# Тема №3 - Выборка данных (оператор SELECT).

Оператор SELECT (выбрать) языка SQL является самым важным и самым часто используемым оператором. Он предназначен для выборкиинформации из таблиц базы данных. 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>];

Вквадратных 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. За ключевым словом FROM указывается список разделенных запятыми имен таблиц, из которых извлекается информация.

**Например:**

SELECT

"ARTICUL", "NAME", "PRICE"

FROM

product;

Приведенный запрос осуществляет выборку всех значений «ARTICUL», «NAME», «PRICE» из таблицы product.

Если необходимо вывести значения всех,столбцов таблицы, то можно вместо перечисления их имен использовать символ **«\*»** (звездочка).

**Например:**

SELECT \*

FROM

product;

В данном случае результатом выполнения запроса будет вся таблица product. Следует заметить, что результат данного запроса может содержать строки с одинаковыми значениями атрибутов. Для исключения из результата SELECT - запроса повторяющихся записей используется ключевое слово DISTINCT (отличный). Если запрос SELECT извлекает множество полей, то DISTINCT исключаетдубликаты строк, в которых значения всехвыбранных полей идентичны.

**Например:**

SELECT DISTINCT "NAME"

FROM product;

Ключевое слово ALL (все), в отличие от DISTINCT, оказывает противоположное действие, то есть при его использовании повторяющиеся строки включаютсяв выходные данные. Режим, задаваемый ключевым словом ALL, действует по умолчанию, поэтому в реальных запросах для этих целей оно практически не используется.

**Например:**

Напишите данный запрос выбирает все данные из таблицы PRODUCT, расположив столбцы таблицы в следующем порядке: ARTICUL, NAME, IMAGE, PRICE.

SELECT ALL

"ARTICUL", "NAME", "IMAGE", "PRICE"

FROM

product;

Использование в операторе SELECT предложения, определяемого ключевым словом WHERE (где), позволяет задавать какие строкиуказанных таблиц должны быть выбраны*.* В таблицу, являющуюся результатом запроса, включаются только те строки, для которых условие, указанное в предложении WHERE, принимает значение истина.

**Например**: запрос выводит запрашиваемые поля определенного товара, артикул которого равен «12743303» из таблицы product базы данных рассмотренной в теме №1.

SELECT ALL

"ARTICUL", "NAME", "IMAGE", "PRICE"

FROM

product

WHERE

"ARTICUL" = '12743274'

Результатом запроса будет следующая информация:

|  |  |  |  |
| --- | --- | --- | --- |
| ARTICUL | NAME | IMAGE | PRICE |
| 12743274 | Наволочка | [null] | $490.00 |

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:**

SELECT ALL

"ARTICUL", "NAME", "IMAGE", "PRICE"

FROM

product

WHERE

"NAME" = 'Наволочка'

В результате мы получим все товары с наименованием “Наволочка”

Также рассмотрим пример с использованием логических операторов

**Например:**

SELECT ALL

"ARTICUL", "NAME", "IMAGE", "PRICE"

FROM

product

WHERE

"NAME" = 'Наволочка' AND "PRICE" < '500$'

В данном случае выведутся все товары с наименованием “Наволочка” и стоимостью меньше 500$.

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. Построенное с использованием IN условие считается истинным, если значение поля, имя которого указано слева от IN, в точности совпадает с одним из значений, перечисленных в списке, указанном в скобках справа от IN. Условие, построенное с использованием NOT IN, считается истинным, если значение поля, имя которого указано слева от NOT IN, не совпадаетни с одним из значений, перечисленных в списке, указанном в скобках справа от NOT IN.

**Например:**

Получим из таблицы товары с стоимостью 130$ и 140$

SELECT

"ARTICUL", "NAME", "PRICE"

FROM

product

WHERE

"PRICE" IN ('130$','140$')

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**:

Получим товары стоимость который находится в пределе от 100$ до 150$

SELECT

"ARTICUL", "NAME", "PRICE"

FROM

product

WHERE

"PRICE" BETWEEN '100$' AND '150$'

Оператор BETWEEN может использоваться как для числовых, так и для символьных типов полей.

Оператор LIKE применим только к символьным полям типа CHAR или VARCHAR. Этот оператор просматривает строковые значения полей с целью определения, входит ли заданная в операторе LIKE подстрока 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.

**Например:**

Осуществим выборку информации из таблицы product всех товаров, фамилии которых начинаются на букву «П»

SELECT \*

FROM

product

WHERE

"NAME" LIKE 'П%'

Следует отметить, что операторы сравнения «=, <, >, <=, >=, <>» и операторы IN, BETWEEN и LIKE нельзя использовать для проверки содержимого поля на наличие в нем пустого значения NULL. Для этих целей предназначены специальные операторы is NULL (является пустым) и IS NOT NULL (является не пустым).

**Например:**

Следующий запрос предназначен для выбора из таблицы product записей, в которых отсутствуют значения цен.

select \*

from product

where "PRICE" is null;

**Например:**

Следующий запрос предназначен для вывода из таблицы product записей, имеющих в поле «PRICE» значения оценок.

select \*

from product

where "PRICE" is not null;

# Использование агрегирующих функций.

Агрегирующие функции позволяют получать из таблицы сводную 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.

Аргументами агрегатных функций могут быть как столбцы таблицы, так и результаты выражений над ними. При этом выражение может быть сколько угодно сложным.

**Функция AVG ().**

Данные, содержащиеся в таблице должны иметь числовой тип, потому что функция AVG вначале суммирует все значения в столбце, а затем делит полученную сумму на количество этих значений.

**Функция SUM().**

Для вычисления суммы значений, содержащихся в столбце, используется функция SUM. При этом столбец должен иметь числовой тип данных, потому что результат, возвращаемый этой функцией, имеет тот же тип данных, что и столбец.

**Функции MAX и MIN.**

Для нахождения максимального и минимального значения в столбце, используются агрегатные функции MAX и MIN. При этом столбец может содержать числовые и строковые значения, либо значения даты/времени.

**Функция** **COUNT** возвращает количество записей в запросе.  
Синтаксис функции **COUNT**:

SELECT COUNT(expression) FROM table WHERE predicates;

**Функции MAXVALUE и MINVALUE** в чем-то похожи на агрегатные функции MAX и MIN, однако выбирают максимальное и минимальное значение не из множества значений строк в одном столбце, а из значений, заданных в списке аргументов.

**Функция CASE**

Выражение CASE позволяет возвращать разные значения в зависимости от условия.

**Функция COALESCE**

Возвращает первое выражение из списка аргументов, не равное NULL.

**Функция NULLIF**

Данная функция позволяет при выборке делать замену заданного значения на NULL.

Приведем примеры использования агрегированных функций.

SELECT COUNT (\*)

FROM product

Данный запрос выведет количество элементов в данной таблице.

SELECT COUNT (DISTINCT "NAME")

FROM product

В данном случае происходит подсчет количества уникальных имен в таблице.

SELECT MIN("PRICE")

FROM product

Данный запрос находит наименьшую стоимость товара.

SELECT AVG(CAST ("PRICE" AS NUMERIC))

FROM product

Данный запрос находит среднюю стоимость товара, перед этим приводя данные к числовому типу, т.к. функция AVG в psql не определена для денежных типов

Рассмотрим ряд примеров с использованием пустых значений (NULL) в агрегирующих функциях и упорядочивания выходных полей с использованием конструкции ORDER BY.

Предположим что в магазине проходит акция, и стоимость всех товаров по акции снижена на 20%. Мы хотим вывести данные о товарах, упорядочив их по тем или иным параметрам. Выходные данные упорядочить: а)по стоимости товара; б) в алфавитном порядке названий товаров.

а)

SELECT "NAME", "PRICE", "PRICE" \* 0.8

FROM product

ORDER BY "NAME";

б)

SELECT "NAME", "PRICE", "PRICE" \* 0.8

FROM product

ORDER BY "PRICE"

# Использование числовых функций

Числовые функции возвращают числовые значения на основании значений типа заданного в аргументе. Числовые функции используются для обработки данных, а также в условиях поиска. Основные типы числовых функций, а также основные математические операторы приведены в таблице

Таблица №2 – Математические операторы

|  |  |  |  |
| --- | --- | --- | --- |
| **Математические операторы** | | | |
| **Оператор** | **Описание** | **Пример** | **Результат** |
| + | сложение | 2 + 3 | 5 |
| - | вычитание | 2 – 3 | -1 |
| \* | умножение | 2 \* 3 | 6 |
| / | деление (при целочисленном делении остаток отбрасывается) | 4 / 2 | 2 |
| % | остаток от деления | 5 % 4 | 1 |
| ^ | возведение в степень (вычисляется слева направо) | 2.0 ^ 3.0 | 8 |
| |/ | квадратный корень | |/ 25.0 | 5 |
| ||/ | кубический корень | ||/ 27.0 | 3 |
| ! | факториал | 5 ! | 120 |
| !! | факториал (префиксная форма) | !! 5 | 120 |
| @ | модуль числа (абсолютное значение) | @ -5.0 | 5 |
| & | битовый AND | 91 & 15 | 11 |
| | | битовый OR | 32 | 3 | 35 |
| # | битовый XOR | 17 # 5 | 20 |
| ~ | битовый NOT | ~1 | -2 |
| << | битовый сдвиг влево | 1 << 4 | 16 |
| >> | битовый сдвиг вправо | 8 >> 2 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Математические функции** | | | |
| **Функция** | **Описание** | **Пример** | **Результат** |
| abs(*x*) | модуль числа (абсолютное значение) | abs(-17.4) | 17.4 |
| cbrt(dp) | кубический корень | cbrt(27.0) | 3 |
| ceil(dp или  numeric) | ближайшее целое, большее или равное аргументу | ceil(-42.8) | -42 |
| ceiling(dp или   numeric) | ближайшее целое, большее или равное аргументу (равнозначно ceil) | ceiling(-95.3) | -95 |
| degrees(dp) | преобразование радианов в градусы | degrees(0.5) | 28.6478897565​41 |
| div(*y* numeric,  *x* numeric) | целочисленный результат *y*/*x* | div(9,4) | 2 |
| exp(dp или numeric) | экспонента | exp(1.0) | 2.7182818284​590 |
| floor(dp или numeric) | ближайшее целое, меньшее или равное аргументу | floor(-42.8) | -43 |
| ln(dp или numeric) | натуральный логарифм | ln(2.0) | 0.6931471805​599 |
| log(dp или numeric) | логарифм по основанию 10 | log(100.0) | 2 |
| log(*b* numeric,  *x* numeric) | логарифм по основанию *b* | log(2.0, 64.0) | 6.0000000000 |
| mod(*y*, *x*) | остаток от деления *y*/*x* | mod(9,4) | 1 |
| pi() | константа «π» | pi() | 3.1415926535​897 |
| power(*a* dp, *b* dp) | *a* возводится в степень *b* | power(9.0, 3.0) | 729 |
| power(*a* numeric, *b* numeric) | *a* возводится в степень *b* | power(9.0, 3.0) | 729 |
| radians(dp) | преобразование градусов в радианы | radians(45.0) | 0.7853981633​974 |
| round(dp или numeric) | округление до ближайшего целого | round(42.4) | 42 |
| round(*v* numeric, *s* int) | округление *v* до *s* десятичных знаков | round(42.4382, 2) | 42.44 |
| scale(numeric) | масштаб аргумента (число десятичных цифр в дробной части) | scale(8.41) | 2 |
| sign(dp или numeric) | знак аргумента (-1, 0, +1) | sign(-8.4) | -1 |
| sqrt(dp или numeric) | квадратный корень | sqrt(2.0) | 1.4142135623​731 |
| trunc(dp или numeric) | округление к нулю | trunc(42.8) | 42 |
| trunc(*v* numeric, *s* int) | округление к 0 до *s* десятичных знаков | trunc(42.4382, 2) | 42.43 |

| **Случайные функции** | | |
| --- | --- | --- |
| **Функция** | **Тип результата** | **Описание** |
| random() | dp | случайное число в диапазоне 0.0 <= x < 1.0 |
| setseed(dp) | void | задаёт отправную точку для последующих вызовов random() (значение между -1.0 и 1.0, включая границы) |

| **Функции (в радианах)** | **Функции (в градусах)** | **Описание** |
| --- | --- | --- |
| acos(*x*) | acosd(*x*) | арккосинус |
| asin(*x*) | asind(*x*) | арксинус |
| atan(*x*) | atand(*x*) | арктангенс |
| atan2(*y*, *x*) | atan2d(*y*, *x*) | арктангенс *y*/*x* |
| cos(*x*) | cosd(*x*) | косинус |
| cot(*x*) | cotd(*x*) | котангенс |
| sin(*x*) | sind(*x*) | синус |
| tan(*x*) | tand(*x*) | тангенс |

**Примечание**

Также можно работать с углами в градусах, применяя вышеупомянутые функции преобразования единиц radians() и degrees(). Однако предпочтительнее использовать тригонометрические функции с градусами, так как это позволяет избежать ошибок округления в особых случаях, например, при вычислении sind(30).

**Строковые функции.**

Строковые функции используют в качестве аргумента строку символов, а в качестве результата возвращают символьную строку или числовое значение.

|  |  |  |  |
| --- | --- | --- | --- |
| **Строковые функции** | | | |
| **Функция** | **Описание** | **Пример** | **Результат** |
| *string* || *string* | Конкатенация строк | 'Post' || 'greSQL' | PostgreSQL |
| *string* || *не string* или *не string* || *string* | Конкатенация строк с одним не строковым операндом | 'Value: ' || 42 | Value: 42 |
| bit\_length(*string*) | Число бит в строке | bit\_length('jose') | 32 |
| char\_length(*string*) или  character\_length(*string*) | Число символов в строке | char\_length('jose') | 4 |
| lower(*string*) | Переводит символы строки в нижний регистр | lower('TOM') | tom |
| octet\_length(*string*) | Число байт в строке | octet\_length  ('jose') | 4 |
| ovelay(*string* placing  *string* from int [for int]) | Заменяет подстроку | ovelay  ('Txxxxas' placing 'hom' from 2 for 4) | Thomas |
| position(*substring*  in *string*) | Положение указанной подстроки | position  ('om' in 'Thomas') | 3 |
| sustring(*string* [from int] [for int]) | Извлекает подстроку | substring  ('Thomas' from 2 for 3) | hom |
| substring(*string*  from *шаблон*) | Извлекает подстроку, соответствующую регулярному выражению в стиле POSIX. | substring  ('Thomas' from '...$') | mas |
| substring  (*string* from *шаблон* for *спецсимвол*) | Извлекает подстроку, соответствующую регулярному выражению в стиле SQL. | substring  ('Thomas' from '%#"o\_a#"\_' for '#') | oma |
| trim([leading | trailing | both] [*characters*] from *string*) | Удаляет наибольшую подстроку, содержащую только символы *characters* (по умолчанию пробелы), с начала (leading), с конца (trailing) или с обеих сторон (both, (по умолчанию)) строки *string* | trim(both 'xyz' from 'yxTomxx') | Tom |
| trim([leading | trailing | both] [from] *string* [, *characters*] ) | Нестандартный синтаксис trim() | trim(both from 'yxTomxx', 'xyz') | Tom |
| upper(*string*) | Переводит символы строки в верхний регистр | upper('tom') | TOM |

|  |  |  |  |
| --- | --- | --- | --- |
| Другие функции работы со строками | | | |
| Функция | Описание | Пример | Результат |
| ascii(*string*) | Возвращает ASCII-код первого символа аргумента. Для UTF8 возвращает код символа в Unicode. Для других многобайтных кодировок аргумент должен быть ASCII-символом. | ascii('x') | 120 |
| btrim(*string* text  [, *characters* text]) | Удаляет наибольшую подстроку, состоящую только из символов *characters* (по умолчанию пробелов), с начала и с конца строки *string* | btrim('xyxtrimyyx', 'xyz') | trim |
| chr(int) | Возвращает символ с данным кодом. Для UTF8 аргумент воспринимается как код символа Unicode, а для других кодировок он должен указывать на ASCII-символ. Код 0 (NULL) не допускается, так как байты с нулевым кодом в текстовых строках сохранить нельзя. | chr(65) | A |
| concat  (*str* "any" [, *str* "any" [, ...] ]) | Соединяет текстовые представления всех аргументов, игнорируя NULL. | concat('abcde', 2, NULL, 22) | abcde222 |
| concat\_ws  (*sep* text, *str* "any"  [, *str* "any" [, ...] ]) | Соединяет все аргументы, кроме первого, через разделитель, игнорируя аргументы NULL. Разделитель указывается в первом аргументе. | concat\_ws(',', 'abcde', 2, NULL, 22) | abcde,2,22 |
| convert(*string* bytea,  *src\_encoding* name,  *dest\_encoding* name) | Преобразует строку string из кодировки src\_encoding в dest\_encoding. Переданная строка должна быть допустимой для исходной кодировки. | convert(  'text\_in\_utf8', 'UTF8', 'LATIN1') | строка text\_in\_utf8, представленная в кодировке Latin-1 (ISO 8859-1) |
| convert\_from  (*string* bytea,  *src\_encoding* name) | Преобразует строку *string* из кодировки *src\_encoding* в кодировку базы данных. Переданная строка должна быть допустимой для исходной кодировки. | convert\_from  ('text\_in\_utf8', 'UTF8') | строка text\_in\_utf8, представленная в кодировке текущей базы данных |
| convert\_to(*string* text,  *dest\_encoding* name) | Преобразует строку в кодировку *dest\_encoding*. | convert\_to  ('некоторый текст', 'UTF8') | некоторый текст, представленный в кодировке UTF8 |
| initcap(*string*) | Переводит первую букву каждого слова в строке в верхний регистр, а остальные — в нижний. Словами считаются последовательности алфавитно-цифровых символов, разделённые любыми другими символами. | initcap('hi THOMAS') | Hi Thomas |
| left(*str* text, *n* int) | Возвращает первые *n* символов в строке. Когда *n* меньше нуля, возвращаются все символы слева, кроме последних |*n*|. | left('abcde', 2) | ab |
| length(*string*) | Число символов в строке *string* | length('jose') | 4 |
| lpad(*string* text,  *length* int [, *fill* text]) | Дополняет строку *string* слева до длины *length* символами *fill* (по умолчанию пробелами). Если длина строки уже больше заданной, она обрезается справа. | lpad('hi', 5, 'xy') | xyxhi |
| ltrim(*string* text [,  *characters* text]) | Удаляет наибольшую подстроку, содержащую только символы *characters* (по умолчанию пробелы), с начала строки *string* | ltrim('zzzytest', 'xyz') | test |
| md5(*string*) | Вычисляет MD5-хеш строки *string* и возвращает результат в 16-ричном виде | md5('abc') | 90015098​3cd24fb0 d6963f7d​28e17f72 |
| pg\_client\_encoding() | Возвращает имя текущей клиентской кодировки | pg\_client\_encoding() | SQL\_ASCII |
| quote\_literal  (*value* anyelement) | Переводит данное значение в текстовый вид и заключает в апострофы как текстовую строку. Символы апостроф и обратная косая черта при этом дублируются. | quote\_literal  (42.5) | '42.5' |
| repeat(*string* text,  *number* int) | Повторяет содержимое *string* указанное число (*number*) раз | repeat('Pg', 4) | PgPgPgPg |
| replace(*string* text,  *from* text, *to* text) | Заменяет все вхождения в *string* подстроки *from* подстрокой *to* | replace('abcdefabcdef',  'cd', 'XX') | abXXefabXXef |
| reverse(*str*) | Возвращает перевёрнутую строку | reverse('abcde') | edcba |
| right(*str* text, *n* int) | Возвращает последние *n* символов в строке. Когда *n* меньше нуля, возвращаются все символы справа, кроме первых |*n*|. | right('abcde', 2) | de |
| rpad(*string* text,  *length* int [, *fill* text]) | Дополняет строку *string* справа до длины *length* символами *fill* (по умолчанию пробелами). Если длина строки уже больше заданной, она обрезается. | rpad('hi', 5, 'xy') | hixyx |
| rtrim(*string* text [,  *characters* text]) | Удаляет наибольшую подстроку, содержащую только символы *characters* (по умолчанию пробелы), с конца строки *string* | rtrim  ('testxxzx', 'xyz') | test |
| split\_part(*string* text,  *delimiter* text,  *field* int) | Разделяет строку *string* по символу *delimiter* и возвращает элемент по заданному номеру (считая с 1) | split\_part  ('abc~@~def~@~ghi', '~@~', 2) | def |
| strpos(*string*,  *substring*) | Возвращает положение указанной подстроки (подобно position(*substring* in *string*), но с другим порядком аргументов) | strpos('high', 'ig') | 2 |
| substr(*string*, *from* [,  *count*]) | Извлекает подстроку (подобно substring(*string* from *from* for *count*)) | substr('alphabet', 3, 2) | ph |

**Предложение GROUP BY**

Запрос, включающий в себя предложение GROUP BY, называется запросом с группировкой, поскольку он объединяет строки исходной таблицы в группы и для каждой группы строк генерирует одну строку.

Элементы, указанные в предложении GROUP BY, называются элементами группировки*,* и именно они определяют, по какому признаку строки делятся нагруппы. При этом группой называется набор строк, имеющих одинаковоезначение в элементе (элементах) группировки.

Синтаксис предложения GROUP BY имеет следующий вид:

GROUP BY <элемент\_группировки1> [, <элемент\_группировки2>] …,

Фактически в качестве элемента группировки может выступать любой возвращаемый элемент, указанный в предложении SELECT, кроме значений агрегатных функций. В выражение, представляющее собой <элемент\_группировки>, могут входить скалярные функции, агрегатные функции из различных контекстов или это может быть любая CASE-операция. Использование предложения GROUP BY имеет смысл только при наличии в списке возвращаемых элементов предложения SELECT хотя бы одного вычисляемого столбца или агрегатной функции. Агрегатная функция берет столбец значений и возвращает одно значение. Предложение GROUP BY указывает, что результаты запроса следует разделить на группы, применить агрегатную функцию по отдельности к каждой группе и получить для каждой группы одну строку результатов. Например, если необходимо вычислить среднее значение цены за единицу товара из таблицы Товар, то можно воспользоваться запросом, приведенном на рисунке 19.



Результаты запроса:

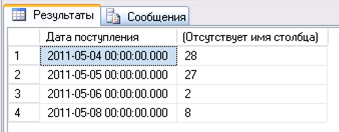
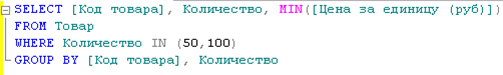


Рисунок 19 – Запрос и результаты его выполнения

Возможна группировка результатов запроса на основании порядкового номера возвращаемого элемента в предложении SELECT. Таким образом, запрос **SELECT [Дата поступления], AVG([Цена за единицу (руб)]) FROM Товар GROUP BY 1** выдаст такой же результат, как и предыдущий запрос.

В SQL можно группировать результаты запроса на основании двух или более элементов. Например, необходимо для каждого товара (по коду товара) необходимо вывести минимальную цену за единицу продукции. Для этого нужно сгруппировать таблицу Товар по коду товара и количеству товара в наличии, применяя к значениям цен агрегатную функцию MIN (рисунок 20).



Результаты запроса:

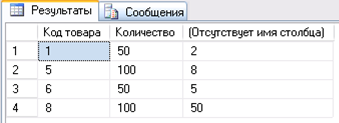


Рисунок 20 – Запрос с функцией MIN и результаты его выполнения

Как следует из предыдущих примеров, на запросы с группировкой накладываются следующие ограничения.

1. В предложении GROUP BY должны быть указаны столбцы или выражения, которые используются в качестве возвращаемых элементов предложения SELECT (за исключением агрегатных функций).

2. Все элементы списка возвращаемых столбцов должны иметь одно значение для каждой группы строк. Это означает, что возвращаемым элементом в предложении SELECT может быть:

* константа;
* агрегатная функция, возвращающая одно значение для всех строк, входящих в группу;
* элемент группировки, который по определению имеет одно и то же значение во всех строках группы;
* функция, которая используется в качестве элемента группировки;
* выражение, включающее в себя перечисленные выше элементы.

**Предложение HAVING**

Предложение HAVING запроса SELECT применяется для наложения условий на строки, возвращаемые при использовании предложения GROUP BY. Оно состоит из ключевого слова HAVING, за которым следует <условие\_поиска>:

<условие\_поиска> ::= [NOT] <условие\_поиска1>

[[AND|OR][NOT] <условие\_поиска2>]…, где <условие\_поиска> позволяет исключить из результата группы, не удовлетворяющие заданным условиям. Условие поиска совпадает с условием поиска, рассмотренным выше для предложения WHERE. Однако в качестве значения часто используется значение, возвращаемое агрегатными функциями. Результат совместной работы HAVING с GROUP BY аналогичен результату работы запроса SELECT с предложением WHERE с той разницей, что HAVING выполняет те же функции над строками (группами) возвращаемого набора данных, а не над строками исходной таблицы. Из этого следует, что предложение HAVING начинает свою работу после того, как предложение GROUP BY разделит базовую таблицу на группы. В противоположность этому использование предложения WHERE приводит к тому, что сначала отбираются строки из базовой таблицы и только после этого отобранные строки начинают использоваться.

Например, чтобы для каждого из товара, количество которого больше 1, вывести сумму таковых в наличии, нужно выполнить запрос, приведенный на рисунке 21.



Результаты запроса:

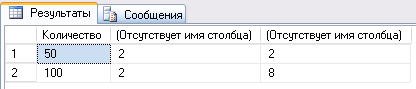


Рисунок 21 – Запрос и результаты его выполнения

Работа этого запроса заключается в следующем. Вначале GROUP BY из таблицы Товар формирует группы, состоящие из одинаковых значений поля Количество. После этого в предложении HAVING происходит подсчет числа строк, входящих в каждую группу, в результирующую таблица включаются все группы, которые содержат более одной строки. Следует отметить, что если задать условие COUNT(\*)>1 в предложении WHERE, то такой запрос потерпит неудачу, так как предложение WHERE производит оценку в терминах одиночной строки, а агрегатные функции оцениваются в терминах групп строк. В то же время из этого не следует, что предложение WHERE не используется с предложением HAVING.

Следует учесть, что предложение HAVING должно ссылаться только на агрегатные функции и элементы, выбранные GROUP BY.

Например, следующий запрос потерпит неудачу:

SELECT Количество, MAX ([Цена за единицу]) FROM Товар

GROUP BY Количество

HAVING [Код товара] = 1;

Поле [Код товара] не может быть использовано в предложении HAVING, потому что оно может иметь (и действительно имеет) больше чем одно значение на группу вывода.

Запрос на рисунке 22 является корректным.



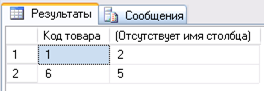
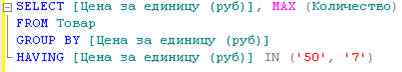


Рисунок 22 – Запрос и результаты его выполнения

Если необходимо узнать максимальные количество товаров в наличии, цена которых начинается на «50» и «7», то можно выполнить запрос, представленный на рисунке 23.



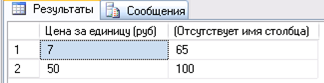


Рисунок 23 – Запрос и результаты его выполнения

# Вложенные подзапросы

В предложении SELECT могут использоваться простые соотнесенные вложенные запросы. При использовании простого подзапроса возвращенный им результат вставляется во все строки, формируемые внешним запросом. В предложении SELECT может использоваться только <скалярный\_подзапрос>, то есть подзапрос, который возвращает только одно значение.

Пусть имеются две таблицы (рисунок 24) и необходимо по каждому абоненту вывести среднее значение его оплаты, а также среднее значение начислений по всем абонентам.

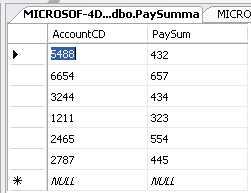
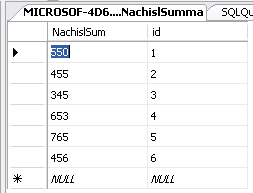


Рисунок 24 – Исходные данные

Запрос может выглядеть следующим образом:

SELECT AccountCD, AVG (PaySum) AS AVG\_Pay,

(SELECT AVG (NachislSum) FROM NachislSumma )

AS AVG\_All\_Nachisl

FROM PaySumma

GROUP BY AccountCD

Результат выполнения запроса представлены на рисунке 25.

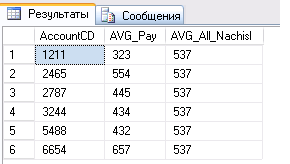


Рисунок 25 – Результаты выполнения запроса

Как следует из этого примера, связь между значением, возвращаемым простым вложенным запросом (среднее значение всех начисленных сумм по всем абонентам), и значениями внешнего запроса фактически отсутствует.

В предложении FROM могут быть определены две и более производные таблицы. Например, требуется вывести среднее количество ремонтных заявок, приходящихся на одного абонента. Для этого нужно определить общее количество ремонтных заявок, общее число абонентов и поделить полученное количество заявок на число абонентов.

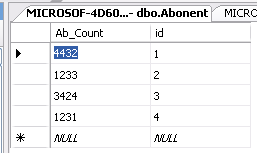
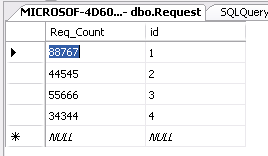


Рисунок 26 – Исходные данные

Запрос к таблицам, представленным на рисунке 26, будет выглядеть следующим образом:

SELECT (CAST (R.Req\_Count AS NUMERIC(5,2)) / A.Ab\_Count)

AS Req\_on\_Ab

FROM (SELECT COUNT (\*) FROM Abonent) AS A (Ab\_Count),

(SELECT COUNT (\*) FROM Request) AS R (Req\_Count)

Результат выполнения запроса представлены на рисунке 27.

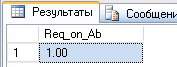


Рисунок 27 – Результаты выполнения запроса

Чтобы получить данные о заказах, все количество которых не будет превышать 80 и исключить из этой группы заказ с номером 76, предыдущий запрос можно модифицировать для таблице, представленной на рисунке 28, следующим образом:

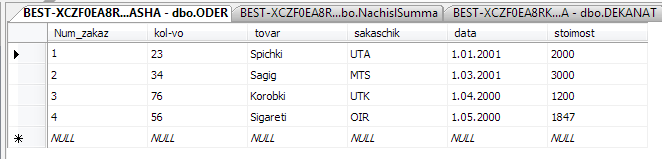


Рисунок 28 – Исходные данные

SELECT \*

FROM ODER

WHERE num\_zakaz IN

(SELECT num\_zakaz

from ODER

where [kol-vo] < 80)

and num\_zakaz NOT IN

(SELECT num\_zakaz from ODER

where [kol-vo] = 76 GROUP BY num\_zakaz);

Результат выполнения запроса представлен на рисунке 29.

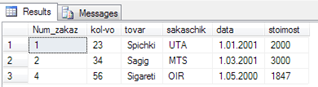


Рисунок 29 – Результат выполнения запроса

Во вложенном запросе можно использовать агрегатные функции. Допустим, необходимо вывести абонентов и значения их начислений за 2001 год (исходные данные представлены в таблице на рисунке 30), превышающие среднюю сумму начислений по всем абонентам за этот год.

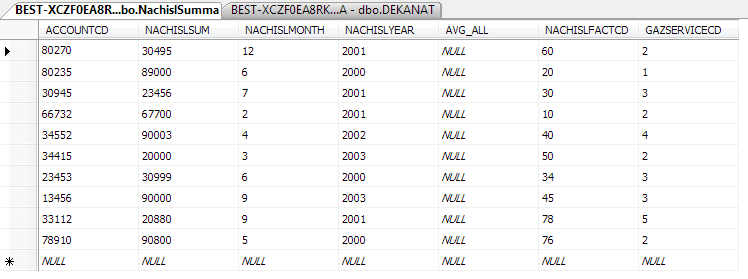


Рисунок 30– Исходные данные

Запрос будет иметь следующий вид:

SELECT AccountCD, NachislSum, NachislMonth, NachislYear,(SELECT AVG(NachislSum)

FROM NachislSumma

GROUP BY NachislYear

HAVING NachislYear=2001) AS Avg\_All

FROM NachislSumma

WHERE NachislSum > (SELECT AVG(NachislSum)

FROM NachislSumma

GROUP BY NachislYear

HAVING NachislYear=2001)

AND NachislYear=2001

ORDER BY 1;

Результат выполнения запроса представлен на рисунке 31.

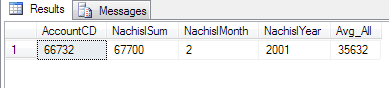
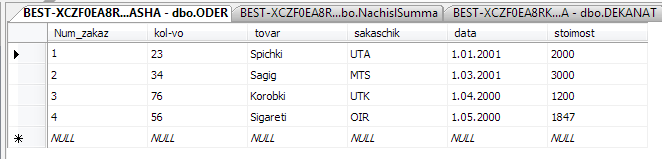


Рисунок 31 – результат выполнения запроса

В этом примере вложенный запрос выполняется один раз, возвращая среднее значение поля NachislSum за 2001 год. Затем это значение последовательно сравнивается с каждой выбираемой строкой из таблицы NachislSumma.

Рассмотрим еще два примера. Для вывода погашенных ремонтных заявок с наиболее поздней датой поступления можно (исходные данные представлены в таблице на рисунке 32), используя следующий запрос:

Рисунок 32 – Исходные данные

SELECT \* FROM ODER

WHERE [kol-vo] = (SELECT MAX ([kol-vo])

FROM ODER

WHERE num\_zakaz < 5);

Результат выполнения запроса представлен на рисунке 33.

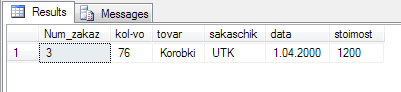


Рисунок 33 – Результат выполнения запроса

Вложенные запросы можно применять в предложении HAVING. Они могут использовать свои собственные агрегатные функции (если эти функции не возвращают многочисленных значений). Также в подзапросе, включенном в условие поиска предложения HAVING внешнего запроса, могут использоваться свои собственные предложения GROUP BY и HAVING. Следует помнить, что аргументы, указанные в HAVING, должны присутствовать в качестве аргументов и в GROUP BY.

Например, для подсчёта числа абонентов с максимальными значениями оплаты за 2000 год (исходные данные представлены в таблице на рисунке 34) можно использовать следующий запрос:

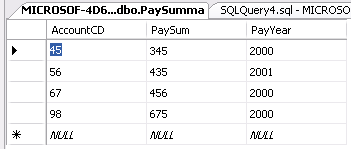


Рисунок 34 – Исходные данные

SELECT COUNT(DISTINCT AccountCD), PaySum

FROM PaySumma

GROUP BY PaySum

HAVING PaySum = (SELECT MAX(PaySum)

FROM PaySumma WHERE PayYear = 2000)

Результат выполнения запроса представлен на рисунке 35:



Рисунок 35 – Результат выполнения запроса

Функции **LEFT** и **RIGHT** используются для выделения нужного количества символов из начала или конца определенной строки соответственно и имеют следующий формат:

LEFT (строковое выражение, длина),

RIGHT (строковое выражение, длина),

где строковое выражение – выражение, из которого будут выделяться

символы; длина – количество выделяемых символов в начале (для LEFT) или конце (для RIGHT) строки.

Функция **OVERLAY** заменяет в исходной строке подстроку, начинающуюся с номера позиция и имеющую размер длина, на значение строки для замены. Функция имеет следующий формат:

**OVERLAY** («исходная строка» PLACING «строка для замены» FROM позиция [ FOR длина ] ), где «исходная строка» и «строка для замены» могут представлять собой строковое выражение.

Если длина не указана, то по умолчанию принимается длина в символах строки для замены (CHAR\_LENGTH («строка для замены»)).

Функция **REPLACE** заменяет все вхождения «подстроки» в «строковое выражение» на указанную «строку для замены». Имеет следующий формат:

REPLACE (строковое выражение, подстрока, строка для замены).

К скалярным функциям относится также функция **TRIM**. Эта функция возвращает строку аргумента, удаляя символы (по умолчанию – пробелы) из начала и/или конца строки.

Для вызова функции используется следующий синтаксис:

TRIM ( [ [LEADING | TRAILING | BOTH ] [удаляемая подстрока ]FROM ] «строковое выражение»).

LEADING указывает на то, что надо удалить указанную подстроку из начала строки, TRAILING – из конца строки, BOTH – из начала и конца строки. При использовании функции следует учитывать следующее:

1) если LEADING, TRAILING или BOTH не указаны, то принимается BOTH по умолчанию;

2) если удаляемая подстрока не определена, то за нее принимается по умолчанию пустая строка;

3) если LEADING, или TRAILING, или BOTH и/или удаляемая подстрока указаны, то после слова FROM должно быть обязательно указано строковое выражение, из которого удаляются символы;

4) указание строки, из которой удаляются символы, после слова FROM не может быть использовано самостоятельно (если LEADING, TRAILING, BOTH и удаляемая подстрока не указаны).

Существуют функции **LPAD** и **RPAD**, которые дополняют строку аргумента слева (LPAD) или справа (RPAD) указанной последовательностью символов (строка заполнитель) до заданного размера (длина). В случае если «строка заполнитель» не указана, для дополнения используется пробел. «Строка заполнитель» обрезается, когда результирующая строка достигает заданной длины. Функции имеют следующий формат:

LPAD(«строковое выражение», длина, [строка заполнитель]),

RPAD(«строковое выражение», длина, [строка заполнитель]).

Для определения символа по известному коду используется функция **ASCII\_CHAR**. Функция имеет следующий формат:

ASCII\_CHAR (код символа)

Код должен лежать в диапазоне от 0 до 255. Следует учесть, что функция возвращает символ для кодировки NONE.

Рассмотренные выше строковые функции возвращают результат в виде строки символов. Существует также ряд строковых функций, которые в качестве результата возвращают числовое значение.

Для определения кода первого символа в указанной строке используется функция **ASCII\_VAL**. Функция имеет следующий формат:

ASCII\_VAL (строка)

Функция вернет ноль, если указана пустая строка.

Для определения позиции первого вхождения заданной подстроки в строку можно использовать функцию **POSITION**. Функция имеет следующий формат:

POSITION (подстрока IN строковое выражение)

Функция возвращает ноль, если подстрока отсутствует внутри строки.

**Функции даты и времени.**

Функции выполняют различные действия над входными значениями времени и даты и возвращают строковое, числовое значение или значение в формате даты и времени.

Для получения значений текущей даты и системного времени сервера используются следующие функции:

* Получение текущей системной даты и времени:

SELECT CURRENT\_TIMESTAMP;

Получение текущего системного времени:

SELECT CONVERT (TIME, CURRENT\_TIMESTAMP)

* Функция **DATEADD** возвращает значение типа дата, время или дата/время, увеличенное или уменьшенное (если количество отрицательное) по сравнению с исходным значением на заданное количество лет, месяцев, дней, недель, часов, минут или секунд. Синтаксис имеет следующий вид: DATEADD (datepart , number , date ).

SELECT DATEADD(MONTH, 2, ’04-05-2006’)

Следует отметить следующее: YEAR, MONTH, DAY и WEEKDAY в качестве временного отрезка не могут использоваться со значениями типа время (например, тип TIME); HOUR, MINUTE и SECOND в качестве временного отрезка не могут использоваться со значениями типа дата (например, тип DATE); все значения временного отрезка могут быть использованы для типов дата/время (тип TIMESTAMP).

* Для определения величины временного промежутка от первого заданного значения типа дата, время или дата/время до второго может использоваться функция **DATEDIFF**. Синтаксис выглядит следующим образом: DATEDIFF («временной отрезок», «значение1», «значение2» ), где «временной отрезок» имеет тот же синтаксис, что и в функции DATEADD.

SELECT DATEDIFF(YEAR, ‘2002-12-31 23:59:59.9999999’,’2011-01-01 00:00:00.0000000’;

Следует отметить следующее: функция возвращает положительное число, если «значение 2» превышает «значение 1», отрицательное, если «значение 1» превышает «значение 2», и ноль – если значения равны. Если результат вычисления дробный, то выводится округленное значение, сравнение значения типа DATE со значением типа TIME недопустимо, как и для функции DATEADD, определенные временные отрезки могут использоваться только с соответствующим им типом.

**Формирование связанных подзапросов**

При использовании подзапросов во внутреннем запросе можно ссылаться на таблицу, имя которой указано в предложении FROM внешнего запроса. В этом случае такой связанныйподзапрос выполняется по одному разу для каждойстроки таблицы основного запроса. Предложение GROUP BY позволяет группировать выводимые SELECT – запросом записи по значению некоторого поля. Использование предложения HAVING позволяет при выводе осуществлять фильтрацию групп данных. Предикат предложения HAVING оценивается не для каждой строки результата, а для каждой группы выходных записей, сформированной предложением GROUP BY внешнего запроса.

Напишем запрос с подзапросом для получения данных обо всех оценках студента с фамилией «Иванов». Предположим, что его персональный номер неизвестен.

select MARK

from [EXAM MARKS]

where [STUDENT ID] =

(select [STUDENT ID]

from STUDENT

where SERNAME = 'Иванов');

Напишем запрос, выбирающий данные об именах всех студентов, имеющих по предмету с идентификатором 10 балл выше общего среднего балла.

select NAME, SERNAME

from STUDENT

where [STUDENT ID] in

(select [STUDENT ID]

from [EXAM MARKS]

where [SUBJ ID] = 10);

Напишем запрос, который выполняет выборку имен всех студентов, имеющих по предмету с идентификатором 10 балл ниже общего среднего балла

select NAME, SERNAME

from STUDENT

where [STUDENT ID] in

(select [STUDENT ID]

from [EXAM MARKS]

where [SUBJ ID] = 10);

Напишем запрос, выполняющий вывод количества предметов, по которым экзаменовался каждый студент, сдававший более 1 предмета.

select COUNT([SUBJ ID])

from [EXAM MARKS] A

group by [STUDENT ID]

having 1 < (select COUNT([SUBJ ID])

from [EXAM MARKS] B

where A.[STUDENT ID] = B.[STUDENT ID]);

Напишем команду SELECT, использующую связанные подзапросы и выполняющую вывод имен и идентификаторов студентов, у которых стипендия совпадает с максимальным значением стипендии для города, в котором живет студент.

select NAME, [STUDENT ID]

from STUDENT A

where STIPEND =

(select MAX (STIPEND)

from STUDENT B

where A.CITY = B.CITY);

Напишем запрос, который позволяет вывести имена и идентификаторы всех студентов, для которых точно известно, что они проживают в городе, где нет ни одного университета.

select NAME, [STUDENT ID]

from STUDENT A

where CITY not in

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

Напишем два запроса, которые позволяют вывести имена и идентификаторы всех студентов, для которых точно известно, что он проживают не в том городе, где расположен их университет.

select NAME, [STUDENT ID]

from STUDENT A

where CITY not in

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

**Использование оператора EXISTS**

Используемый в SQL оператор EXISTS 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 хотя бы одного найденного значения*.* В противном случае результат подзапроса ложный. Оператор EXISTS не может принимать значение UNKNOWN (не известно).

Напишем запрос с EXISTS, позволяющий вывести данные обо всех студентах, обучающихся в вузах, которые имеют рейтинг выше 300

select \*

from STUDENT as A

where exists

(select RATING

from UNIVERSITY as B

where A.[UNIV ID] = B.[UNIV ID] and RATING > 300);

Результаты запроса представлены на рисунке 36.

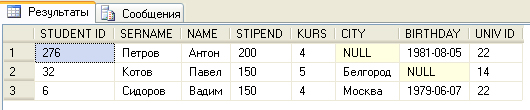


Рисунок 36 – Результаты запроса с использованием EXISTS

Напишем запрос с EXISTS, выбирающий сведения обо всех студентах, для которых в том же городе, где живет студент, существуют университеты, в которых он не учится.

select \*

from STUDENT A

where exists

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

Результаты запроса представлены на рисунке 37.



Рисунок 37 – Результаты запроса с использованием EXISTS

Напишем запрос, выбирающий из таблицы SUBJECT данные о названиях предметов обучения, экзамены по которым сданы более чем одним студентом.

select distinct [SUBJ ID]

from [EXAM MARKS] as A

where exists

(select \*

from [EXAM MARKS] as B

where A.[SUBJ ID] = B.[SUBJ ID]

AND A.[STUDENT ID] < > B.[STUDENT ID]);

Напишем запрос, выбирающий данные о названиях университетов, рейтинг которых равен или превосходит рейтинг Воронежского государственного университета.

SELECT \*

FROM UNIVERSITY

WHERE RATING > = ALL

(SELECT RATING

FROM UNIVERSITY

WHERE CITY = 'Воронеж') ;

Результаты запроса представлены на рисунке 38.

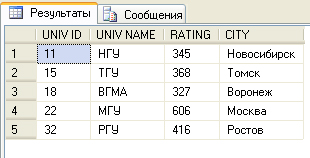


Рисунок 38 – Результаты запроса

Напишем запрос, использующий ALL и выполняющий выборку данных о студентах, у которых в городе их постоянного местожительства нет университета.

select NAME, [STUDENT ID]

from STUDENT A

where CITY < > ALL

(select CITY

from UNIVERSITY B

where A.CITY = B.CITY);

Результаты запроса представлены на рисунке 39.

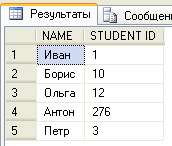


Рисунок 39 – Результаты запроса с использованием ALL

Напишем запрос, выбирающий из таблицы EXAM\_MARKS данные о названиях предметов обучения, для которых значение полученных на экзамене оценок (поле MARK) превышает любое значение оценки для предмета, имеющего идентификатор, равный 10.

SELECT \*

FROM [EXAM MARKS]

WHERE MARK > ANY

(SELECT MARK

FROM [EXAM MARKS]

WHERE [SUBJ ID] = 10);

Результаты запроса представлены на рисунке 40.

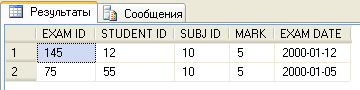


Рисунок 40 – Результаты запроса с использованием ANY

**Связанные подзапросы в предложениях WHERE и HAVING**

При использовании связанного вложенного запроса в условиях поиска предложений WHERE и HAVING он может представлять собой скалярный подзапрос, подзапрос столбца или табличный подзапрос, как и для простых вложенных запросов. Однако так как запрос связанный, внутренний запрос выполняется отдельно для каждой строки внешнего запроса (*текущая строка-кандидат*). Рассмотрим примеры, в которых используются скалярный подзапрос и подзапрос столбца. Подзапросы, представляющие собой табличный подзапрос, будут рассмотрены позднее при изучении предикатов EXISTS и SINGULAR.

Например, чтобы вывести все сведения об абонентах, которые подали заявки на ремонт газового оборудования 17 декабря 2001 года, можно использовать следующий связанный вложенный запрос:

SELECT \* FROM Abonent Out

WHERE '17.12.2001' IN

(SELECT IncomingDate FROM Request Inn

WHERE Out. AccountCD = Inn. AccountCD);.

В этом примере Out и Inn - это псевдонимы таблиц Abonent и Request соответственно (могут задаваться произвольно). Так как значение в поле AccountCD внешнего запроса меняется (при переборе строк), внутренний запрос должен выполняться отдельно для каждой строки внешнего запроса.

В этом примере SQL осуществляет следующую процедуру:

1) выбирает строку c данными об абоненте, имеющем номер лицевого счета «005488» (первая строка) из таблицы Abonent;

2) сохраняет эту строку как текущую строку – кандидат под псевдонимом Out;

3) выполняет вложенный запрос, просматривающий всю таблицу Request, чтобы найти строки, где значение поля Inn.AccountCD – такое же, как значение Out.AccountCD (005488). Затем из каждой такой строки таблицыRequest извлекается поле IncomingDate. В результате вложенный запрос, представляющий собой подзапрос столбца, формирует набор значений поля IncomingDate для текущей строки – кандидата;

4) после получения набора всех значений поля IncomingDate для поля AccountCD= «005488» анализируется условие поиска основного запроса, чтобы проверить, имеется ли значение 17 декабря 2001 в наборе всех значений поля IncomingDate. Если это так (а это так), то выбирается строка с номером лицевого счета «005488» для вывода ее из основного запроса;

5) повторяются пункты 1- 4 (для строки с номером лицевого счета «015527» и т.д.), пока каждая строка таблицы Abonent не будет проверена.

Рассмотрим пример сравнения значения, возвращаемого вложенным запросом, с константой. Вывести информацию об исполнителях ремонтных заявок, назначенных на выполнение четырех и более заявок, можно с помощью следующего запроса:

SELECT \*

FROM Executor E

WHERE 4<= (SELECT COUNT(R.RequestCD) FROM Request R

WHERE E.ExecutorCD = R.ExecutorCD);.

В данном примере связанный подзапрос в условии поиска представляет собой скалярный подзапрос. Он возвращает одно единственное значение (количество ремонтных заявок) для текущей строки – кандидата, выбранной из таблицы Executor. Если это значение больше или равно 4, то текущая строка - кандидат выбирается для вывода из основного запроса. Эта процедура повторяется, пока каждая строка таблицы Executor не будет проверена.

В SQL имеется возможность использовать соотнесенный вложенный запрос, основанный на той же самой таблице, что и основной запрос*.* Это позволяет использовать соотнесенные вложенные запросы для извлечения сложных форм производной информации. Например, вывести размеры начислений, превышающие среднее значение начислений для каждого абонента, можно с помощью следующего запроса:

SELECT F.NachislSum,

(SELECT AVG(D.NachislSum)

FROM NachislSumma D

WHERE F.AccountCD = D.AccountCD) AS AVG\_D,

A.AccountCD, A.Fio

FROM Abonent A, NachislSumma F

WHERE F.NachislSum >

(SELECT AVG(S.NachislSum)

FROM NachislSumma S

WHERE F.AccountCD = S.AccountCD)

AND A.AccountCD = F.AccountCD

ORDER BY 3;

В этом примере производится одновременная оценка среднего значения для всехстрок, удовлетворяющих условию поиска в предложении WHERE вложенного связанного запроса, одной и той же таблицы со значениями строки-кандидата. Выбирается первая строка-кандидат из таблицы NachislSumma и сохраняется под псевдонимом F. Выполняется вложенный запрос, просматривающий ту же самую таблицу NachislSumma с самого начала, чтобы найти все строки, где значение поля S.AccountCD - такое же, как значение F.AccountCD. Затем по всем таким строкам в таблицеNachislSumma вложенный запрос (скалярный подзапрос) подсчитывает среднее значение поля NachislSum. Анализируется условие поиска основного запроса, чтобы проверить, превышает ли значение поля NachislSum из текущей строки - кандидата среднее значение, вычисленное вложенным запросом. Если это так, то текущая строка-кандидат выбирается для вывода. Таким образом, производятся одновременно и вычисление среднего, и отбор строк, удовлетворяющих условию.

**Примечание.** Запрос, использующий агрегатную функцию в условии поиска основного запроса (данная функция является возвращаемым элементом вложенного запроса), нельзя сформулировать с помощью техники соединения таблиц.

Рассмотрим использование соотнесенного вложенного запроса в условии поиска предложения HAVING. Условие поиска предложения HAVING в подзапросе оценивается для каждой группыиз внешнего запроса, а не для каждой строки. Следовательно, вложенный запрос будет выполняться один раз для каждой группы, выведенной из внешнего запроса, а не для каждой строки (как это было при использовании в предложении WHERE).

Например, чтобы подсчитать общие суммы начислений за услуги газоснабжения для абонентов, чьи фамилии начинаются с буквы C, можно использовать следующий соотнесенный вложенный запрос:

SELECT N.AccountCD, SUM(N.NachislSum)

FROM NachislSumma N

GROUP BY N.AccountCD

HAVING N.AccountCD =

(SELECT A.AccountCD FROM Abonent A

WHERE A.AccountCD = N.AccountCD AND A.Fio LIKE 'C%');

Этот запрос выполняется следующим образом. Основной запрос один раз группирует таблицу NachislSumma по полю AccountCD. Затем для каждой группы выполняется связанный вложенный запрос, возвращая единственное значение поля AccountCD таблицы Abonent (т.к. поле AccountCD содержит уникальные значения).

**Предикаты ANY и ALL**

Операции сравнения можно расширить до многократного сравнения с использованием предикатов ANY и ALL. Это расширение используется при сравнении значения определенного столбца со значениями, возвращаемыми вложенным запросом. Предикат ANY, указанный после знака любой из операций сравнения, означает, что будет возвращено TRUE, если хотя бы для одного значения из подзапроса результат сравнения истинен. Предикат ALL требует, чтобы результат сравнения был бы истинен для всех значений, возвращаемых подзапросом.

Рассмотрим использование предиката **ANY**.

Например, требуется вывести информацию об оплатах абонентами за услугу газоснабжения с кодом, равным 2, за период до 2001 года, которые превышают хотя бы одну из сумм, оплаченных за эту же услугу за 2001 год. Запрос будет выглядеть следующим образом:

SELECT \*

FROM PaySumma

WHERE PaySum > ANY (SELECT PaySum

FROM PaySumma

WHERE PayYear=2001 AND

GazServiceCD=2)

AND PayYear<2001 AND GazServiceCD=2;

В этом примере вложенный запрос выполняется один раз, возвращая все значения поля PaySum, для которых истинно условие PayYear=2001 и GazServiceCD=2 (58.7, 250, 20, 20, 80…). Затем значения, выбранные подзапросом, последовательно сравниваются со значением поля PaySum для каждой строки из таблицы PaySumma основного запроса. При первом обнаруженном совпадении сравнение прекращается и соответствующая строка выводится. Условие «> ANY» равносильно утверждению «больше, чем минимальное из существующих», а условие «< ANY» - «меньше, чем максимальное из существующих». Становится очевидным, что такие условия можно записать иначе, используя агрегатные функции MIN и MAX.

Таким образом, предыдущий запрос можно переписать так:

SELECT \*

FROM PaySumma

WHERE PaySum > (SELECT MIN (PaySum)

FROM PaySumma

WHERE PayYear=2001 AND

GazServiceCD=2)

AND PayYear<2001 AND GazServiceCD=2;

Следует отметить, что использование сравнения «= ANY» эквивалентно использованию предиката IN.

Рассмотрим использование предиката **ALL**.

Например, требуется вывести информацию о ремонтных заявках абонентов, даты подачи заявок которых позднее, чем заявки любых абонентов с неисправностью с кодом, равным 2. Запрос будет выглядеть следующим образом:

SELECT \* FROM Request

WHERE IncomingDate > ALL (SELECT IncomingDate

FROM Request

WHERE FailureCD=2);

Если требуется вывести информацию о ремонтных заявках абонентов, даты выполнения заявок у которых позднее, чем даты выполнения заявок любых абонентов по неисправности с кодом, равным 2, то запрос будет выглядеть следующим образом:

SELECT \* FROM Request

WHERE ExecutionDate > ALL (SELECT ExecutionDate

FROM Request

WHERE FailureCD=2);

В процессе выполнения данного запроса подзапросом формируется набор значений поля ExecutionDate, взятых из строк, где FailureCD=2. В результате условие поиска внешнего запроса будет выглядеть следующим образом: ExecutionDate > ALL (24.10.1998, 11.10.2001, 14.09.2001).

Условие «> ALL» равносильно утверждению «больше, чем максимальное», а условие «< ALL» - «меньше, чем минимальное». Становится очевидным, что такие условия можно записать иначе, используя агрегатные функции MAX и MIN. Таким образом, предыдущий запрос можно переписать так:

SELECT \* FROM Request

WHERE IncomingDate > (SELECT MAX (IncomingDate)

FROM Request

WHERE FailureCD=2);

Результат выполнения будет таким же, как и в предыдущем примере. Следует отметить, что использование сравнения «<> ALL» эквивалентно использованию предиката NOT IN, независимо от того, простой или связанный подзапрос используется. Рассмотрим использование связанного подзапроса с предикатом ALL. Пусть требуется вывести названия неисправностей, по которым все ремонтные заявки подавались позднее 1 мая 2001 года. Запрос будет выглядеть следующим образом:

SELECT d.FailureNM

FROM Disrepair d

WHERE '01.05.2001' < ALL (SELECT r.IncomingDate

FROM Request r

WHERE d.FailureCD=r.FailureCD);

**Предикат SINGULAR**

Совместно с подзапросами можно использовать предикат SINGULAR, который проверяет, возвращает ли подзапрос в точности одно значение. Если возвращается NULL или более одного значения, то SINGULAR дает ложь (а NOT SINGULAR - истину). Предикат SINGULAR похож на предикат ALL, за исключением того, что он проверяет наличие одного и только одного соответствующего значения в наборе. С предикатом SINGULAR могут использоваться как простые, так и соотнесенные подзапросы, однако часто использование простого запроса не имеет логического смысла. Следующий запрос отыскивает всех абонентов, которые имеют только одну ремонтную заявку:

SELECT \* FROM Abonent A

WHERE SINGULAR (SELECT RequestCD FROM Request R

WHERE A.AccountCD=R.AccountCD);

Если использовать в предыдущем запросе предикат NOT SINGULAR, то будут выведены абоненты, у которых имеется более одной ремонтной заявки или вообще нет заявок:

SELECT \*

FROM Abonent A

WHERE NOT SINGULAR (SELECT RequestCD FROM Request R

WHERE A.AccountCD=R.AccountCD);.