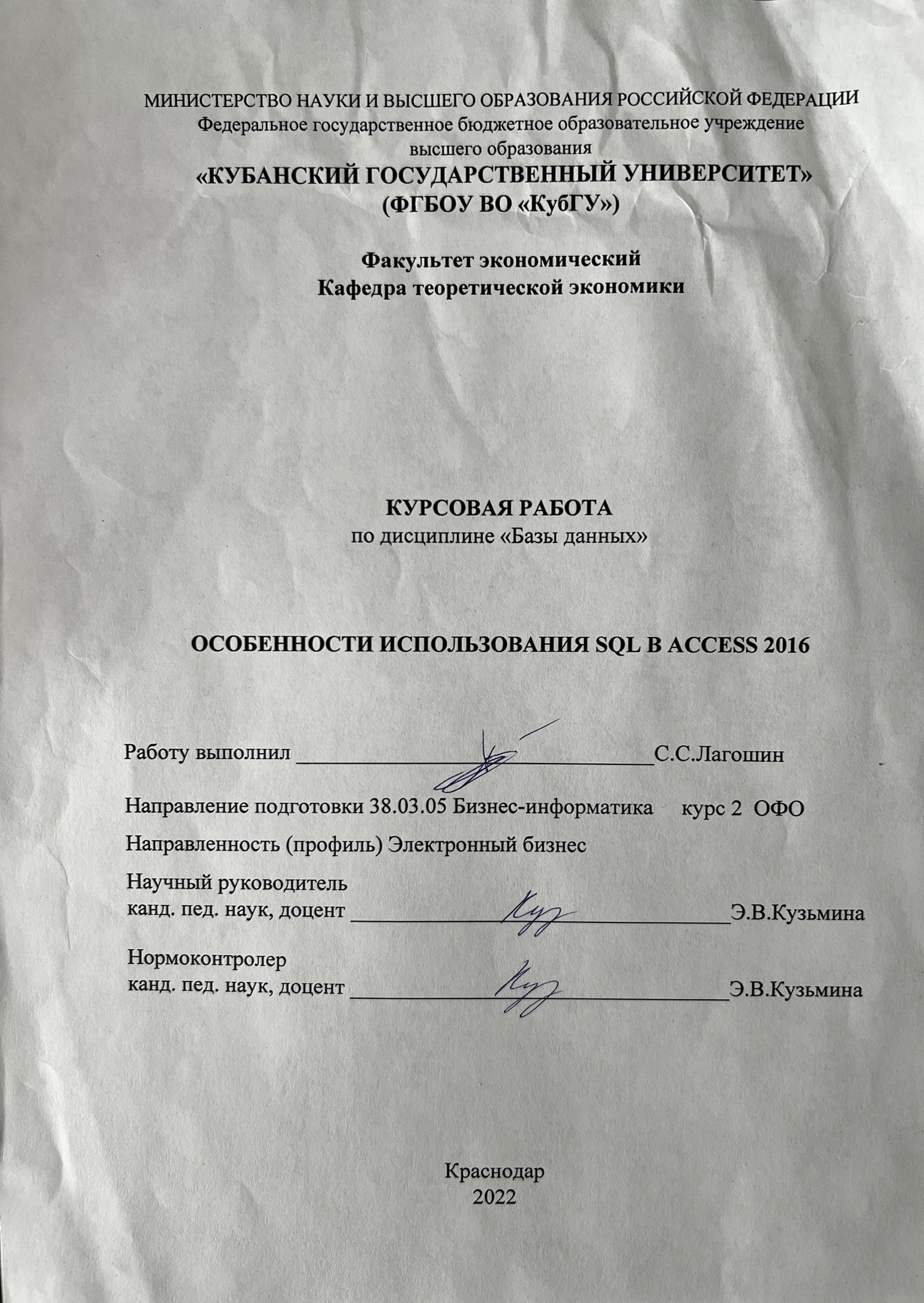
****

**Выборка данных (оператор select)**

**Простейшие SELECT-запросы**

**1.** 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>] ; 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 ниже запрос осуществляет выборку всех значений полей NAME и SURNAME из таблицы STUDENT.

Его результатом является таблица следующего вида:

|  |  |
| --- | --- |
| NAME | SURNAME |
| Иван | Иванов |
| Петр | Петров |
| 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, в не их порядку во входной таблице STUDENT.

**2.** 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:

|  |
| --- |
| CITY |
| Орел |
| Курск |
| Москва |
| Брянск |
| Липецк |
| **Воронеж** |
| Белгород |
| **Воронеж** |
| NULL |
| **Воронеж** |
| …………………… |

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

**3.** 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 |
| …………………… |

**4.** 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.

**5.** 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 |

**Операторы in, between, like, is null**

При задании логического условия в предложении where могут быть использованы операторы in, between, like, is null.

**1.** 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) считаются 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 NОТ 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);

**2.** Оператор between (между) 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.

**3.** 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 «\*» допускает присутствие в указанном месте проверяемой строки одного любого символа или последовательности любых символов произвольной длины.  
Например: SURNAME LIKE ‘П\*\*\*ов’ 🡺 Петров, Павлов

SUBJECT LIKE ‘О\*’ 🡺 Обществознание

**Пример**

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

SELECT \*

FROM STUDENT

WHERE SURNAME LIKE 'Р\*';

**4.** Обращаем внимание на то, что рассмотренные выше операторы сравнения «=, <, >, <=, >=, <>» и операторы IN, between и like ни в коем случае нельзя использовать для проверки 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)

**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.

**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. 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, KURS, (-(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 |
|  |  |  |  |  |

**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 |
|  |  |

**4. Функции преобразования символов в строке**

• LCase — перевод в строчные символы (нижний регистр)

lower (<строка>)

• UCase — перевод в прописные символы (верхний регистр)

upper (< строка >)

Например:

SELECT LCase (SURNAME), UCase (NAME)

FROM STUDENT

WHERE KURS = 4 AND STIPEND > 0;

Результат запроса будет выглядеть следующим образом:

|  |  |
| --- | --- |
| SURNAME | NAME |
| сидоров | ВАДИМ |
| петров | АНТОН |
|  |  |

**5. Строковые функции (работает только в SQL)**

• Left — Возврат знаков с начала строки

Left([СерийныйНомер],2)

Если [СерийныйНомер] = "CD234", результат — "CD"

• Right — Возврат знаков с конца строки

Right([СерийныйНомер],3)

Если [СерийныйНомер] = "CD234", результат — "234"

• Mid — Возврат знаков из середины строки

Mid([СерийныйНомер],2,2)

Если [СерийныйНомер] = "CD234", результат — "D2"

• trim — Удаление начальных и конечных пробелов из строки.

Trim([Имя])

Если [Имя] = " Алексей ", результатом будет: "Алексей".

• INSTR — Поиск позиции знака в строке

InStr(1,[Имя],"к")

Если [Имя] = "Алексей", результат — 4

• len — определение длины строки

Len([Имя])

Если [Имя] = "Алексей", результат — 7

**6. Функции работы с числами**

• **ABS** — модуль числа

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

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

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

• Экспоненциальная функция — **ехр** (<значимое числовое выражение>)

• Логарифмические функции — LN, **LOG** (<значимое числовое выражение>)

• Функция возведения в степень — **POWER**

**power** (<значимое числовое выражение>, <экспонента>)

• Вычисление квадратного корня — **SQRT** SQRT (<значимое числовое выражение>)

**7. Функции преобразования значений**

Format – позволяет преобразовать значение.

**FORMAT** (<значимое выражение>,[<символьный формат>])

• <значимое выражение> — числовое значение или значение типа дата-время;

• для значений типа дата-время <символьный формат> имеет вид (то есть вид отображения значений даты и вре­мени):

— в части даты

**‘DD-MMM-YY’ 🡪** 03-дек-82

**‘DD-MMM-YYYY’ 🡪** 03-дек-1982

**‘MM/DD/YY’ 🡪** 12.03.82

**‘MM/DD/YYYY’ 🡪** 12.03.1982

**‘DD.MM.YY’ 🡪** 03.12.82

**‘DD.MM.YYYY’ 🡪** 03.12.1982

Разницы между (.) и (/) нет.

**Пример**

SELECT SURNAME, NAME, FORMAT (BIRTHDAY, 'DD-MMM-YYYY'), FORMAT (BIRTHDAY, 'DD.MM.YY')

FROM STUDENT;

**Вернет результат:**

|  |  |  |  |
| --- | --- | --- | --- |
| SURNAME | NAME |  |  |
| Иванов | Иван | З-дек-1982 | 3.12.82 |
| Петров | Петр | 1-дек-1980 | 1.12.80 |
| Сидоров | Вадим | 7-июн-1979 | 7.06.79 |
| Кузнецов | Борис | 8-дек-1981 | 8.12.81 |
| Зайцева | Ольга | 1-май-1981 | 1.05.81 |
| Павлов | Андрей | 5-ноя-1979 | 5.11.79 |
| Котов | Павел | **NULL** | **NULL** |
| Лукин | Артем | 1-дек-1981 | 1.12.81 |
| Петров | Антон | 5-авг-1981 | 5.08.81 |
| Белкин | Вадим | 7-янв-1980 | 7.01.80 |
|  |  |  |  |

**Агрегирование и групповые функции**

**1.** 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;

• МАХ 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.

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

Например, для определения среднего значения поля mark (оценки) по всем записям таблицы exam marks можно использовать запрос с функцией AVG следующего вида:

SELECT AVG (MARK)

FROM EXAM\_MARKS;

**3.** 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 вес равно не войдут в число подсчитываемых значений.

SELECT DISTINCT (SUBJ\_NAME), COUNT (SUBJ\_NAME) AS [КОЛИЧЕСТВО ПРЕДМЕТОВ]

FROM SUBJECT

GROUP BY (SUBJ\_NAME);

**4.** Предложение group by (группировать по) позволяет группировать записи в подмножества, определяемые значениями какого-либо поля.

Предположим, требуется найти максимальное значение оценки, полученной каждым студентом. Запрос будет выглядеть следующим образом:

SELECT STUDENT\_ID, MAX(MARK) AS [МАКСИМАЛЬНАЯ ОЦЕНКА]

FROM EXAM\_MARKS

GROUP BY STUDENT\_ID;

Выбираемые из таблицы exam\_marks записи группируются по значениям поля student\_id, указанного в предложении group by, и для каждой группы находится максимальное значение поля mark. 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(MARK) AS [МАКСИМАЛЬНАЯ ОЦЕНКА]

FROM EXAM\_MARKS

GROUP BY STUDENT\_ID, SUBJ\_ID;

В ЭТОМ случае строки вначале группируются по значениям 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.

**5.** При необходимости часть сформированных с помощью group by групп может быть исключена с помощью предложения HAVING.

Предложение HAVING определяет критерий, по которому группы следует включать в выходные данные, по аналогии с предложением where, которое осуществляет это для отдельных строк.

SELECT SUBJ\_NAME, MAX(HOUR)

FROM SUBJECT

GROUP BY SUBJ\_NAME

HAVING MAX (HOUR >= 56);

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

**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 цементы 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 столбце присутствуют пустые значения, то может быть возвращено и третье значение: unknown (неизвестно). В этой схеме, например, условие where А = 2, где А — имя столбца, значения которого могут быть неизвестны, при А = 2 будет соответствовать TRUE, при А = 4 в результате будет получено значение FALSE, а при отсутствующем значении А (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, тo 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, to 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 SUBJNAME 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, ввиду, 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.

**Вложенные подзапросы**

**1.** SQL позволяет использовать одни запросы внутри других запросов, то есть вкладывать запросы друг в друга. Предположим, известна фамилия студента («Зайцева»), 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

WHERE SURNAME = 'Зайцева');

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.

**2.** 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');

**3.** Подзапросы можно применять внутри предложения HAVING. Пусть требуется определить количество предметов обучения с оценкой, превышающей среднее значение оценки студента  
с идентификатором 12:

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

FROM EXAM\_MARKS

GROUP BY SUBJ\_ID, MARK

HAVING (MARK) >

(SELECT AVG(MARK)

FROM EXAM\_MARKS

WHERE STUDENT\_ID =12);

**Формирование связанных подзапросов**

**4.** 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 23 марта 2000 г.

SELECT \*

FROM SUBJECT

WHERE SUBJ\_ID IN

(SELECT SUBJ\_ID

FROM EXAM\_MARKS

WHERE EXAM\_DATE = #23/03/2000#);

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 El

WHERE STIPEND >

(SELECT AVG(STIPEND)

FROM STUDENT E2

WHERE El.KURS - E2.KURS);

**Связанные подзапросы в having**

**5.** 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

GROUP BY EXAM\_DATE

HAVING COUNT (EXAM\_DATE) > 10

**Использование оператора exists**

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

Пусть, например, нужно извлечь из таблицы exam\_marks данные о студентах, получивших хотя бы одну неудовлетворительную оценку.

SELECT \*

FROM STUDENT

WHERE EXISTS

(SELECT \*

FROM EXAM\_MARKS

WHERE STUDENT.STUDENT\_ID=EXAM\_MARKS.STUDENT\_ID AND MARK<3);

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 предмета.

SELECT \*

FROM STUDENT

WHERE EXISTS

(SELECT SUBJ\_ID

FROM EXAM\_MARKS

WHERE STUDENT.STUDENT\_ID = EXAM\_MARKS.STUDENT\_ID AND COUNT(SUBJ\_ID>1));

**Операторы сравнения с множеством значений in, any, аLL**

**1.** 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, >= 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

WHERE CITY = ANY

(SELECT CITY

FROM UNIVERSITY

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

Другой вариант этого запроса:

SELECT \*

FROM STUDENT

WHERE CITY IN

(SELECT CITY

FROM UNIVERSITY

WHERE UNIVERSITY.UNIV\_ID = STUDENT.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 STUDENT\_ID, MARK

FROM EXAM\_MARKS

WHERE MARK > ANY

(SELECT MARK

FROM EXAM\_MARKS

WHERE EXAM\_DATE = #17/06/1999#);

**2.** 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 RATING > ANY

(SELECT RATING

FROM UNIVERSITY

WHERE CITY = 'Воронеж');

**Особенности применения операторов any, all, exists при обработке пустых значений (null)**

**3.** 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

SELECT \*

FROM UNIVERSITY

WHERE RATING > ALL

(SELECT RATING

FROM UNIVERSITY

WHERE CITY = 'New York');

полностью воспроизведет таблицу university.

**4.** 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 UNKNOW, 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

FROM 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.

**Использование count вместо exists**

**5.** При отсутствии NULL-значений оператор exists может быть использован вместо ANY или 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');

* 1. **Команды манипулирования данными**

ВSQL для выполнения операций ввода данных в таблицу, их изменения и удаления предназначены три команды языка манипулирования данными (DML). Это команды

* INSERT (вставить),
* DELETE (удалить),
* UPDATE (обновить).

**1.** Команда INSERT осуществляет **вставку** в таблицу новой

строки. Впростейшем 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, вкотором соответствующие столбцы указаны при создании таблицы, то есть воператоре CREATE TABLE.

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

INSERT INTO STUDENT

VALUES (101, 'Иванов', 'Александр', 200, 3, 'Москва', #6/10/1979#, 15);

Чтобы 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 STUDENT1 должна быть предварительно создана командой CREATE TABLE и иметь структуру, идентичную таблице STUDENT.

**2. Удаление**строк из таблицы осуществляется с помощью команды DELETE.

Следующее выражение удаляет все строки таблицы EXAM\_MARKS1.

DELETE \*

FROM EXAM\_MARKS1;

Врезультате 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 STUDENT1

WHERE CITY = 'Москва';

**3.** Команда UPDATE позволяет изменять, то есть обновлять значения некоторых или всех полей в существующей строке или строках таблицы. Например, чтобы для всех университетов, сведения о которых находятся в таблице UNIVERSITY1, изменить рейтинг на значение 200, можно использовать конструкцию:

UPDATE UNIVERSITY1

SET RATING = 200;

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

UPDATE UNIVERSITY1 SET RATING = 200

WHERE CITY = 'Москва';

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

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 UNIVERSITY1

SET RATING = RATING\*2;

Например, для увеличения в таблице STUDENT1 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';

**Создание таблиц базы данных**

**1.** 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 столбца Х тип данных>[(<размер>)]);

Используемые в SQL типы данных как минимум поддерживают стандарты ANSI (American National Standards Institute — Американский национальный институт стандартов)

**2. «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);

**3. 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 (таких, как INSERT и DELETE), так как эти операции требуют перестройки индексов.

Индексы можно создавать как по одному, так и по множеству полей. Если указано более одного поля для создания единственного индекса, 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.

**3.1** 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);

**3.2** 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. Изменение существующей таблицы**

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

**5. 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>;