При этом следует отличать случай NOT NULL от условия IS NOT NULL, которое является противоположностью IS NULL, отделяя известные значения от неизвестных.

Оператор AND

- Если результат двух условий, объединенных оператором AND, известен, то применяются правила булевой логики, то есть 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 является определяющим, и, следовательно, итоговый результат также неизвестен.
- Если результат одного из утверждений неизвестен, а другой оценивается как FALSE, итоговый результат будет FALSE.
- Если результат обоих утверждений неизвестен, то результат также остается неизвестным.

Оператор OR

- Если результат двух условий, объединенных оператором OR, известен, 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 оцениваются как FALSE, то составное утверждение будет FALSE.
- Если результат одного из утверждений неизвестен, а другой оценивается как TRUE, итоговый результат будет TRUE.
- Если результат одного из утверждений неизвестен, а другой оценивается как FALSE, то состояние неизвестного утверждения имеет определяющее значение. Следовательно, итоговый результат также неизвестен.
- Если результат обоих утверждений неизвестен, то результат также остается неизвестным.

Примечание

Отсутствующие (NULL) значения целесообразно использовать в столбцах, предназначенных для агрегирования, чтобы извлечь преимущества из способа обработки пустых значений в функциях COUNT и AVERAGE. Практически во всех остальных случаях пустых значений следует избегать, так как при их наличии существенно усложняется корректное построение условий отбора, приводя иногда к непредсказуемым результатам выборки. Для индикации же отсутствующих, неприменимых или по какой-то причине неизвестных данных можно использовать значения по умолчанию, устанавливаемые заранее (например, с помощью команды CREATE TABLE (раздел 4.1)).

2.7. Упорядочение выходных полей (ORDER BY)

Как уже отмечалось, записи в таблицах реляционной базы данных не упорядочены. Однако данные, выводимые в результате выполнения запроса, могут быть упорядочены. Для этого используется оператор ORDER BY, который позволяет упорядочивать выводимые записи в соответствии со значениями одного или нескольких выбранных столбцов. При этом можно задать возрастающую (ASC) или убывающую (DESC) последовательность 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:

```
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 значениям семестров, а внутри семестров — по наименованиям предметов.

```
SELECT *
   FROM SUBJECT
   ORDER BY SEMESTER, SUBJ_NAME;
```

Предложение ORDER BY может использоваться с GROUP BY для упорядочения групп записей. При этом оператор ORDER BY в запросе *всегда должен быть последним.*

```
SELECT SUBJ_NAME, SEMESTER, MAX(HOUR)
   FROM SUBJECT
   GROUP BY SEMESTER, SUBJ_NAME
   ORDER BY SEMESTER;
```

При упорядочении вместо наименований столбцов можно указывать 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 расположения в таблице.

```
SELECT SUBJ_ID, SEMESTER
   FROM SUBJECT
   ORDER BY 2 DESC;
```

В этом запросе выводимые записи будут упорядочены по полю SEMESTR.

Если в поле, которое используется для упорядочения, существуют 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 номера студента, фамилию студента и величину увеличенной стипендии. Выходные данные упорядочить: а) по значению последнего столбца (величине стипендии); б) в алфавитном порядке фамилий студентов.

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

2.8. Вложенные подзапросы

SQL позволяет использовать одни запросы внутри других запросов, то есть вкладывать запросы друг в друга. Предположим, известна фамилия 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');
```

Как работает запрос SQL со связанным подзапросом?

- Выбирается строка из таблицы, имя которой указано во внешнем запросе.
- Выполняется подзапрос и полученное значение применяется для анализа этой строки в условии предложения WHERE внешнего запроса.
- По результату оценки этого условия принимается решение о включении или не включении строки в состав выходных данных.

- Процедура повторяется для следующей строки таблицы внешнего запроса.

Следует обратить внимание, что приведенный выше запрос корректен только в том случае, если в результате выполнения указанного в скобках подзапроса возвращается *единственное значение. Если* в результате выполнения подзапроса будет возвращено несколько значений, то этот подзапрос будет ошибочным. В данном примере это произойдет, если в таблице STUDENT будет несколько записей со значениями поля SURNAME = 'Петров'.

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

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

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

```
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, превышающей среднее значение оценки студента с идентификатором 301:

```
SELECT COONT(DISTINCT SUBJ_IDj, MARK
   FROM EXAM_MARKS
   GROUP BY MARK
   HAVING MARK >
```

```
 (SELECT AVG(MARK)
FROM EXAM_MARKS
WHERE STUDENT_ID = 301);
```

2.9. Формирование связанных подзапросов

При использовании подзапросов во внутреннем запросе можно ссылаться на таблицу, имя которой указано в предложении FROM внешнего запроса. В этом случае такой *связанный* подзапрос выполняется по одному разу для *каждой* строки таблицы основного запроса.

Пример: выбрать сведения обо всех предметах обучения, по которым проводился экзамен **20** января 1999 **г.**

```
SELECT *
    FROM SUBJECT SU
    WHERE '20/01/1999' IN
         (SELECT EXAM_DATE
         PROM EXAM_MARKS EX
         WHERE SU.SUBJ_ID = EX.SUBJ_ID);
```

В некоторых СУБД для выполнения этого запроса может потребоваться преобразование значения даты в символьный тип. В приведенном запросе su и EX являются псевдонимами (алиасами), то есть 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 SU.SUBJ_ID, SUBJJAME, HOUR, SEMESTER
    PROM SUBJECT FIRST,EXAM_MARKS SECOND
    WHERE FIRST.SUBJ_ID = SECOND.SUBJ_ID
    AND SECOND.EXAM_DATE = '20/01/1999';
```

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 курсе, 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:

```
SELECT DISTINCT STUDENT_ID, SURNAME, STIPEND
   FROM STUDENT E1,
      (SELECT KURS, AVG (STIPENDj 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 будет выполнен 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 таблица (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 реальное время выполнения запроса в большой степени зависит от оптимизатора запросов конкретной СУБД.

2.10. Связанные подзапросы в HAVING

В разделе 2.4 указывалось, что предложение GROUP BY позволяет 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 с одной и той же датой, совпадающей с датой, для которой сформирована очередная группа основного запроса.

Упражнения

1. Напишите **запрос с подзапросом для** получения **данных обо всех** оценках студента **с** фамилией «Иванов». Предположим, что его персональный **номер** неизвестен. **Всегда** ли такой **запрос** будет корректным?
2. Напишите запрос, выбирающий **данные об** именах **всех** студентов, имеющих **по** предмету **с** идентификатором **101** балл **выше общего среднего** балла.
3. Напишите запрос, который выполняет выборку имен **всех** студентов, имеющих **по** предмету **с** идентификатором **102** балл ниже **об щего среднего** балла
4. Напишите запрос, выполняющий **вывод** количества предметов, по **которым** экзаменовался каждый студент, сдававший более **20** предметов.
5. Напишите команду SELECT, использующую **связанные** подзапросы и выполняющую **вывод** имен **и** идентификаторов студентов,

у которых стипендия совпадает с максимальным значением стипендии для города, в котором живет студент.

6. Напишите запрос, который позволяет вывести имена и идентификаторы всех студентов, для которых точно известно, что они про живают в городе, где нет ни одного университета.
7. Напишите два запроса, которые позволяют вывести имена и идеи тификаторы всех студентов, для которых точно известно, что он проживают не в том городе, где расположен их университет. Оди запрос с использованием соединения, а другой — с использовани ем связанного подзапроса.

2.11. Использование оператора EXISTS

Используемый в SQL оператор EXISTS (существует) гене рирует значение **истина** или **ложь,** подобно булеву выражению Используя подзапросы в качестве аргумента, этот операто оценивает результат выполнения подзапроса как истинный если этот подзапрос генерирует выходные данные, то ест в случае *существования (возврата) хотя бы одного найденного значения.* В противном случае результат подзапроса ложный Оператор EXISTS не может принимать значение UNKNOWN (не известно).

Пусть, например, нужно извлечь из таблицы EXAM_MARK данные о студентах, получивших хотя бы одну неудовлетвори тельную 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 проверку условия. Для каждой строки-кандидата выполняется подзапрос. Как только подзапрос находит

строку, где в столбце MARK значение удовлетворяет условию, он прекращает выполнение и возвращает значение **истина** внешнему запросу, который затем анализирует свою строку-кандидата.

Например, требуется получить идентификаторы предметов обучения, экзамены по которым сдавались не одним, а несколькими студентами:

```
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 предыдущий запрос сформулировать следующим образом — найти идентификаторы предметов обучения, которые сдавались одним, и только одним студентом (другими словами, для которых не существует другого сдававшего студента), то достаточно просто поставить NOT перед EXISTS.

Следует иметь в виду, что в подзапросе, указываемом в операторе EXISTS, *нельзя использовать агрегирующие функции.*

Возможности применения вложенных запросов весьма разнообразны. Например, пусть из таблицы STUDENT требуется извлечь строки для каждого студента, сдавшего более одного I предмета.

```
SELECT *
   FROM STUDENT FIRST
   WHERE EXISTS
       (SELECT SUBJ_ID
       FROM EXAM_MARKS SECOND
       GROUP BY SUBJ_ID
       HAVING COUNT (SUBJ_IDj > 1
       WHERE FIRST.STUDENT ID = SECOND.STUDENT ID);
```

Упражнения

1. Напишите запрос с EXISTS, позволяющий вывести данные обо всех студентах, обучающихся в вузах, которые имеют рейтинг выше 300
2. Напишите предыдущий запрос, используя соединения.
3. Напишите запрос с EXISTS, выбирающий сведения обо всех студентах, для которых в том же городе, где живет студент, существуют университеты, в которых он не учится.
4. Напишите запрос, выбирающий из таблицы SUBJECT данные о названиях предметов обучения, экзамены по которым *сданы* более чем одним студентом.

2.12. Операторы сравнения с множеством значений IN, ANY, All

Операторы сравнения с множеством значений имеют следующий смысл.

IN	*Равно* любому 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 оператору OR.
> ANY, > = ANY	*Больше, чем* (либо *больше или равно)* любое полученное число. Эквивалентно > или > = для самого меньшего полученного числа.
< ANY, < = ANY	*Меньше, чем* (либо *меньше или равно)* любое полученное число. Эквивалент < или < = для самого большего полученного числа.
= ALL	Равно всем полученным значениям. Эквивалентно логическому оператору AND
> ALL, > = ALL	*Больше, чем* (либо *больше или равно)* все полученные числа. Эквивалент > или > = для самого большего полученного числа.

< ALL, < = ALL	*Меньше, чем* (либо *меньше или равно)* все полученные числа. Эквивалентно < или < = самого меньшего полученного числа.

Следует иметь в виду, что в некоторых СУБД поддерживаются не все из этих операторов.

Примеры запросов **с** использованием приведенных операторов

Выбрать сведения о студентах, проживающих в городе, где расположен университет, в котором они учатся.

```
SELECT *
   FROM STUDENT S
   WHERE CITY = ANY
       (SELECT  CITY
       FROM UNIVERSITY U
       WHERE U.UNIV_ID = S.UNIV_ID);
```

Другой вариант этого запроса:

```
SELECT *
   FROM STUDENT S
   WHERE CITY  IN
       (SELECT CITY
       FROM UNIVERSITY U
       WHERE U.UNIV_ID = S . UNIV_ID);
```

Выборка данных об идентификаторах студентов, у которых оценки превосходят величину, по крайней мере, одной из оценок, полученных ими же 6 октября 1999 года.

```
SELECT  DISTINCT STUDENT_ID
   FROM  EXAM_MARKS
   WHERE MARK > ANY
       (SELECT  MARK
       FROM EXAM_MARKS
       WHERE  EXAM  DATE  =  '06/10/1999');
```

Оператор ALL, как правило, эффективно используется с неравенствами, 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 подзапрос генерирует множество различных значений, то никакое одно значение не может быть равно сразу всем значениям в обычном смысле. В SQL выражение < > ALL реально означает *не равно ни одному* из результатов подзапроса.

Подзапрос, выбирающий данные о названиях всех университетов с рейтингом более высоким, чем рейтинг любого университета Воронежа:

```
SELECT *
   FROM UNIVERSITY
   WHERE RATING > ALL
       (SELECT RATING
      FROM UNIVERSITY
      WHERE CITY = 'Воронеж') ;
```

В этом запросе вместо ALL можно 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');
```

2.13. Особенности применения операторов ANY, ALL, EXISTS при обработке пустых значений (NULL)

Необходимо иметь в виду, что при обработке NULL-значений следует учитывать различие реакции на них операторов EXISTS, ANY И ALL.

Когда правильный подзапрос не генерирует никаких выходных данных, оператор ALL автоматически принимает значение истина, а оператор ANY — значение ложь.

Запрос

```
SELECT *
   FROM UNIVERSITY
   WHERE RATING > ANY
       (SELECT RATING
       FROM UNIVERSITY
       WHERE CITY = 'New York');
```

не генерирует выходных данных (подразумевается, что в базе нет данных об университетах города New York), в то время как запрос

```
SKLECT *
   FROM UNIVERSITY
   WHERE RATING > ALL
       (SELECT RATING
       FROM UNIVERSITY
       WHERE CITY = 'New York');
```

полностью воспроизведет таблицу UNIVERSITY.

Использование NULL-значений создает определенные проблемы для рассматриваемых операторов. Когда в SQL сравниваются два значения, одно из которых NULL-значение, результат принимает значение UNKNOWN (неизвестно). Предикат UNKNOWN, так **же,** как и FALSE-предикат, создает ситуацию, когда строка не включается в состав выходных данных, но результат при этом будет различен для разных типов запросов, в зависимости от использования в них ALL или ANY вместо EXISTS. Рассмотрим в качестве примера две реализации запроса: найти все данные об университетах, рейтинг которых меньше рейтинга любого университета в Москве.

```
1) SELECT *
       FROM UNIVERSITY
       WHERE RATING < ANY
           (SELECT RATING
```

```
    ' PROM UNIVERSITY
      WHERE CITY = 'Москва');
2) SELECT *
     FROM UNIVERSITY A
     WHERE NOT EXISTS
       (SELECT *
       FROM UNIVERSITY B
       WHERE A.RATING >= B.RATING
         AND B.CITY = 'Москва');
```

При отсутствии в таблицах NULL оба эти запроса ведут себя совершенно одинаково. Пусть теперь в таблице UNIVERSITY есть строка с NULL-значениями в столбце RATING. В версии запроса с ANY в основном запросе, когда выбирается поле RATING с NULL, предикат принимает значение UNKNOWN и строка не включается в состав выходных данных. Во втором же варианте запроса, когда NOT EXISTS выбирает эту строку в основном запросе, NULL-значение используется в предикате подзапроса, присваивая ему значение UNKNOWN. Поэтому в результате выполнения подзапроса не будет получено ни одного значения, и подзапрос примет значение **ложь.** Это в свою очередь сделает NOT EXISTS истинным, и, следовательно, строка с NULL-значением в поле RATING попадет в выходные данные. По смыслу запроса такой результат является неправильным, так как на самом деле рейтинг университета, описываемого данной строкой, может быть и больше рейтинга какого-либо московского университета (он просто неизвестен). Указанная проблема связана с тем, что значение EXISTS всегда принимает значения **истина** или ложь, и никогда — UNKNOWN. Это является доводом для использования **в** таких случаях оператора ANY вместо EXISTS.

2.14. Использование COUNT вместо EXISTS

При отсутствии NULL-значений оператор EXISTS может быть использован вместо ANY и ALL. Также вместо EXISTS и NOT EXISTS могут быть использованы те же самые подзапросы, но

с использованием 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 университетов, рейтинг которых равен или превосходит рейтинг Воронежского государственного университета.
2. Напишите запрос, использующий 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 данные о названиях предметов обучения, для которых значение полученных на экзамене оценок (поле MARK) превышает любое значение оценки для предмета, имеющего идентификатор, равный 105.
4. Напишите этот же запрос с использованием MAX.

2.15. 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 использовать следующий запрос:

```
SELECT 'Студент______', SURNAME, ST0DENT_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 следующий за ним запрос так же, как и он сам, является частью общего запроса с UNION.

Использование оператора UNION возможно только при объединении запросов, соответствующие столбцы которых *совместимы по объединению,* то есть соответствующие числовые поля должны иметь полностью совпадающие тип и размер, символьные поля должны иметь точно совпадающее количество символов. Если NULL-значения запрещены для столбца хотя бы одного любого подзапроса объединения, то они должны быть запрещены и для всех соответствующих столбцов в других подзапросах объединения.

2.16. Устранение дублирования в UNION

В отличие от обычных 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 объединяемых 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 экзаменов сведения по каждому студенту, получившему максимальную или минимальную оценки.

```
SELECT 'МЭКСОЦ', 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 'МИНОЦ', 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 отличия строк, выводимых первой и второй частями запроса, в них вставлены текстовые константы 'Макс оц' и 'Мин оц'.

33ак 444

В приведенном запросе агрегирующие функции используются в подзапросах. Это является нерациональным с точки зрения времени, затрачиваемого на выполнение запроса (см. раздел 2.9). Более эффективна форма запроса, возвращающего аналогичный результат:

```
SEIECT 'МаКСОЦ', 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
SEMCT 'МИНОЦ', 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
```

2.17. Использование UNION с ORDER BY

Предложение ORDER BY применяется для упорядочения выходных данных объединения запросов так же, как и для отдельных запросов. Последний пример, при необходимости упорядо-

чения выходных данных запроса по фамилиям студентов и датам экзаменов, может выглядеть следующим образом:

```
SELECT 'МЭКСОЦ', 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 'МИНОЦ', 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;
```

2.18. Внешнее объединение

Часто бывает полезна операция объединения двух запросов, в которой второй запрос выбирает строки, исключенные первым. Такая операция называется внешним объединением.

Рассмотрим пример. Пусть в таблице STUDENT имеются записи о студентах, в которых не указан идентификатор университета. Требуется составить список студентов с указанием наименова-

Соединение, использующее предикаты, основанные на равенствах, называется *эквисоединением*. Рассмотренный пример соединения таблиц относятся к виду так называемого *внутрен него* (INNER) *соединения*. При этом соединяются только те строки таблиц, для которых истинным является предикат, задаваемый в предложении ON выполняемого запроса.

Приведенный выше запрос может быть записан иначе, с использованием 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.

Рассмотренный выше случай полного соединения (декартова произведения) таблиц с использованием ключевого слова JOIN будет выглядеть следующим образом:

```
SELECT * FROM STUDENT JOIN UNIVERSITY;
```

что эквивалентно

```
SELECT * FROM STUDENT, UNIVERSITY ;
```

Заметим, что в СУБД Oracle задаваемый стандартом языка SQL оператор JOIN не поддерживается.

2.19.1. 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 следующий запрос:

```
SELECT SURNAME, MARK, SUBJ_ID
   FROM STUDENT,EXAM_MARKS
   WHERE  STUDENT.STUDENT_ID  =  EXAM_MARKS.STUDENT_ID;
```

Тот же 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;
```

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

Пусть требуется найти фамилии всех студентов, получивших неудовлетворительную 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 SUBJECT JOIN EXAM_MARKS
   ON  STUDENT.STUDENT_ID  =  EXAM_MARKS.STUDENT_ID
      AND  SUBJECT.SUBJ_ID  =  EXAM_MARKS.SUBJ_ID
      AND  EXAM MARKS.MARK  =  2;
```

2.19.2. Внешнее соединение таблиц

Как отмечалось ранее, при использовании *внутреннего* (INNER) соединения таблиц соединяются только те их строки, в которых совпадают значения полей, задаваемые в запросе предложением WHERE. Однако во многих случаях это может привести к нежелательной потере информации. Рассмотрим еще раз приведенный выше пример запроса на выборку списка фамилий студентов 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, отсутствуют в таблице EXAMJMARKS. 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 CHARdV  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 обеспечения совместимости типов полей объединяемых запросов).

Нужный результат может быть получен и путем использования *внешнего соединения,* точнее, одной из его разновидностей — *левого внешнего соединения,* с применением которого 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 таблицы осуществляется за счет записей входной таблицы, имя которой указано *слева* от оператора JOIN.

Следует заметить, что нотация запросов с внешним соединением в СУБД Oracle отличается от приведенной нотации, задаваемой стандартом языка SQL. В нотации, используемой в Oracle, этот же запрос будет иметь 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 отношение попали и те записи другой таблицы, для которых в таблице со знаком (+) не находится строк с соответствующими значениями атрибутов, используемых для соединения. То есть для *левого* внешнего соединения (по нотации стандарта SQL) 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, указана справа от оператора JOIN.

В нотации Oracle этот запрос будет выглядеть следующим образом:

```
SELECT SURNAME, MARK, SUBJ_ID
    FROM STUDENT,EXAM_MARKS
    WHERE EXAM_MARKS.STUDENT_ID(+) = STUDENT.STUDENT_ID;
```

Видно, что использование внешнего правого или левого 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 или UNION JOIN.

Упражнения

1. Напишите запрос, который выполняет вывод данных о фамилиях *сдававших* экзамены студентов (вместе с идентификаторами каждого сданного ими предмета обучения).
2. Напишите запрос, который выполняет выборку значений фамилии *всех* студентов с указанием для студентов, сдававших экзамены, идентификаторов сданных ими предметов обучения.
3. Напишите запрос, который выполняет вывод данных о фамилиях студентов, *сдававших* экзамены, вместе с наименованиями каждого сданного ими предмета обучения.
4. Напишите запрос на выдачу для каждого студента названий всех предметов обучения, по которым этот студент получил оценку 4 или 5.
5. Напишите запрос на выдачу данных о названиях всех предметов, по которым 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 университетов с рейтингом, превышающим 300, вместе со значением максимального размера стипендии, получаемой студентами в этих университетах.
7. Напишите запрос на выдачу списка фамилий студентов (в алфавитном порядке) вместе со значением рейтинга университета, где каждый из них учится, включив в список и тех студентов, для которых в базе данных не указано место их учебы.

2.19.3. Использование псевдонимов при соединении таблиц

Часто при запросе информации необходимо осуществлять соединение таблицы с ее же копией. Например, это требуется в случае, когда нужно найти фамилии студентов, имеющих оди-

наковые 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 предикат асимметричным.

```
SELECT FIRST. SURNAME, SECOND. SURNAME
   FROM STUDENT FIRST, STUDENT SECOND
   WHERE FIRST.NAME = SECOND.NAME
      AND FIRST.SURNAME < SECOND.SURNAME
```

Упражнения

1. Написать запрос, выполняющий вывод списка всех пар фамилий студентов, проживающих в одном городе. При этом не включать в список комбинации фамилий студентов самих с собой (то есть комбинацию типа «Иванов-Иванов») и комбинации фамилий студентов, отличающиеся порядком следования (то есть включать одну из двух комбинаций типа «Иванов-Петров» и «Петров-Иванов»).
2. Написать 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. Написать запрос, который позволяет получить данные о названиях университетов и городов, в которых они расположены, с рейтингом, равным или превышающим рейтинг ВГУ.

3 Манипулирование данными

3.1. Команды манипулирования данными

В SQL для выполнения операций ввода данных в таблицу, **их** изменения и удаления предназначены три команды языка манипулирования данными (DML). Это команды INSERT (вставить), UPDATE (обновить), DELETE (удалить).

Команда INSERT осуществляет **вставку** в таблицу новой строки. **В** простейшем случае она имеет 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 строки в том порядке, **в** котором соответствующие столбцы указаны при создании таблицы, то есть **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[1],
                                              '6/10/1979', 15);
```

Чтобы такая команда могла быть выполнена, таблица с указанным в ней именем (STUDENT) должна быть предварительно определена (создана) командой CREATE TABLE. Если в какое-либо поле необходимо 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 командой CREATE TABLE, или требуется ввести значения не во все столбцы, следует использовать следующую форму команды INSERT:

```
INSERT INTO STUDENT (STUDENT_ID, CITY, SURNAME, NAME)
VALUES (101, 'Москва', 'Иванов', 'Саша');
```

Столбцам, наименования которых не указаны в приведенном в скобках списке, автоматически присваивается значение по умолчанию, если оно назначено при описании таблицы (команда CREATE TABLE), либо значение NULL.

С помощью команды INSERT можно извлечь значение из одной таблицы и разместить его в другой, например, запросом следующего вида:

```
INSERT INTO STUDENT1
  SELECT *
    FROM STUDENT
    WHERE CITY = 'Москва';
```

При этом таблица STUDENTI должна быть предварительно создана командой CREATE TABLE (раздел 4.1) и иметь структуру, идентичную таблице STUDENT.

Удаление строк из таблицы осуществляется с помощью команды DELETE.

Следующее выражение удаляет все строки таблицы EXAM_MARKSI .

```
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, удовлетворяющих некоторому условию, можно воспользоваться предложением WHERE, например:

```
DELETE FROM EXAM_MARKS1
  WHERE STUDENT_ID = 103;
```

Можно удалить группу строк:

```
DELETE FROM STUDENTI
  WHERE CITY = 'Москва';
```

Команда UPDATE позволяет **изменять,** то есть обновлять значения некоторых или всех полей в существующей строке или строках таблицы. Например, чтобы для всех университетов, све-

дения о которых находятся в таблице UNIVERSITYI, изменить рейтинг на значение 200, можно использовать конструкцию:

```
UPDATE UNIVERSITYI
SET RATING = 200;
```

Для указания конкретных строк таблицы, значения полей которых должны быть изменены, в команде UPDATE можно использовать предикат, указываемый в предложении WHERE.

```
UPDATE UNIVERSITYI
SET RATING = 200
WHERE CITY = 'Москва';
```

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

Команда UPDATE позволяет изменять не только один, но и множество столбцов. Для указания конкретных столбцов, значения которых должны быть модифицированы, используется предложение SET.

Например, наименование предмета обучения 'Математика' (для него SUBJ_ID = 43) должно быть заменено на название '**Высшая** математика', при этом идентификационный номер необходимо сохранить, но в соответствующие поля строки таблицы ввести новые данные 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 команды UPDATE можно использовать скалярные выражения, указывающие способ изменения значений поля, в которые могут входить значения изменяемого и других полей.

```
UPDATE UNIVERSITYI
SET RATING = RATING*2;
```

Например, для увеличения в таблице STUDENTi значения поля STIPEND в два раза для студентов из Москвы можно использовать запрос

```
UPDATE STUDENT1
SET STIPEND = STIPEND*2
WHERE CITY = 'Москва';
```

Предложение SET не является предикатом, поэтому в нем можно указать значение NULL следующим образом:

```
UPDATE UNIVERSITY1
SET RATING = NULL
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 Николай, 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 имеющихся в базе данных университетов, расположенных в Санкт-Петербурге.
5. Измените в таблице значение города, в котором проживает студент Иванов, на «Воронеж».

3.2. Использование подзапросов в INSERT

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

4 Зак. 444

Например, пусть таблица STUDENTI имеет структуру, полностью совпадающую со структурой таблицы STUDENT. Запрос, позволяющий заполнить таблицу STUDENTI записями обо всех студентах из Москвы из таблицы STUDENT, выглядит следующим образом:

```
INSERT INTO STUDENTI
  •SELECT *
     FROM STUDENT
     WHERE CITY = 'Москва';
```

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

```
INSERT INTO STUDENTI
  SELECT *
     FROM STUDENT
     WHERE UNIV_ID IN
      (SELECT UNIV_ID
     FROM UNIVERSITY
     WHERE CITY = 'Москва');
```

3.2.1. 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 таблицы STUDENT, используя связанные подзапросы, следующим образом:

```
INSERT INTO SSTUD
SELKCT *
FROM STUDENT A
WHERE CITY IN
 (SELECT CITY
FROM UNIVERSITY B
WHERE A.UNIV ID = B.UNIV ID);
```

Предположим, 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, EXAMJ5ATE
FROM EXAM_MARKS A
WHERE MARK =
 (SELECT MAX(MARK;
FROM EXAM_MARKS B
WHERE A.EXAM_DATE = B.EXAM_DATE);
```

3.2.2. 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
FROMUNIVERSITY
WHERE CITY = 'Нью-Васюки');
```

В предикате предложения FROM (подзапроса) нельзя ссылаться на таблицу, из которой осуществляется удаление. Однако можно ссылаться на текущую строку из таблицы, являющуюся кандидатом на удаление, то есть на строку, которая в настоящее время проверяется в основном предикате.

```
DELETE
FROM STUDENT
WHERE EXISTS
 (SELECT *
FROM UNIVERSITY
```

1,1

```
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 университетах, имеющих рейтинг, равный 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 нужно найти наименьшее значение оценки, полученной в каждый день сдачи экзаменов, и удалить из таблицы сведения о студенте, 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.EXAMJDATE));
```

Так как столбец STUDENT_ID является первичным ключом, то удаляется единственная строка.

Если в какой-то день сдавался только один экзамен (то есть 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 COONT(SUBJ_ID)
FROM EXAM_MARKS B
WHERE A.EXAM_DATE = B.EXAM_DATE)));
```

3.2.3. Использование подзапросов с UPDATE

С помощью команды UPDATE можно применять подзапро-сы в любой форме, приемлемой для команды DELETE.

Например, используя связанные подзапросы, можно увеличить значение 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 студентам, 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 STUDENTI, определения столбцов которой полностью совпадают с определениями столбцов таблицы STUDENT. Вставить в эту таблицу сведения о студентах, успешно сдавших экзамены более чем по пяти предметам обучения.
2. Напишите команду, удаляющую из таблицы SUBJECTI сведения о предметах обучения, по которым студентами не получено ни одной оценки.
3. Напишите запрос, увеличивающий данные о величине стипендии на 20% всем студентам, у которых общая сумма баллов превышает значение 50.

4 Создание объектов базы данных

4.1. Создание таблиц **базы** данных

Создание объектов базы данных осуществляется с помощью операторов языка определения данных (DDL).

Таблицы базы данных создаются с помощью команды CREATE TABLE. Эта команда создает пустую таблицу, то есть таблицу, не имеющую строк. Значения в эту таблицу вводятся с помощью команды INSERT. Команда CREATE TABLE определяет имя таблицы и множество поименованных столбцов в указанном порядке. Для каждого столбца должен быть определен тип и размер. Каждая создаваемая таблица должна иметь, по крайней мере, один столбец. Синтаксис команды CREATE TABLE имеет следующий вид:

```
CREATE TABLE <имя таблицы>
(<имя столбцаХтип данных>[(<размер>)]);
```

Используемые в SQL типы данных как минимум поддерживают стандарты ANSI *(American National Standards Institute — Американский национальный институт стандартов)* (см. раздел 1.5 «Типы данных SQL»):

```
CHAR (CHARACTER),
INT (INTEGER),
SMALLINT,
DEC (DECIMAL),
NUMERIC,
FLOAT,
```

Тип данных, для которого обязательно должен быть указан размер, — это CHAR. Реальное количество символов, которое

может находиться в поле, изменяется от нуля (если в поле содержится NULL-значение) до заданного в CREATE TABLE максимального значения.

Следующий пример показывает команду, которая позволяет создать таблицу STUDENT.

```
CREATE TABLE STUDENT1
 (STUDENT__ID INTEGER,
 SURNAME     VARCHAR(60;,
NAME         VARCHAR (60),
 STIPEND     DOUBLE,
KURS         INTEGER,
CITY         VARCHAR(60>,
BIRTHDAY    DATE,
UNIV_ID      INTEGER);
```

4.2. Использование индексации для быстрого доступа к данным

Операции поиска-выборки (SELECT) данных из таблиц по значениям их полей могут быть существенно ускорены путем использования индексации данных. Индекс содержит упорядоченный (в алфавитном или числовом порядке) список содержимого столбцов или группы столбцов в индексируемой таблице с идентификаторами этих строк (ROWID), Для пользователей индексирование таблицы по тем или иным столбцам представляет собой способ *логического* упорядочения 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.

При использовании индексации следует, однако, иметь в виду, что управление индексом существенно замедляет время выполнения операций, связанных с обновлением данных (та-

ких, 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, данные упорядочиваются по значениям первого поля, по которому осуществляется индексирование. Внутри получившейся группы осуществляется упорядочение по значениям второго поля, для получившихся в результате групп осуществляется упорядочение по значениям третьего поля и т.д.

Синтаксис команды создания индекса имеет следующий вид:

```
CREATE INDEX <имя индекса> ON <имя таблицы> (<имя столбца>
                [,<имя столбца>]);
```

При этом таблица должна быть уже создана и содержать столбцы, 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 пользователя, 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 создать 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 синтаксис:

```
DROP INDEX <имя индекса>;
```

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

4.3. Изменение существующей таблицы

Для модификации структуры и параметров существующей таблицы используется команда ALTER TABLE. Синтаксис ко-

манды ALTER TABLE для добавления столбцов в таблицу имеет вид

```
ALTER TABLE <имя таблицы> ADD (<имя столбца> <тип данных>
        <размер>);
```

По этой команде 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 изменением размеров столбцов, добавлением или удалением ограничений, накладываемых на их значения. Синтаксис команды 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 в том случае, 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.

4.4. Удаление таблицы

Чтобы удалить существующую таблицу, необходимо предварительно удалить все данные из этой таблицы, то есть сделать ее пустой. Таблица, имеющая строки, не может быть удалена. Синтаксис команды, осуществляющей удаление пустой таблицы, имеет следующий вид:

```
DROP TABLE <имя таблицы>;
```

Упражнения

1. Напишите команду CREATE TABLE для создания таблицы LECTURER.
2. Напишите команду CREATE TABLE для создания таблицы SUBJECT.
3. Напишите команду CREATE TABLE для создания таблицы UNIVERSITY.
4. Напишите команду CREATE TABLE для создания таблицы EXAM_MARKS.
5. Напишите команду CREATE TABLE для создания таблицы SUBJ_LECT.
6. Напишите команду, которая позволит быстро выбрать данные о студентах по курсам, на которых они учатся.
7. Создайте индекс, который позволит для каждого студента быстро осуществить поиск оценок, сгруппированных по датам.

4.5. Ограничения на множество допустимых значений данных

До сих пор рассматривалось только следующее ограничение — значения, вводимые в таблицу, должны иметь типы данных и размеры, совместимые с типами/размером данных столбцов, в которые эти значения вводятся (как определено в команде CREATE TABLE или ALTER TABLE). Описание таблицы может быть дополнено более 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 ограничения на вводимые значения. В этом случае SQL будет отвергать любое из них при несоответствии заданным критериям. 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 данным *момент* содержатся также в другом столбце другой или этой же таблицы (внешний ключ). Эти ограничения носят динамический характер.

Существует два основных типа ограничений — ограничения на столбцы и ограничения на таблицу. Ограничения на столбцы (COLUMN CONSTRAINTS) применимы только к отдельным столбцам, а ограничения на таблицу (TABLE CONSTRAINTS) применимы к группам, состоящим из одного или более столбцов. Ограничения на столбец добавляются в конце определения столбца после указания типа данных и перед окончанием описания столбца (запятой). Ограничения на таблицу размещаются в конце определения таблицы, после определения 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, в которых они описаны.

4.5.1. Ограничение NOT NULL

Чтобы запретить возможность использования в поле NULL-значений, можно при создании таблицы командой CREATE TABLE указать для соответствующего столбца ключевое слово NOT NULL. Это ограничение применимо только к столбцам 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 содержать NULL-значений (быть пустыми), поскольку это нарушило бы требование уникальности первичного ключа (более строго — функциональную 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 STUDENT запретить использование NULL-значений для столбцов STUDENT_ID, SURNAME и NAME, можно записать следующее:

```
CREATE TABLE STUDENT
 (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: если для столбца указано NOT NULL, то при использовании команды INSERT обязательно должно быть указано конкретное значение, вводимое в это поле. При отсутствии ограничения 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 команду ALTER TABLE. Однако для того, чтобы для вновь вводимого с помощью команды ALTER TABLE столбца можно было задать ограничение NOT NULL, таблица, в которую добавляется столбец, должна быть пустой.

4.5.2. Уникальность как ограничение на столбец

Иногда требуется, 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 данных отвергает любую попытку ввести в это поле какой-либо строки значение, уже содержащееся в том же поле другой строки. Это ограничение применимо только к тем полям, которые были объявлены NOT NULL. Можно

предложить следующее определение таблицы STUDENT, использующее ограничение UNIQUE:

```
CREATE TABLE STUDENT
 (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 одинаковыми идентификаторами. Столбцы, отличные от первичного ключа, для которых требуется поддержать уникальность значений, называются возможными ключами или уникальными ключами (CANDIDATE KEYS ИЛИ UNIQUE KEYS).

4.5.3. Уникальность как ограничение таблицы

Можно сделать уникальными группу полей, указав UNIQUE в качестве ограничений *таблицы*. При объединении полей в группу важен порядок, в котором они указываются. Ограничение на таблицу UNIQUE является полезным, 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 следующим способом:

```
CREATE TABLE EXAM_MARKS
   (EXAM_ID      INTEGER NOT NULL,
   STUDENT_ID    INTEGER NOT NULL,
   SUBJ_ID       INTEGER NOT NULL,
   MARK          CHAR (1),
```

```
    EXAM_DATE    DATE NOT NULL,
    UNIQUE  (STUDENT_ID, EXAM_DATE));
```

Обратите внимание, 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 значение поля STUDENT_ID должно быть различным для каждой строки в таблице EXAM_MARKS, это можно сделать, объявив UNIQUE как ограничение самого поля STUDENT_ID. В этом случае не будет и двух строк с идентичной комбинацией значений полей STUDENT_ID, EXAM_DATE. Следовательно, указание UNIQUE как ограничение таблицы наиболее полезно использовать в случаях, когда не требуется уникальность индивидуальных полей, как это имеет место на самом деле в рассматриваемом примере.

4,5.4. 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 измененный синтаксис команд CREATE TABLE и ALTER TABLE.

Приведенный выше пример запроса изменяется следующим образом:

```
CREATE TABLE EXAM_MARKS
    (EXAM_ID      INTEGER NOT NULL,
    STUDENT_ID    INTEGER NOT NULL,
    SUBJ_ID       INTEGER NOT NULL,
    MARK          CHAR (1),
    EXAM_DATE     DATE NOT NULL,
    CONSTRAINT    STUD_SUBJ_CONSTR
    UNIQUE        (STUDENT ID, EXAM DATE);
```

В этом запросе STUD__SUBJ_CONSTR — это имя, присвоенное указанному ограничению таблицы.

4.5.5. Ограничение первичных ключей

Первичные ключи таблицы — 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 (строки, однако, могут идентифицироваться и другими способами).

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

```
CREATE TABLE STUDENT
 (STUDENT_ID INTKGER PRIMARY KEY,
    SURNAME CHAR (25) NOT NULL,
    NAME CHAR (10) NOT NULL,
    STIPEND INTEGER,
    KURS INTEGER,
    CITY CHAR (15),
    BIRTHDAY DATE,
    UNIV_ID INTEGER);
```

4.5.6. Составные первичные ключи

Ограничение PRIMARY KEY может также быть применено для нескольких полей, составляющих 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 требование 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 экзамене не может получить более одной оценки), имеет смысл объявить уникальной комбинацию этих полей. Для этого мы можем применить 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 ю, STUDENT  ID));
```

4.5.7. 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 такими, для которых предикат, задаваемый ограничением CHECK, имеет значение **ложь,** будет отвергаться.

Рассмотрим таблицу STUDENT. Значение столбца STIPEND в этой таблице выражается десятичным числом. Наложим на значения этого столбца 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);
```

4.5.8. Проверка ограничивающих условий с использованием составных полей

Ограничение CHECK можно использовать в качестве табличного ограничения, то есть при необходимости включить более одного поля 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'));
```

или в несколько другой 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'));
```

4.5.9. Установка значений по умолчанию

В SQL имеется возможность при вставке в таблицу строки, не указывая значений некоторого 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

OEPAOTлKзначение по умолчанию>.

Строго говоря, опция DEFAULT не имеет ограничительного свойства, так как она не ограничивает значения, вводимые в поле, а просто конкретизирует значение поля в случае, если оно *не было задано.*

Предположим, что основная масса студентов, информация о которых 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 умолчанию — 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 представляет собой признак того, что фактическое значение поля *неизвестно* или *неопределенно*. Следовательно, строго говоря, сравнение с ним любого конкретного значения в рамках двузначной булевой логики является некорректным, 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, подставляемого в поле по умолчанию, может существенно упростить использование 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 этом 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. Для такого применения задания по умолчанию необходимо (до вставки другой строки с установкой по умолчанию) модифицировать предыдущую строку, содержащую такое значение.

Упражнения

1. Создайте таблицу EXAMJVIARKS так, чтобы не допускался ввод в таблицу двух записей об оценках одного студента по конкретным экзамену и предмету обучения и чтобы не допускалось проведение двух экзаменов по любым предметам в один день,
2. Создайте таблицу предметов обучения SUBJECT так, чтобы количество отводимых на предмет часов по умолчанию было равно 36, не допускались записи с отсутствующим количеством часов, поле SUBJ_ID являлось первичным ключом таблицы и значения семестров (поле SEMESTER) 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 полей связаны друг с другом. Так, поле STUDENT_ID в таблице STUDENT и поле STUDENT_ID в таблице EXAM_MARKS связаны тем, что описывают одни и те же объекты, то есть содержат идентификаторы студентов, информация о которых хранится в базе. Более того, значения идентификаторов студентов, которые допустимы в таблице EXAM_MARKS, должны выбираться только из списка значений STUDENT_ID, фактически присутствующих в таблице STUDENT, то есть принадлежащих реально описанным в базе студентам. Аналогично, значения поля UNIV_ID таблицы STUDENT должны соответствовать идентификаторам университетов UNIV_ID, фактически присутствующим в таблице UNIVERSITY, а значения поля SUBJ_ID таблицы EXAM_MARKS должны соответствовать идентификаторам предметов обучения, фактически присутствующим в таблице SYBJECT.

Ограничения, накладываемые указанным типом связи, называются *ограничениями ссылочной целостности.* Они составляют важную часть описания характеристик предметной области, обеспечения корректности данных, хранящихся в таблицах. Команды описания таблиц DML имеют средства, позволяющие описывать ограничения ссылочной целостности и обеспечивать поддержание такой целостности при манипуляциях значениями полей базы данных.

4.6.1. 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,