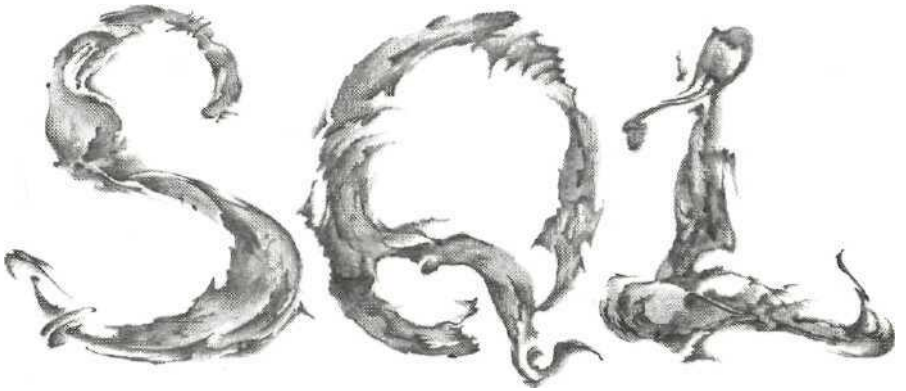
И.Ф. Астахова
А.П. Толстобров
В.М. Мельников



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.

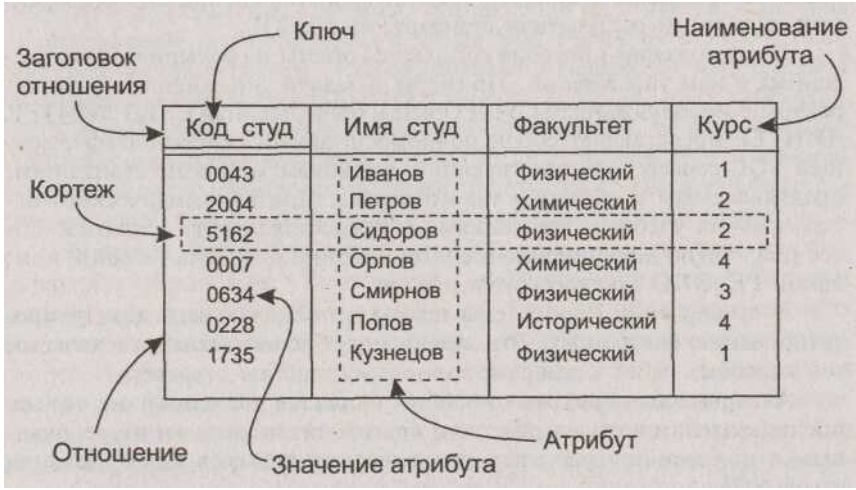


Рис. 1. Пример таблицы-отношения реляционной базы данных

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[,.масшта6])].

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 'О3-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, ИДСНТифиЦИруЮЩИЙ ПрСПОдавателя,

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,

```
SELECT NAME,SURNAME
    FROM STUDENT;
```

Любой SQL-запрос должен заканчиваться символом **«;»** ***(точка 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 'Р%';
```

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 '_\_Р' ESCAPE 'V.
```

В этом выражении символ 'V с помощью ключевого слова ESCAPE объявляется escape-символом. Первый символ «_» в заданном шаблоне поиска '__Р' 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'
'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

l. 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.