МИНОБРНАУКИ РОССИИ

Передовые инженерные школы

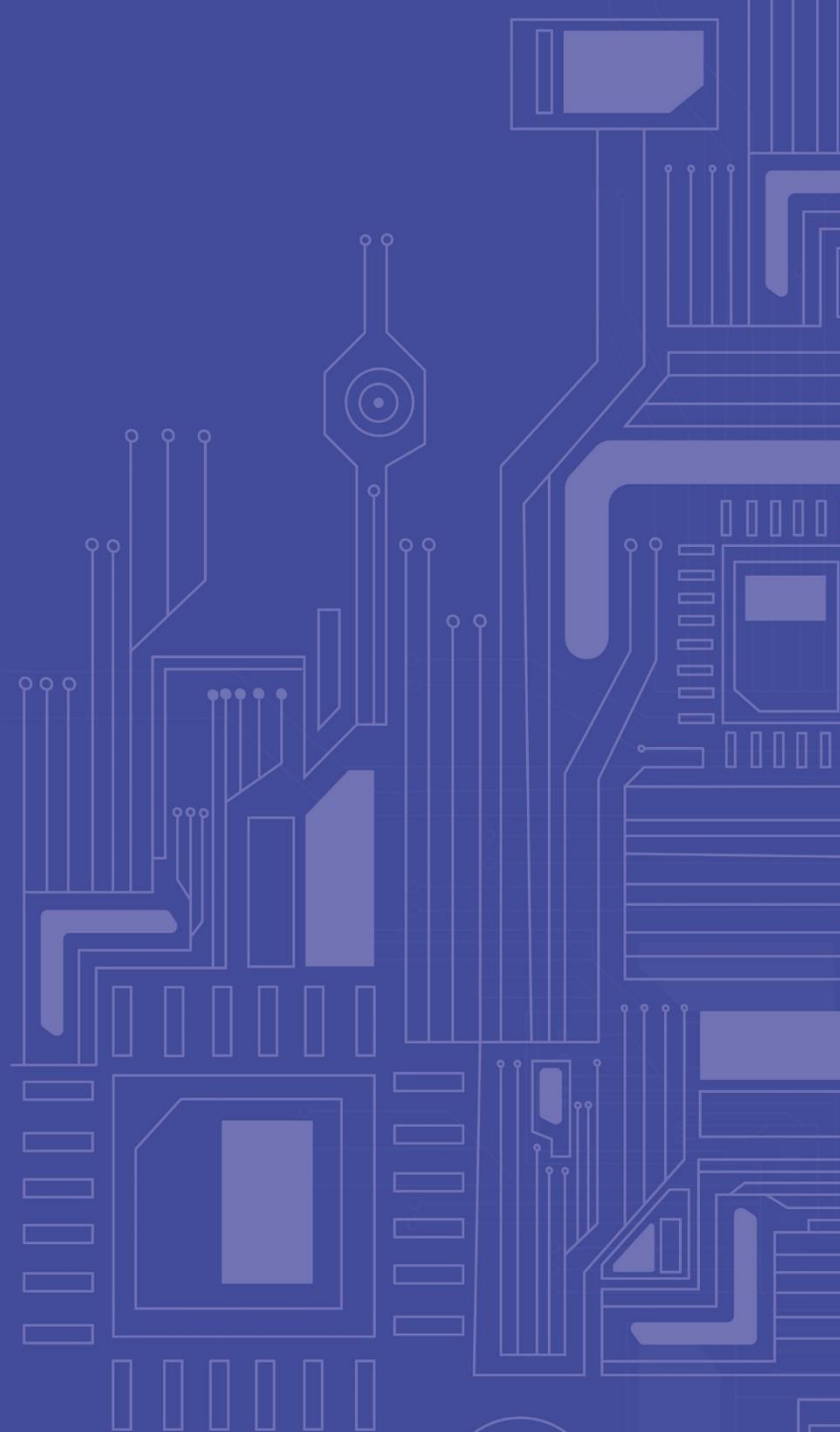
СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Лекция 11

ПЛАН ЛЕКЦИИ

- Логические выражения раздела WHERE
- Соединения
- Агрегатные функции
- Логические выражения раздела HAVING

ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ РАЗДЕЛА WHERE



ПРЕДИКАТЫ WHERE

Синтаксически логическое выражение раздела WHERE определяется как булевское выражение (boolean_value_expression). Основой логического выражения являются предикаты. Предикат позволяет специфицировать условие, результатом вычисления которого может быть true, false или unknown. В языке SQL:1999 допустимы следующие предикаты:

```
predicate ::= comparison_predicate
    | between_predicate
    | null_predicate
    | in_predicate
    | like_predicate
    | similar_predicate
    | exists_predicate
    | unique_predicate
    | overlaps_predicate
    | quantified_comparison_predicate
    | match_predicate
    | distinct_predicate
```

Набор допустимых предикатов в SQL явно избыточен, но тем не менее в языке SQL имеется явная тенденция расширения этого набора. В частности, в SQL:2003 в связи с введением генератора типов мультимножеств в дополнение ко всем разновидностям предикатов SQL:1999 появилось три новых вида предикатов: предикаты для проверки того, что заданное значение является элементом мультимножества (MEMBER); что одно мультимножество входит в другое мультимножество (SUBMULTISET) и что мультимножество не содержит дубликаты (IS A SET).

ПРИМЕРЫ НА ОСНОВЕ БАЗЫ ДАННЫХ

EMP:

EMP_NO : EM_NO
EMP_NAME : VARCHAR
EMP_BDATE : DATE
EMP_SAL : SALARY
DEPT_NO : DEPT_NO
PRO_NO : PRO_NO

DEPT:

DEPT_NO : DEPT_NO
DEPT_NAME : VARCHAR
DEPT_EMP_NO : INTEGER
DEPT_TOTAL_SAL : SALARY
DEPT_MNG : EMP_NO

PRO:

PRO_NO : PRO_NO
PRO_TITLE : VARCHAR
PRO_SDATE : DATEP
PRO_DURAT : INTERVAL
PRO_MNG : EMP_NO
PRO_DESC : CLOB

Столбцы EMP_NO, DEPT_NO и PRO_NO являются первичными ключами таблиц EMP, DEPT и PRO соответственно. Столбцы DEPT_NO и PRO_NO таблицы EMP являются внешними ключами, ссылающимися на таблицы DEPT и PRO соответственно (DEPT_NO указывает на отделы, в которых работают служащие, а PRO_NO – на проекты, в которых они участвуют; оба столбца могут принимать неопределенные значения). Столбец DEPT_MNG является внешним ключом таблицы DEPT (DEPT_MNG указывает на служащих, которые исполняют обязанности руководителей отделов; у отдела может не быть руководителя, и один служащий не может быть руководителем двух или более отделов). Столбец PRO_MNG является внешним ключом таблицы PRO (PRO_MNG указывает на служащих, которые являются менеджерами проектов, у проекта всегда есть менеджер, и один служащий не может быть менеджером двух или более проектов).

ПРЕДИКАТ СРАВНЕНИЯ (1/3)

Этот предикат предназначен для спецификации сравнения двух строчных значений. Синтаксис предиката следующий:

```
comparison_predicate ::=
    row_value_constructor comp_op row_value_constructor
comp_op ::= = | <> («не равно»)| < | >
    | <= («меньше или равно») | >= («больше или равно»)
```

Строки, являющиеся операндами операции сравнения, должны быть одинаковой степени. Типы данных соответствующих значений строк-операндов должны быть совместимы.

Пусть X и Y обозначают соответствующие элементы строк-операндов, а xv и yv – их значения. Тогда:

1. если xv и/или yv являются неопределенными значениями, то значение условия X comp_op Y - unknown;
2. в противном случае значением условия X comp_op Y является true или false в соответствии с естественными правилами применения операции сравнения.

ПРЕДИКАТ СРАВНЕНИЯ (2/3)

Правила применения операции сравнения (1/2):

- Числа сравниваются в соответствии с правилами алгебры.
- Сравнение двух символьных строк производится следующим образом:
  - если длина строки X не равна длине строки Y, то для выравнивания длин строк более короткая строка расширяется *символами набивки* (*pad symbol*); если для используемого набора символов порядок сортировки явным образом не специфицирован, то в качестве символа набивки используется пробел;
  - далее производится лексикографическое сравнение строк в соответствии с предопределенным или явно определенным порядком сортировки символов.
- Сравнение двух битовых строк X и Y основано на сравнении соответствующих бит. Если X_i и Y_i – значения i-тых бит X и Y соответственно и если lx и ly обозначает длину в битах X и Y соответственно, то:
  - X равно Y тогда и только тогда, когда lx = ly и $X_i = Y_i$ для всех i;
  - X меньше Y тогда и только тогда, когда (a) lx < ly и $X_i = Y_i$ для всех i меньших или равных lx, или (b) $X_i = Y_i$ для всех $i < n$ и $X_n = 0$, а $Y_n = 1$ для некоторого n меньшего или равного min (lx, ly).
- Сравнение двух значений типа дата-время производится в соответствии с видом интервала, который получается при вычитании второго значения из первого. Пусть X и Y – сравниваемые значения, а H – наименее значимое поле даты-времени X и Y. Результат сравнения X comp_op Y определяется как (X – Y) H comp_ op INTERVAL (0) H. (Два значения типа дата-время сравнимы только в том случае, если они содержат одинаковый набор полей даты-времени.)

ПРЕДИКАТ СРАВНЕНИЯ (3/3)

Правила применения операции сравнения (2/2):

- Сравнение двух значений анонимного строкового типа производится следующим образом. Пусть Rx и Ry обозначают строки-операнды, а Rx_i и Ry_i – i-тые элементы Rx и Ry соответственно. Вот как определяется результат сравнения Rx comp_op Ry:
  - Rx = Ry есть true тогда и только тогда, когда Rx_i = Ry_i есть true для всех i;
  - Rx <> Ry есть true тогда и только тогда, когда Rx_i <> Ry_i есть true для некоторого i;
  - Rx < Ry есть true тогда и только тогда, когда Rx_i = Ry_i есть true для всех $i < n$, и Rx_n < Ry_n есть true для некоторого n;
  - Rx > Ry есть true тогда и только тогда, когда Rx_i = Ry_i есть true для всех $i < n$, и Rx_n > Ry_n есть true для некоторого n;
  - Rx <= Ry есть true тогда и только тогда, когда Rx = Ry есть true ИЛИ Rx < Ry есть true;
  - Rx >= Ry есть true тогда и только тогда, когда Rx = Ry есть true ИЛИ Rx > Ry есть true;
  - Rx = Ry есть false тогда и только тогда, когда Rx <> Ry есть true;
  - Rx <> Ry есть false тогда и только тогда, когда Rx = Ry есть true;
  - Rx < Ry есть false тогда и только тогда, когда Rx >= Ry есть true;
  - Rx > Ry есть false тогда и только тогда, когда Rx <= Ry есть true;
  - Rx <= Ry есть false тогда и только тогда, когда Rx > Ry есть true;
  - Rx >= Ry есть false тогда и только тогда, когда Rx < Ry есть true;
  - Rx comp_op Ry есть unknown тогда и только тогда, когда Rx comp_op Ry не есть true ИЛИ false.

ПРИМЕР ЗАПРОСОВ С ПРЕДИКАТОМ СРАВНЕНИЯ

Найти номера, имена, номера отделов и имена руководителей отделов служащих, размер заработной платы которых меньше 15000 руб.

```
SELECT EMP1.EMP_NO, EMP1.EMP_NAME,
       EMP1.DEPT_NO, EMP2.EMP_NAME
FROM EMP AS EMP1, EMP AS EMP2, DEPT
WHERE EMP1.EMP_SAL < 15000.00 AND
    EMP1.DEPT_NO = DEPT.DEPT_NO AND
    DEPT.DEPT_MNG = EMP2.EMP_NO;
```

Этот запрос представляет собой эквисоединение ограничения таблицы EMP (по условию EMP_SAL < 15000.00) с таблицами DEPT и EMP (по условиям EMP.DEPT_NO = DEPT.DEPT_NO и DEPT.DEPT_MNG = EMP2.EMP_NO соответственно). Таблица EMP участвует в качестве операнда операции эквисоединения два раза. Поэтому в разделе FROM ей присвоены два псевдонима – EMP1 и EMP2. Следуя предписанному стандартом порядку выполнения запроса, можно считать, что введение этих псевдонимов обеспечивает переименование столбцов таблицы EMP, требуемое для выполнения раздела FROM с образованием расширенного декартова произведения таблиц-операндов.

Есть способ формулировки этого запроса с использованием вложенного подзапроса в качестве элемента списка выборки:

```
SELECT EMP.EMP_NO, EMP.EMP_NAME, EMP.DEPT_NO,
    (SELECT EMP_NAME
      FROM EMP
      WHERE EMP_NO = DEPT_MNG)
FROM EMP, DEPT
WHERE EMP.EMP_SAL < 15000.00 AND
    EMP.DEPT_NO = DEPT.DEPT_NO;
```

ПРЕДИКАТ BETWEEN

Предикат позволяет специфицировать условие вхождения в диапазон значений. Операндами являются строки:

```
between_predicate ::=
    row_value_constructor [ NOT ] BETWEEN
    row_value_constructor AND row_value_constructor
```

Все три строки-операнды должны иметь одну и ту же степень. Типы данных соответствующих значений строк-операндов должны быть совместимыми.

Пусть `X`, `Y` и `Z` обозначают первый, второй и третий операнды. Тогда по определению выражение `X NOT BETWEEN Y AND Z` эквивалентно выражению `NOT (X BETWEEN Y AND Z)`. Выражение `X BETWEEN Y AND Z` по определению эквивалентно булевскому выражению `X >= Y AND X <= Z`.

ПРИМЕР Найти номера, имена и размер зарплаты служащих, получающих зарплату в размере от 12000 до 15000 руб.

```
SELECT EMP_NO, EMP_NAME, EMP_SAL
FROM EMP
WHERE EMP_SAL BETWEEN 12000.00 AND 15000.00;
```

ПРЕДИКАТ IS NULL

Предикат `is null` позволяет проверить, являются ли неопределенными значения всех элементов строки-операнда:

```
null_predicate ::= row_value_constructor IS [ NOT ] NULL
```

Пусть X обозначает строку-операнд. Если значения всех элементов X являются неопределенными, то значением условия X IS NULL является true; иначе – false. Если ни у одного элемента X значение не является неопределенным, то значением условия X IS NOT NULL является true; иначе – false.

Полная семантика предиката IS NULL

Вид операнда	Вид условия			
	X IS NULL	IS NOT NULL	NOT X IS NULL	NOT X IS NOT NULL
Степень 1: значение NULL	true	false	false	true
Степень 1: значение отлично от NULL	false	true	true	false
Степень > 1: у всех элементов значение NULL	true	false	false	true
Степень > 1: у некоторых(не у всех) элементов значение NULL	false	false	true	true
Степень > 1: ни у одного элемента нет значения NULL	false	true	true	false

ПРИМЕР Найти номера и имена служащих, номер отдела которых неизвестен.

```
SELECT EMP_NO, EMP_NAME
FROM EMP
WHERE DEPT_NO IS NULL;
```

ПРЕДИКАТ IN

Предикат позволяет специфицировать условие вхождения строчного значения в указанное множество значений. Синтаксические правила следующие:

```
in_predicate ::= row_value_constructor [ NOT ]
    IN in_predicate_value
in_predicate_value ::= table_subquery
    | (value_expression_comma_list)
```

Строка, являющаяся первым операндом, и таблица-второй операнд должны быть одинаковой степени. В частности, если второй операнд представляет собой список значений, то первый операнд должен иметь степень 1. Типы данных соответствующих столбцов операндов должны быть совместимы.

Пусть `X` обозначает строку-первый операнд, а `S` – множество строк второго операнда. Обозначим через `s` строку-элемент этого множества. Тогда по определению условие `X IN S` эквивалентно булевскому выражению $\mathtt{OR}_{s \in S}$ `(X = s)`. Другими словами, `X IN S` принимает значение `true` в том и только в том случае, когда во множестве `S` существует хотя бы один элемент `s`, такой, что значением предиката `X = s` является `true`. `X IN S` принимает значение `false` в том и только том случае, когда для всех элементов `s` множества `S` значением операции сравнения `X = s` является `false`. Иначе значением условия `X IN S` является `unknown`. Заметим, что для пустого множества `S` значением `X IN S` является `false`.

По определению условие `X NOT IN S` эквивалентно `NOT (X IN S)`.

ПРИМЕРЫ НА ПРЕДИКАТ IN

ПРИМЕР 1 Найти номера, имена и номера отделов служащих, работающих в отделах 15, 17 и 19.

```
SELECT EMP_NO, EMP_NAME, DEPT_NO
FROM EMP
WHERE DEPT_NO IN (15, 17, 19);
```

<=>

```
SELECT EMP_NO, EMP_NAME, DEPT_NO
FROM EMP
WHERE DEPT_NO = 15
  OR DEPT_NO = 17
  OR DEPT_NO = 19;
```

ПРИМЕР 2 Найти номера служащих, не являющихся руководителями отделов и получающих заплату, размер которой равен размеру зарплаты какого-либо руководителя отдела.

```
SELECT EMP_NO
FROM EMP
WHERE EMP_NO NOT IN (SELECT DEPT_MNG FROM DEPT)
    AND EMP_SAL IN (SELECT EMP_SAL FROM EMP, DEPT
                    WHERE EMP_NO = DEPT_MNG);
```

<=>

```
SELECT DISTINCT EMP_NO
FROM EMP, EMP EMP1, DEPT
WHERE EMP_NO NOT IN (SELECT DEPT_MNG FROM DEPT)
 AND EMP_SAL = EMP1_SAL
 AND EMP1.EMP_NO = DEPT.DEPT_MNG;
```

ПРЕДИКАТ LIKE

```
like_predicate ::= source_value [ NOT ]
    LIKE pattern_value [ ESCAPE escape_value ]
source_value ::= value_expression
pattern_value ::= value_expression
escape_value ::= value_expression
```

Все три операнда (source_value, pattern_value и escape_value) должны быть одного типа: либо типа символьных строк, либо типа битовых строк. В первом случае значением последнего операнда должна быть строка из одного символа, во втором – строка из 8 бит. Второй операнд, как правило, задается литералом соответствующего типа. В обоих случаях значение предиката равняется true в том и только в том случае, когда исходная строка (source_value) может быть сопоставлена с заданным шаблоном (pattern_value).

ESCAPE-символ

ESCAPE-символ используется для экранирования специальных символов (`%` и `\`). В случае если вам нужно найти строки, вы можете использовать ESCAPE-символ.

Например, вы хотите получить идентификаторы задач, прогресс которых равен 3%:

```
SELECT job_id FROM Jobs
WHERE progress LIKE '3!%' ESCAPE '!';
```

Если бы мы не экранировали трафаретный символ, то в выборку попало бы всё, что начинается на 3.

В случае обработки битовых строк сопоставление шаблона со строкой производится восьмерками соседних бит (*октетами*). В соответствии со стандартом SQL:1999, при сопоставлении шаблона со строкой производится специальная интерпретация октетов со значениями X'25' и X'5F' (коды символов подчеркивания и процента в кодировке ASCII).

Значение предиката **like** есть unknown, если значение первого или второго операндов является неопределенным. Условие x NOT LIKE y ESCAPE z эквивалентно условию NOT x LIKE y ESCAPE z.

ПРЕДИКАТ SIMILAR/REGEXP

Формально предикат `similar` определяется следующими синтаксическими правилами:

```
similar_predicate ::= source_value [ NOT ]
    SIMILAR TO pattern_value [ ESCAPE escape_value ]
source_value ::= character_expression
pattern_value ::= character_expression
escape_value ::= character_expression
```

УСТАРЕВАЕТ

Все три операнда (`source_value`, `pattern_value` и `escape_value`) должны иметь тип символьных строк. Значением последнего операнда должна быть строка из одного символа. Второй операнд, как правило, задается литералом соответствующего типа. В обоих случаях значение предиката равняется `true` в том и только в том случае, когда шаблон (`pattern_value`) должным образом сопоставляется с исходной строкой (`source_value`).

Основное отличие предиката `similar` от рассмотренного ранее предиката `like` состоит в существенно расширенных возможностях задания шаблона, основанных на использовании правил построения *регулярных выражений*.

ПРЕДИКАТ REGEXP (REGEXP_LIKE)

Синтаксис:

```
column_name REGEXP pattern
```

```
SELECT *
FROM names
WHERE name REGEXP '^J';
```

In this example, the regular expression ^J matches any name that starts with the letter “J”. The ^ character is used to anchor the pattern to the beginning of the string.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ (1/4)

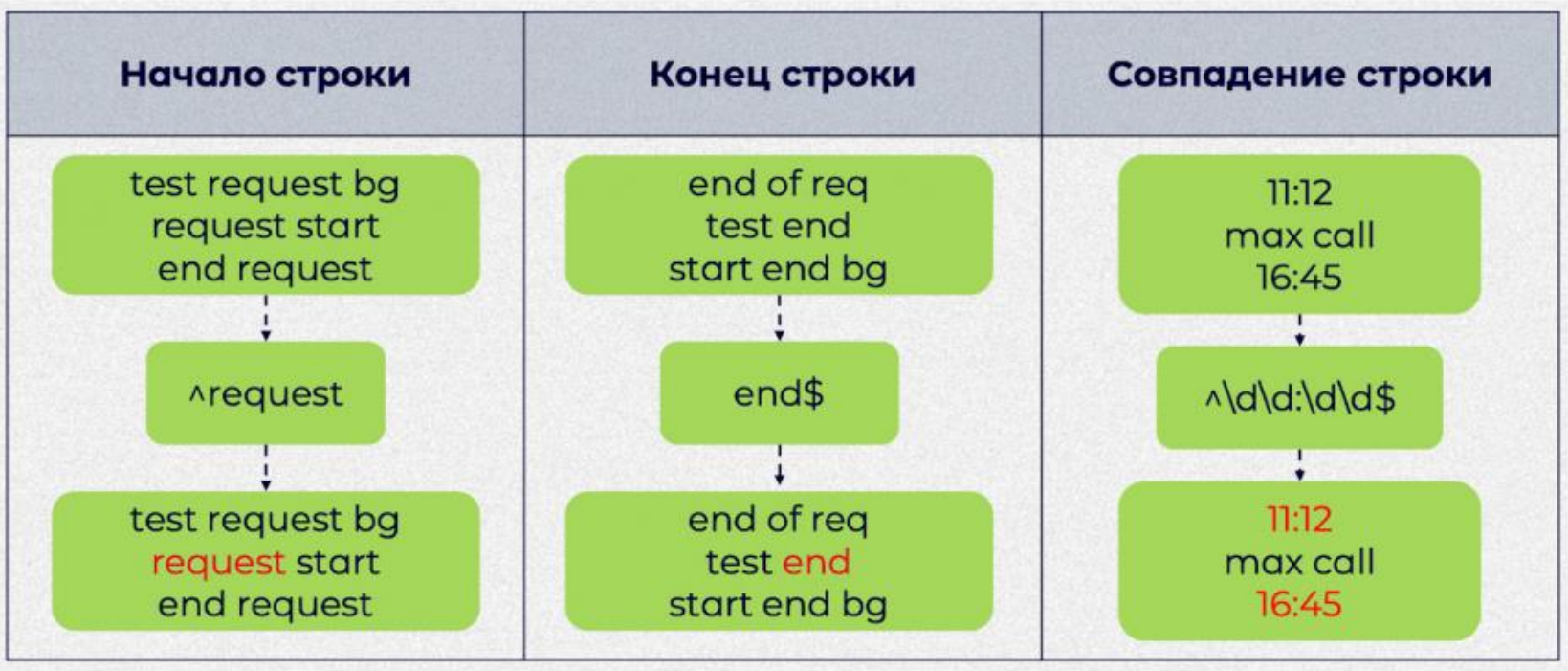
Символьные классы

Чтобы регулярные выражения могли работать с текстом, им необходимы инструменты. И первый из них, о котором мы поговорим — это символьные классы. Символьный класс обозначает принадлежность символа к определенному типу. Это может быть цифра, пробел, класс «слово» (латинские буквы, цифры и знаки подчеркивания), «не цифра», «не пробел» и «не слово». Также существует класс «точка» — он обозначает любой символ кроме символа новой строки.

\d	digit - цифра
\s	space - пробел
\w	word - слово
\D	не цифра (все кроме \d)
\S	не пробел (все кроме \s)
\W	не латиница, не знак подчёркивания и не цифра
.	любой символ кроме новой строки

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ (2/4)

Начало и конец строки

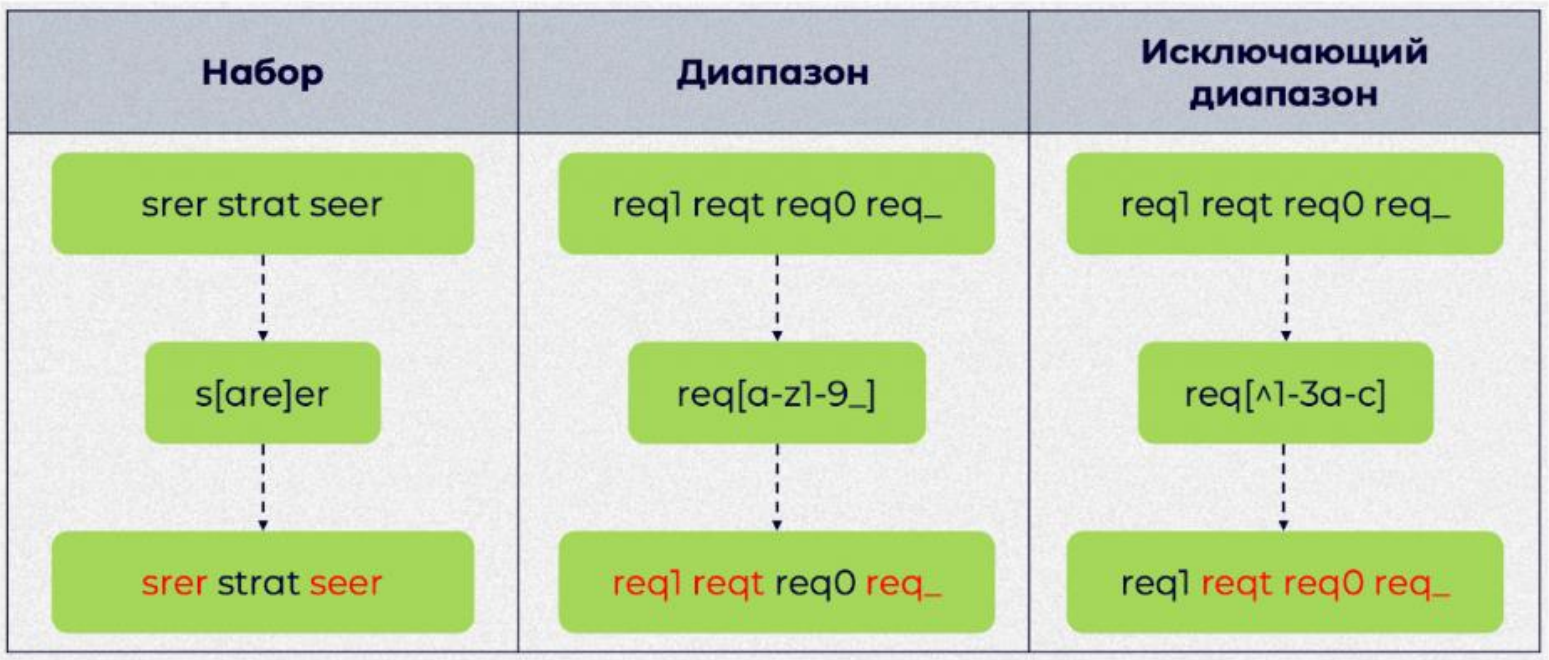


Перейдем к специальным символам начала и конца строки. Слева у нас есть последовательность. К ней мы пишем регулярку, состоящую из символа начала строки «^» и слова «request». Оно найдет все слова request, которые стоят в начале строк. В результате мы видим, что выражение сработало, и мы получили только слово «request» из начала строки.

Мы можем использовать комбинацию этих символов, чтобы найти строки в определенном формате. В третьем примере представлено более сложное регулярное выражение: символы начала и конца строки, а между ними цифры, попарно разделенные через двоеточие. В результате будут найдены все строки, имеющие заданный формат — например, время.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ (3/4)

Наборы и диапазоны



Рассмотрим наборы и диапазоны. Первое выражение начинается с буквы «s». Далее в квадратных скобках есть три символа «a», «r», «e» — это набор, который обозначает, что на втором месте у нас должен быть один из символов набора. В конце регулярного выражения — буквы «e» и «r». Согласно заданным параметрам, здесь было выбрано первое и третье слово, а второе не подошло.

Существует и исключающий диапазон, который задает, что в результат попасть не должно. Он обозначается символом «^» перед диапазоном. В третьем примере мы начинаем регулярку с «req», открываем набор через квадратные скобки, задаем исключающий диапазон через «^» и пишем два диапазона, «1-3» и «a-c». Этим требованиям соответствует все, кроме первого сочетание.

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ (4/4)

Квантификаторы

Квантификаторы в регулярных выражениях обозначают количество повторений.

В примере слева регулярка начинается с символьного класса «цифра». Дальше идут фигурные скобки — это и есть квантификатор. «{3,5}» означает, что цифра должна повторяться от 3 до 5 раз. Затем идет класс «слово» и квантификатор «{3,}» — то есть всё, что относится к классу «слово», должно повторяться 3 раза и более. Этим требованиям соответствует только второе выражение.

У квантификаторов есть свои краткие обозначения. От 0 до 1 повторений — знак «?», от 1 и более — «+», от 0 и более — «*».

В задаче справа нам нужно найти все числа с двумя знаками после запятой. Регулярное выражение начинаем с «[\s^]» — это значит, что результат должен начинаться либо с пробела, либо с начала строки. «\d+» значит, что цифра должна повторяться 1 или более раз. Символ «\» перед точкой означает, что точка здесь экранированная, то есть не используется как служебный символ. После точки мы пишем символьный класс «цифра» и квантификатор «от 1 до 2», то есть цифра после точки должна повторяться от 1 до 2 раз. Заканчиваться результат должен или пробелом, или символом конца строки — «[\s" class="formula inline">]».

ПРЕДИКАТ EXISTS

Предикат exists определяется следующим синтаксическим правилом:

```
exists_predicate ::= EXISTS (query_expression)
```

Значением условия EXISTS (query_expression) является true в том и только в том случае, когда мощность таблицы-результата выражения запросов больше нуля, иначе значением условия является false.

ПРИМЕР Найти номера отделов, среди служащих которых имеются менеджеры проектов.

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE EXISTS
    (SELECT EMP.EMP_NO
    FROM EMP
    WHERE EMP.DEPT_NO = DEPT.DEPT_NO
        AND EXISTS
            (SELECT PRO.PRO_MNG
            FROM PRO
            WHERE PRO.PRO_MNG = EMP.EMP_NO));
```

ПРЕДИКАТ UNIQUE

Этот предикат позволяет сформулировать условие отсутствия дубликатов в результате запроса:

```
unique_predicate ::= UNIQUE (query_expression)
```

Результатом вычисления условия `UNIQUE (query_expression)` является `true` в том и только в том случае, когда в таблице-результате выражения запросов отсутствуют какие-либо две строки, одна из которых является дубликатом другой. В противном случае значение условия есть `false`.

ПРИМЕР Найти номера отделов, служащих которых можно различить по имени и дате рождения.

```
SELECT DEPT_NO
FROM DEPT
WHERE UNIQUE
    (SELECT EMP_NAME, EMP_BDATE
      FROM EMP
      WHERE EMP.DEPT_NO = DEPT.DEPT_NO);
```

ПРЕДИКАТ OVERLAPS

Не все СУБД поддерживают, в SQLite этого предиката нет.

```
(start1, end1) OVERLAPS (start2, end2)
(start1, length1) OVERLAPS (start2, length2)
```

This expression yields true when two time periods (defined by their endpoints) overlap, false when they do not overlap. The endpoints can be specified as pairs of dates, times, or time stamps; or as a date, time, or time stamp followed by an interval. When a pair of values is provided, either the start or the end can be written first; `OVERLAPS` automatically takes the earlier value of the pair as the start. Each time period is considered to represent the half-open interval `start <= time < end`, unless `start` and `end` are equal in which case it represents that single time instant. This means for instance that two time periods with only an endpoint in common do not overlap.

```
SELECT (DATE '2001-02-16', DATE '2001-12-21') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Result: true
SELECT (DATE '2001-02-16', INTERVAL '100 days') OVERLAPS
       (DATE '2001-10-30', DATE '2002-10-30');
Result: false
SELECT (DATE '2001-10-29', DATE '2001-10-30') OVERLAPS
       (DATE '2001-10-30', DATE '2001-10-31');
Result: false
SELECT (DATE '2001-10-30', DATE '2001-10-30') OVERLAPS
       (DATE '2001-10-30', DATE '2001-10-31');
Result: true
```

ПРЕДИКАТ СРАВНЕНИЯ С КВАНТОРОМ (1/2)

Этот предикат позволяет специфицировать квантифицированное сравнение строчного значения и определяется следующим синтаксическим правилом:

```
quantified_comparison_predicate ::= row_value_constructor
    comp_op { ALL | SOME | ANY } query_expression
```

Степень первого операнда должна быть такой же, как и степень таблицы-результата выражения запросов. Типы данных значений строки-операнда должны быть совместимы с типами данных соответствующих столбцов выражения запроса. Сравнение строк производится по тем же правилам, что и для предиката сравнения.

Обозначим через `x` строку-первый операнд, а через `S` – результат вычисления выражения запроса. Пусть `s` обозначает произвольную строку таблицы `S`. Тогда:

- условие `x comp_op ALL S` имеет значение `true` в том и только в том случае, когда `S` пусто, или значение условия `x comp_op s` равно `true` для каждой строки `s`, входящей в `S`. Условие `x comp_op ALL S` имеет значение `false` в том и только в том случае, когда значение предиката `x comp_op s` равно `false` хотя бы для одной строки `s`, входящей в `S`. В остальных случаях значение условия `x comp_op ALL S` равно `unknown`;
- условие `x comp_op SOME S` имеет значение `false` в том и только в том случае, когда `S` пусто, или значение условия `x comp_op s` равно `false` для каждой строки `s`, входящей в `S`. Условие `x comp_op SOME S` имеет значение `true` в том и только в том случае, когда значение предиката `x comp_op s` равно `true` хотя бы для одной строки `s`, входящей в `S`. В остальных случаях значение условия `x comp_op SOME S` равно `unknown`;
- условие `x comp_op ANY S` эквивалентно условию `x comp_op SOME S`.

ПРЕДИКАТ СРАВНЕНИЯ С КВАНТОРОМ (2/2)

ПРИМЕР 1

Найти номера служащих отдела номер 65, зарплата которых в этом отделе не является минимальной.

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
    AND EMP_SAL > SOME (SELECT EMP1.EMP_SAL
      FROM EMP EMP1
      WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

ПРИМЕР 2

Найти номера служащих отдела номер 65, зарплата которых в этом отделе является максимальной.

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
 AND EMP_SAL >= ALL(SELECT EMP1.EMP_SAL
    FROM EMP EMP1
    WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

ПРЕДИКАТ MATCH (1/2)

In MySQL, the MATCH() function performs a full-text search. It accepts a comma separated list of table columns to be searched. The table/s must have a FULLTEXT index before you can do a full-text search against them. You can create a FULLTEXT index when creating the table (using the CREATE TABLE statement), or you can use the ALTER TABLE statement or the CREATE INDEX statement if the table already exists.
By default, the search is case-insensitive. To perform a case-sensitive search, use a case-sensitive or binary collation for the indexed columns.

Syntax

The syntax for the `MATCH()` function goes like this:

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])
```

Where `col1,col2,...` is the comma-separated list of columns to search, and `expr` is the input string/expression.

The optional `search_modifier` argument allows you to specify the search type. It can be any of the following values:

- `IN NATURAL LANGUAGE MODE`
- `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION`
- `IN BOOLEAN MODE`
- `WITH QUERY EXPANSION`

The default mode is `IN NATURAL LANGUAGE MODE`.

Example 1 – Basic Usage

Here's an example of how to use this function:

```
SELECT AlbumId, AlbumName
FROM Albums
WHERE MATCH(AlbumName) AGAINST('cool');
```

Result:

```
+---------+--------------------+
| AlbumId | AlbumName          |
+---------+--------------------+
|       5 | Casualties of Cool |
+---------+--------------------+
```

ПРЕДИКАТ MATCH (2/2)

Полнотекстовый индекс: Full text index

Full text index – полнотекстовый специализированный индекс, для больших объемов текстовых данных, которые хранятся в столбцах строкового типа VARCHAR, TEXT. Для относительно небольших типов данных используется обычный индекс.

Суть работы алгоритма в том, что из каждой ячейки вытаскивается текст, разбивается на слова, и для каждого слова создается отдельная таблица индекса. Появляется связь конкретного слова и ячеек, в которых это слово собственно и встречается.

Если вы задавались вопросом, как же работают поисковые системы, то в них есть подобный механизм.

Полнотекстовый индекс может так же включать в себя одновременно несколько столбцов таблицы, в этом случае содержимое склеивается в одно единое, и создается полнотекстовый индекс, по алгоритму, который уже был описан выше. Кроме всего этого поддерживает морфологию языков, стоп слова, а так же перестановки в словосочетаниях. Так же есть минимальное значение длины слова, которое равно 4-м символам, т.е. если длина слова меньше 4-х, то слово не попадает в индекс.

Рассмотрим пример создания полнотекстового индекса с именем f_text_index для столбцов preview_text и detail_text.

```
1 | CREATE FULLTEXT INDEX `f_text_index` ON `table_name` (`preview_text`, `detail_text`);
```

После создания полнотекстового индекса, поиск можно производить с помощью оператора LIKE, это весьма может быть удобно, к примеру, если вам необходимо организовать поиск информации у себя на сайте.

ПРЕДИКАТ IS DISTINCT

Предикат позволяет проверить, являются ли две строки дубликатами. Условие определяется следующим синтаксическим правилом:

```
distinct_predicate ::= row_value_constructor IS DISTINCT FROM
    row_value_constructor
```

Строки-операнды должны быть одинаковой степени. Типы данных соответствующих значений строк-операндов должны быть совместимы.

Напомним, что две строки `s1` с именами столбцов `c1, c2, ..., cn` и `s2` с именами столбцов `d1, d2, ..., dn` считаются строками-дубликатами, если для каждого `i ( i = 1, 2, …, n )` либо `ci` и `di` не содержат `NULL`, и `(ci = di) = true`, либо и `ci`, и `di` содержат `NULL`. Значением условия `s1 IS DISTINCT FROM s2` является `true` в том и только в том случае, когда строки `s1` и `s2` не являются дубликатами. В противном случае значением условия является `false`.

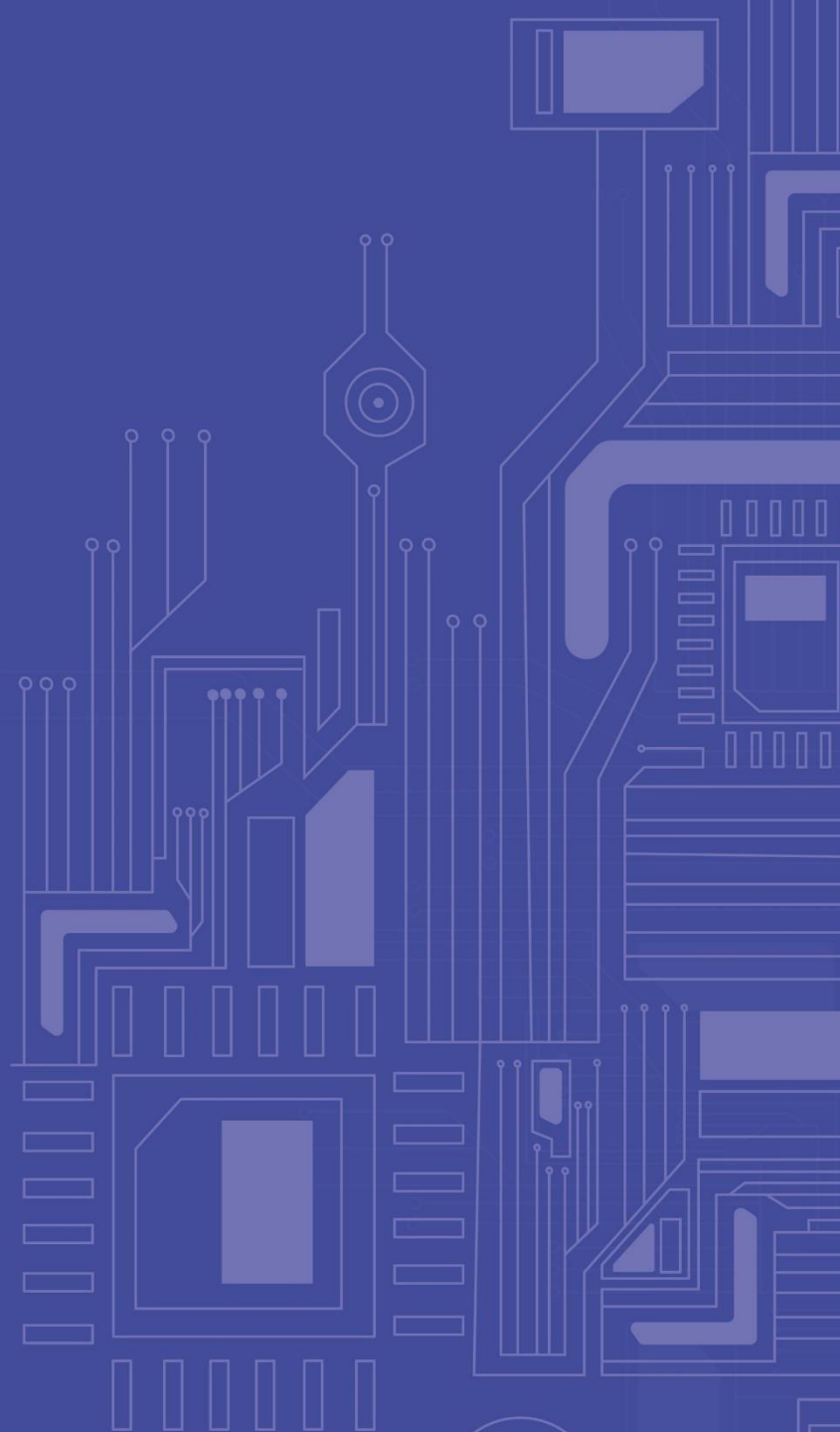
Заметим, что отрицательная форма условия – `IS NOT DISTINCT FROM` – в стандарте SQL не поддерживается. Вместо этого можно воспользоваться выражением `NOT s1 IS DISTINCT FROM s2`.

ПРИМЕР

Найти все пары номеров таких служащих отдела 65, которых нельзя различить по данным об имени и дате рождения.

```
SELECT EMP1.EMP_NO, EMP2.EMP_NO
FROM EMP EMP1, EMP EMP2
WHERE DEPT_NO = 65
    AND EMP1.EMP_NO <> EMP2.EMP_NO
    AND NOT ((EMP1.EMP_NAME, EMP1.EMP_BDATE) IS DISTINCT FROM
            (EMP2.EMP_NAME, EMP2.EMP_BDATE));
```

СОЕДИНЕНИЯ



ВНЕШНИЕ СОЕДИНЕНИЯ КОДДА

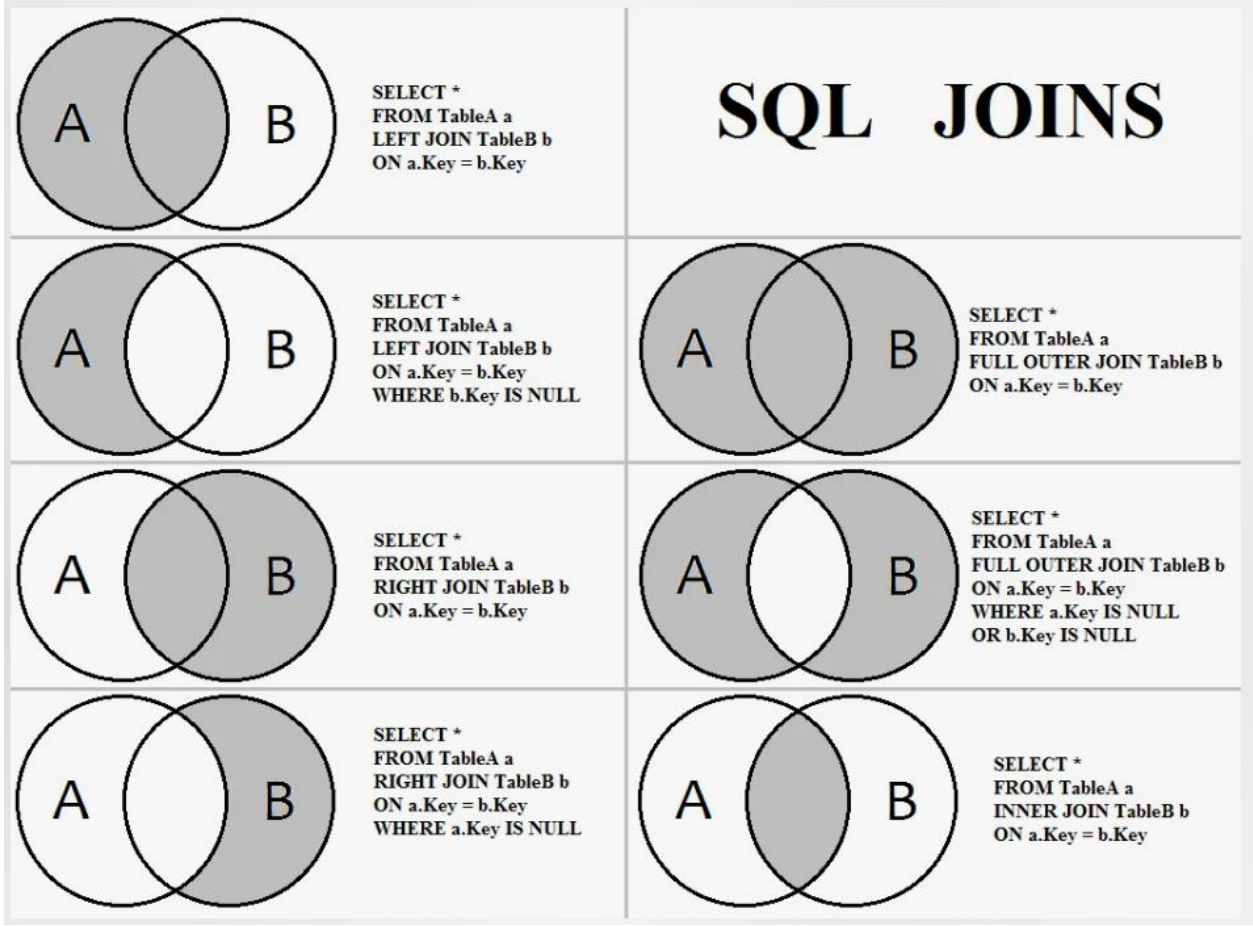
Пусть имеются отношения r_1 и r_2, совместимые относительно операции взятия расширенного декартова произведения. Пусть s является результатом операции r_1 `LEFT OUTER JOUN` r_2 `WHERE comp` (левое внешнее соединение r_1 и r_1 по условию `comp`). Тогда $Hs = Hr_1$ `union` Hr_2. Пусть $tr_1 \in Br_1$ и $tr_2 \in Br_2$. Тогда tr_1 `union` $tr_2 \in Bs$ в том и только в том случае, когда `comp` $(tr_1$ `union` $tr_2) =$ `true`. Если имеется кортеж $tr_1 \in Br_1$, для которого нет ни одного кортежа $tr_2 \in r_2$, такого, что `comp` $(tr_1$ `union` $tr_2) =$ `true`, то tr_1 `union` $tr_{2null} \in Bs$, где tr_{2null} – кортеж, соответствующий Hr_2, все значения которого являются неопределенными.

Пусть s является результатом операции r_1 `RIGHT OUTER JOUN` r_2 `WHERE comp` (правое внешнее соединение r_1 и r_2 по условию `comp`). Тогда $Hs = Hr_1$ `union` Hr_2. Пусть $tr_1 \in Br_1$ и $tr_2 \in Br_2$. Тогда tr_1 `union` $tr_2 \in Bs$ в том и только в том случае, когда `comp` $(tr_1$ `union` $tr_2) =$ `true`. Если имеется кортеж $tr_2 \in Br_2$, для которого нет ни одного такого кортежа $tr_1 \in Br_1$, что `comp` $(tr_1$ `union` $tr_2) =$ `true`, то tr_{1null} `union` $tr_2 \in Bs$, где tr_{1null} – кортеж, соответствующий Hr_1, все значения которого являются неопределенными.

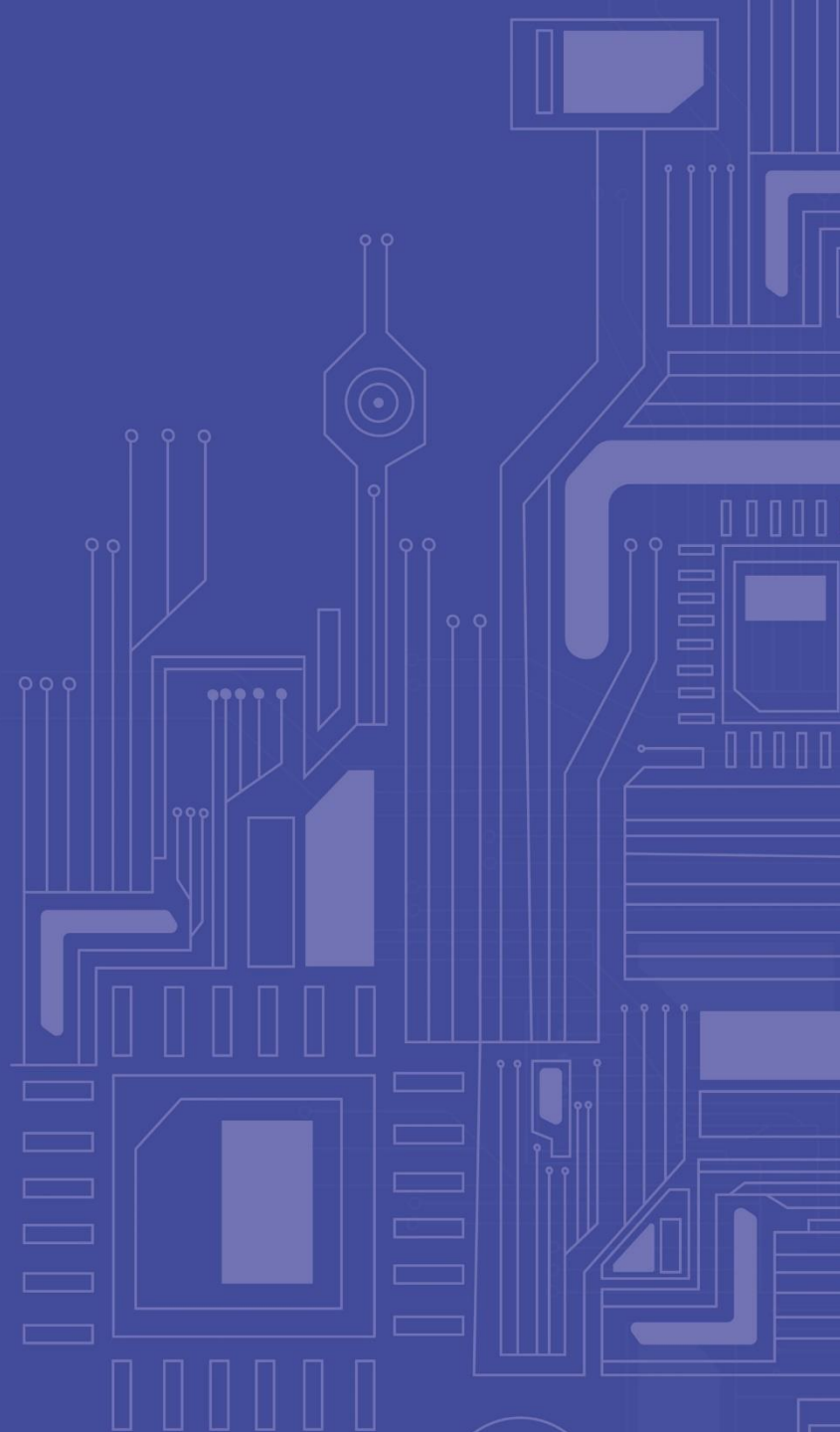
Наконец, пусть s является результатом операции r_1 `FULL OUTER JOUN` r_2 `WHERE comp` (полное внешнее соединение r_1 и r_2 по условию `comp`). Тогда $Hs = Hr_1$ `union` Hr_2. Пусть $tr_1 \in Br_1$ и $tr_2 \in Br_2$. Тогда tr_1 `union` $tr_2 \in Bs$ в том и только в том случае, когда `comp` $(tr_1$ `union` $tr_2) =$ `true`. Если имеется кортеж $tr_1 \in Br_1$, для которого нет ни одного кортежа $tr_2 \in Br_2$, такого, что `comp` $(tr_1$ `union` $tr_2) =$ `true`, то tr_1 `union` $tr_{2null} \in Bs$, где tr_{2null} – кортеж, соответствующий Hr_2, все значения которого являются неопределенными. Если имеется кортеж $tr_2 \in Br_2$, для которого нет ни одного кортежа $tr_1 \in Br_1$, такого, что `comp` $(tr_1$ `union` $tr_2) =$ `true`, то tr_{1null} `union` $tr_2 \in Bs$, где tr_{1null} – кортеж, соответствующий Hr_1, все значения которого являются неопределенными.

14 ВИДОВ СОЕДИНЕНИЙ В SQL

1. прямое соединение;
2. внутреннее соединение по условию;
3. внутреннее соединение по совпадению значений указанных одноименных столбцов;
4. естественное внутреннее соединение;
5. левое внешнее соединение по условию;
6. правое внешнее соединение по условию;
7. полное внешнее соединение по условию;
8. левое внешнее соединение по совпадению значений указанных одноименных столбцов;
9. правое внешнее соединение по совпадению значений указанных одноименных столбцов;
10. полное внешнее соединение по совпадению значений указанных одноименных столбцов;
11. естественное левое внешнее соединение;
12. естественное правое внешнее соединение;
13. естественное полное внешнее соединение;
14. соединение объединением.



АГРЕГАТНЫЕ ФУНКЦИИ



АГРЕГАТНЫЕ ФУНКЦИИ

> Агрегатная функция – это функция, которая выполняет вычисление на наборе значений и возвращает одиночное значение.

Общая структура запроса с агрегатной функцией

```
SELECT [литералы, агрегатные_функции, поля_группировки]
FROM имя_таблицы
GROUP BY поля_группировки;
```

Например, запрос с использованием агрегатной функции `AVG` может выглядеть так:

```
SELECT home_type, AVG(price) as avg_price FROM Rooms
GROUP BY home_type
```

home_type	avg_price
Private room	89.4286
Entire home/apt	148.6667
Shared room	40

Описание агрегатных функций

Функция	Описание
`SUM(поле_таблицы)`	Возвращает сумму значений
`AVG(поле_таблицы)`	Возвращает среднее значение
`COUNT(поле_таблицы)`	Возвращает количество записей
`MIN(поле_таблицы)`	Возвращает минимальное значение
`MAX(поле_таблицы)`	Возвращает максимальное значение

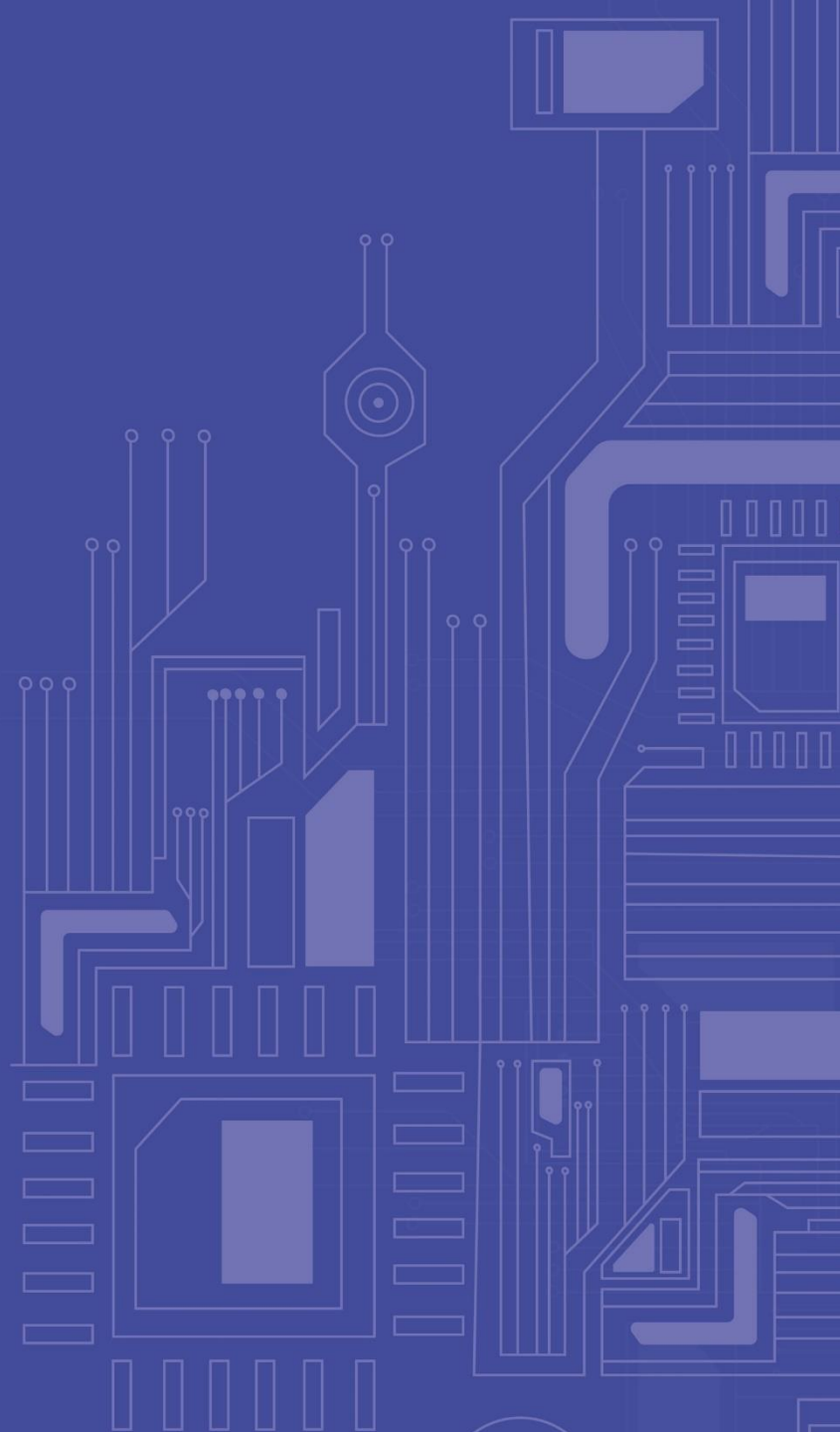
> Агрегатные функции применяются для значений, не равных `NULL`. Исключением является функция `COUNT(*)`.

АГРЕГАТНЫЕ ФУНКЦИИ И РЕЗУЛЬТАТЫ ЗАПРОСОВ

Разные случаи применения агрегатных функций

1. Табличное выражение R не является сгруппированной (нет GROUP BY или HAVING) таблицей, но в теле запроса есть агрегатная функция, тогда R автоматически рассматривается как сгруппированная (нельзя писать явно названия столбцов за пределами выражения агрегатной функции)
2. Табличное выражение R является по факту сгруппированной (раньше были GROUP BY или HAVING) таблицей, но в теле запроса есть агрегатная функция, тогда R автоматически рассматривается как сгруппированная (нельзя писать явно названия столбцов за пределами выражения агрегатной функции)
3. Табличное выражение R не является сгруппированной (есть GROUP BY или HAVING) таблицей, и в теле запроса есть агрегатная функция, тогда для R автоматически работают правила как для HAVING (можно писать явно названия столбцов за пределами выражения агрегатной функции)

ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ РАЗДЕЛА HAVING



ПРЕДИКАТЫ СРАВНЕНИЯ

ПРИМЕР 1 Найти номера отделов, в которых работает ровно 30 служащих.

```
SELECT DEPT_NO
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING COUNT(*) = 30;
```

<=>

```
SELECT DISTINCT DEPT_NO
FROM EMP
WHERE (SELECT COUNT (*)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO = EMP.DEPT_NO) = 30;
```

ПРИМЕР 2 Найти номера всех отделов, в которых средний размер зарплаты служащих превосходит 12000 руб.

```
SELECT DEPT_NO
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING AVG(EMP_SAL) > 12000.00;
```

<=>

```
SELECT DISTINCT DEPT_NO
FROM EMP
WHERE (SELECT AVG(EMP1.EMP_SAL)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO = EMP.DEPT_NO) > 12000.00;
```

ПРЕДИКАТ BETWEEN

ПРИМЕР

Найти номера отделов и минимальный и максимальный размер зарплаты служащих для отделов, в которых средний размер зарплаты служащих не меньше среднего размера зарплаты служащих во всей компании и не больше 30000 руб.

```
SELECT DEPT_NO, MIN(EMP_SAL), MAX(EMP_SAL)
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING AVG(EMP_SAL) BETWEEN
   (SELECT AVG(EMP_SAL)
   FROM EMP) AND 30000.00;
```

<=>

```
SELECT DISTINCT DEPT_NO, (SELECT MIN(EMP1.EMP_SAL)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO = EMP.DEPT_NO),
   (SELECT MAX(EMP1.EMP_SAL)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO = EMP.DEPT_NO)
FROM EMP
WHERE (SELECT AVG(EMP1.EMP_SAL)
     FROM EMP EMP1
     WHERE EMP1.DEPT_NO = EMP.DEPT_NO) BETWEEN
   (SELECT AVG(EMP_SAL)
     FROM EMP) AND 30000.00;
```

Как видно, отказ от использования раздела GROUP BY приводит к размножению однотипных подзапросов, строящих одну и ту же группу строк, над которой вычисляется агрегатная функция.

ПРЕДИКАТ NULL

ПРИМЕР

Найти номера и число служащих отделов, данные о руководителях которых не содержат номер отдела (конечно, в этом случае нас интересуют только те отделы, у которых имеется руководитель).

```
SELECT DEPT.DEPT_NO, COUNT(*)
FROM DEPT, EMP EMP1, EMP EMP2
WHERE DEPT.DEPT_NO = EMP2.DEPT_NO
 AND DEPT.DEPT_MNG = EMP1.EMP_NO
GROUP BY DEPT.DEPT_NO, EMP1.DEPT_NO
HAVING EMP1.DEPT_NO IS NULL;
```

<=>

```
SELECT DEPT.DEPT_NO, (SELECT COUNT(*)
   FROM EMP
   WHERE DEPT.DEPT_NO = EMP.DEPT_NO)
FROM DEPT, EMP
WHERE DEPT.DEPT_MNG = EMP.EMP_NO AND
      EMP.DEPT_NO IS NULL;
```

В этом случае, поскольку в запросе присутствует только один вызов агрегатной функции, формулировка без использования раздела GROUP BY оказывается более понятной и не менее эффективной (даже при следовании предписанной семантике выполнения оператора SELECT).

ПРЕДИКАТ IN

ПРИМЕР Найти номера отделов, в которых средний размер зарплаты служащих равен максимальному размеру зарплаты служащих какого-либо другого отдела.

```
SELECT DEPT.DEPT_NO
FROM DEPT, EMP
WHERE DEPT.DEPT_NO = EMP.DEPT_NO
GROUP BY DEPT.DEPT_NO
HAVING AVG(EMP.EMP_SAL) IN
   (SELECT MAX(EMP1.EMP_SAL)
     FROM EMP, DEPT DEPT1
     WHERE EMP.DEPT_NO = DEPT1.DEPT_NO
     AND DEPT1.DEPT_NO <> DEPT.DEPT_NO
     GROUP BY DEPT.DEPT_NO);
```

<=>

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE (SELECT AVG(EMP_SAL)
   FROM EMP
   WHERE EMP.DEPT_NO = DEPT.DEPT_NO) IN
   (SELECT MAX(EMP1.EMP_SAL)
   FROM EMP, DEPT DEPT1
   WHERE EMP.DEPT_NO = DEPT1.DEPT_NO
     AND DEPT1.DEPT_NO <> DEPT.DEPT_NO
   GROUP BY DEPT.DEPT_NO);
```

ПРЕДИКАТ LIKE

ПРИМЕР

Во всех отделах найти имена и число служащих, у которых в данном отделе имеются однофамильцы и фамилии которых начинаются со строки символов, изображающей фамилию руководителя отдела.

```
SELECT EMP_NAME, COUNT(*)
FROM EMP, DEPT
WHERE EMP.DEPT_NO = DEPT.DEPT_NO
GROUP BY DEPT.DEPT_NO, EMP_NAME
HAVING COUNT(*) > 1
   AND EMP.EMP_NAME LIKE (SELECT EMP1.EMP_NAME
     FROM EMP EMP1
     WHERE EMP1.EMP_NO = DEPT.DEPT_MNG) || '%';
```

<=>

```
SELECT EMP_NAME, (SELECT COUNT(*)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO = EMP.DEPT_NO
     AND EMP1.EMP_NAME = EMP.EMP_NAME
     AND EMP1.EMP_NO <> EMP.EMP_NO) + 1
FROM EMP
WHERE (SELECT COUNT(*)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO = EMP.DEPT_NO
     AND EMP1.EMP_NAME = EMP.EMP_NAME
     AND EMP1.EMP_NO <> EMP.EMP_NO) > 1
 AND EMP_NAME LIKE (SELECT EMP1.EMP_NAME
   FROM EMP EMP1, DEPT
   WHERE EMP.DEPT_NO = DEPT.DEPT_NO
   AND EMP1.EMP_NO = DEPT.DEPT_MNG) || '%';
```

ПРЕДИКАТ EXISTS

ПРИМЕР Найти номера отделов, в которых средний размер зарплаты служащих равен максимальному размеру зарплаты служащих какого-либо другого отдела.

```
SELECT DEPT.DEPT_NO
FROM DEPT, EMP
WHERE DEPT.DEPT_NO = EMP.DEPT_NO
GROUP BY DEPT.DEPT_NO
HAVING EXISTS (SELECT *
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO <> DEPT.DEPT_NO
   GROUP BY EMP1.DEPT_NO
   HAVING MAX (EMP1.EMP_SAL)= AVG (EMP.EMP_SAL));
```

<=>

```
SELECT DEPT.DEPT_NO
FROM DEPT
WHERE EXISTS (SELECT EMP.DEPT_NO
   FROM EMP
   WHERE EMP.DEPT_NO <> DEPT.DEPT_NO
   GROUP BY EMP.DEPT_NO
   HAVING MAX (EMP.EMP_SAL)=
     (SELECT AVG (EMP1.EMP_SAL)
       FROM EMP EMP1
       WHERE EMP1.DEPT_NO = DEPT.DEPT_NO));
```

ПРЕДИКАТ UNIQUE

ПРИМЕР Найти номера отделов и средний размер зарплаты служащих для таких отделов, где средний размер зарплаты служащих отличается от среднего размера зарплаты всех других отделов.

```
SELECT DEPT.DEPT_NO, AVG (EMP.EMP_SAL)
FROM DEPT, EMP
WHERE DEPT.DEPT_NO = EMP.DEPT_NO
GROUP BY DEPT.DEPT_NO
HAVING UNIQIUE (SELECT AVG (EMP1.EMP_SAL)
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO IS NOT NULL
   GROUP BY EMP1.DEPT_NO
   HAVING AVG (EMP1.EMP_SAL) = AVG (EMP.EMP_SAL));
```

<=>

```
SELECT DEPT.DEPT_NO, AVG (EMP.EMP_SAL)
FROM DEPT, EMP
WHERE DEPT.DEPT_NO = EMP.DEPT_NO
GROUP BY DEPT.DEPT_NO
HAVING NOT EXISTS (SELECT EMP1.DEPT_NO
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO <> DEPT.DEPT_NO
   GROUP BY EMP1.DEPT_NO
   HAVING AVG (EMP1.EMP_SAL)= AVG (EMP.EMP_SAL));
```

ПРЕДИКАТЫ СРАВНЕНИЯ С КВАНТОРОМ

ПРИМЕР 1 Найти номера отделов и средний возраст служащих для таких отделов, что найдется хотя бы один другой отдел, средний возраст служащих которого больше, чем в данном.

```
SELECT DEPT_NO, AVG (CURRENT_DATE - EMP_BDATE)
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING AVG (CURRENT_DATE - EMP_BDATE)< SOME
   (SELECT AVG (CURRENT_DATE - EMP1.EMP_BDATE)
     FROM EMP EMP1
     WHERE EMP1.DEPT_NO IS NOT NULL
     GROUP BY EMP1.DEPT_NO);
```

<=>

```
SELECT DEPT_NO, AVG (CURRENT_DATE - EMP_BDATE)
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING EXISTS (SELECT EMP1.DEPT_NO
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO IS NOT NULL
   GROUP BY EMP1.DEPT_NO
   HAVING AVG (CURRENT_DATE - EMP1.EMP_BDATE) >
     AVG (CURRENT_DATE - EMP.EMP_BDATE));
```

ПРИМЕР 2 Найти номера отделов и средний возраст служащих для отделов с минимальным средним возрастом служащих.

```
SELECT DEPT_NO, AVG (CURRENT_DATE - EMP_BDATE)
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING AVG (CURRENT_DATE - EMP_BDATE) <= ALL
   (SELECT AVG (CURRENT_DATE - EMP_BDATE)
     FROM EMP
     WHERE DEPT_NO IS NOT NULL
     GROUP BY DEPT_NO);
```

<=>

```
SELECT DEPT_NO, AVG (CURRENT_DATE - EMP_BDATE)
FROM EMP
WHERE DEPT_NO IS NOT NULL
GROUP BY DEPT_NO
HAVING NOT EXISTS (SELECT EMP1.DEPT_NO
   FROM EMP EMP1
   WHERE EMP1.DEPT_NO IS NOT NULL
   GROUP BY EMP1.DEPT_NO
   HAVING AVG (CURRENT_DATE - EMP1.EMP_BDATE) <
     AVG (CURRENT_DATE - EMP.EMP_BDATE));
```

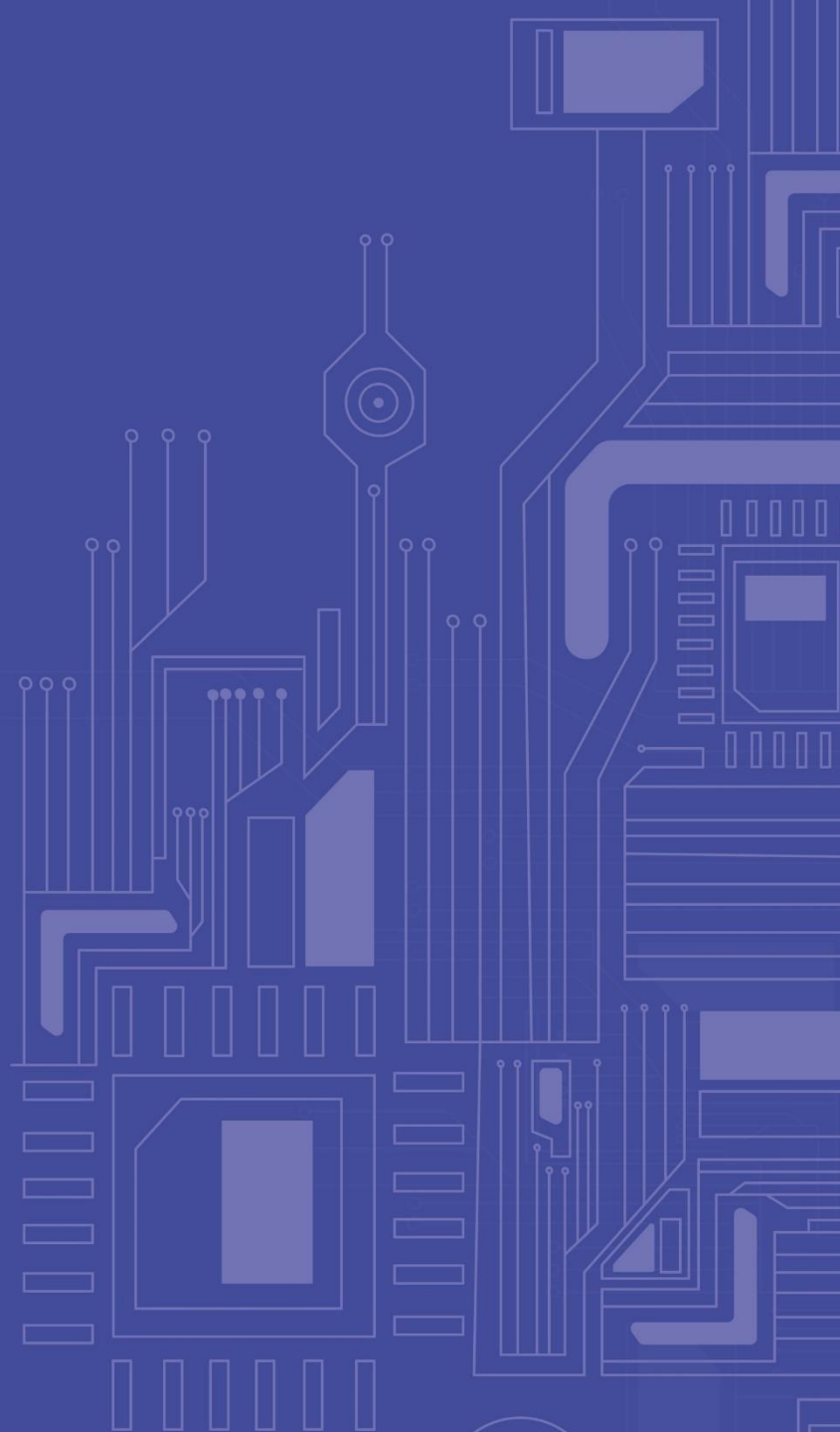
ПРЕДИКАТ DISTINCT

ПРИМЕР Найти номера отделов, которые можно отличить от любого другого отдела по дате рождения руководителя и среднему размеру зарплаты.

```
SELECT DEPT.DEPT_NO
FROM DEPT, EMP EMP1, EMP EMP2
WHERE DEPT.DEPT_NO = EMP1.DEPT_NO AND
   DEPT.DEPT_MNG = EMP2.EMP_NO
GROUP BY DEPT.DEPT_NO, EMP2.EMP_BDATE
HAVING (EMP2.EMP_BDATE, AVG (EMP1.EMP_SAL)) DISTINCT FROM
   (SELECT EMP2.EMP_BDATE, AVG (EMP1.EMP_SAL)
     FROM DEPT DEPT1, EMP EMP1, EMP EMP2
     WHERE DEPT1.DEPT_NO = EMP1.DEPT_NO AND
     DEPT1.DEPT_MNG = EMP2.EMP_NO AND
     DEPT1.DEPT_NO <> DEPT.DEPT_NO
     GROUP BY DEPT.DEPT_NO, EMP2.EMP_BDATE);
```


ВСЁ!

СЕМИНАР



ПРИМЕР НА BETWEEN В WHERE

Найти номера, имена и размер зарплаты служащих, получающих зарплату, размер которой не меньше средней зарплаты служащих своего отдела и не больше зарплаты руководителя отдела.

```
SELECT EMP_NO, EMP_NAME, EMP_SAL
FROM EMP
WHERE EMP_SAL BETWEEN
    (SELECT AVG(EMP1.EMP_SAL)
      FROM EMP EMP1
        WHERE EMP.DEPT_NO = EMP1.DEPT_NO)
          AND
    (SELECT EMP1.EMP_SAL
      FROM EMP EMP1
        WHERE EMP1.EMP_NO =
          (SELECT DEPT.DEPT_MNG
            FROM DEPT
            WHERE DEPT.DEPT_NO = EMP.DEPT_NO));
```

В этом запросе можно выделить три интересных момента. Во-первых, диапазон значений предиката BETWEEN задан двумя подзапросами, результатом каждого из которых является единственное значение. Первый подзапрос выдает единственное значение, поскольку в списке выборки содержится агрегатная функция (AVG) и отсутствует раздел GROUP BY, а второй – потому что в его разделе WHERE присутствует условие, задающее единственное значение первичного ключа. Во-вторых, в обоих подзапросах таблица EMP получает псевдоним EMP1 (в формулировке этого запроса мы старались использовать как можно меньше вспомогательных идентификаторов). Поскольку подзапросы выполняются независимо один от другого, использование общего имени не вызывает проблем. Наконец, в условии второго подзапроса присутствует более глубоко вложенный подзапрос, и в условии его раздела WHERE используется ссылка на столбец таблицы из самого внешнего раздела FROM.

ORM

Краткое упоминание:

Проектировать БД можно вручную на берегу, продумывая архитектуру, а можно на берегу писать ко приложения, а на основе объектов автоматически создать БД посредством Object Relation Manager.

Плюсы и минусы.

Вернемся подробнее на 13й лекции. Здесь нужно, чтоб «с вертолета» оглядеть практику создания БД.

JOIN