



Условные выражения

COALESCE

NULLIF

GREATEST и LEAST

CASE: простая форма

Поисковая форма CASE

- код выполняется, если заданное условие истинно
- аналоги в других языках программирования — if/else и switch

Команды условных выражений позволяют выполнять код, если некоторое заданное условие истинно.

COALESCE

```
COALESCE(значение [, ...])
```

- возвращает первый попавшийся аргумент, отличный от NULL.

Если же все аргументы равны NULL, результатом тоже будет NULL.

- используется при отображении данных для подстановки некоторого значения по умолчанию вместо значений NULL

COALESCE вычисляет только те аргументы, которые необходимы для получения результата.

Аргументы правее первого отличного от NULL аргумента не вычисляются.

💬 Эта функция соответствует стандарту SQL, а в некоторых других СУБД её аналоги называются NVL и IFNULL.

NULLIF

```
NULLIF(значение1, значение2)
```

- возвращает значение NULL, если значение1 и значение2 равны
- в противном случае возвращает значение1
- можно реализовать обратную операцию к COALESCE

```
SELECT NULLIF(value, <none>)
```

Если аргумент value равен (none), результатом выражения будет NULL, а в противном случае — значение аргумента value.

GREATEST и LEAST

```
GREATEST (значение [, ...])  
LEAST (значение [, ...])
```

- выбирают наибольшее или наименьшее значение из списка выражений. Все эти выражения должны приводиться к общему типу данных, который станет типом результата
- значения NULL в этом списке игнорируются
- выражение будет равно NULL, только если все его аргументы равны NULL

CASE: простая форма

```
CASE  
  WHEN значение THEN результат  
  WHEN ... ELSE результат  
END
```

Предложения CASE можно использовать там, где допускаются выражения (ORDER BY, GROUP BY). Каждое условие в нём представляет собой выражение, возвращающее результат типа boolean.

- если условие true, то результат и другие ветки не вычисляются
- если не выполняется ни одно из условий WHEN, значением CASE становится результат, записанный в предложении ELSE
- если предложение ELSE отсутствует, результатом выражения будет NULL

Типы данных всех выражений результатов должны приводиться к одному выходному типу, чтобы база данных понимала, как это выражение трактовать.

Пример

Нужно присвоить студентам категорию в зависимости от стипендии:

до 10000 — первая категория,

от 10000 до 20000 — вторая,

свыше 20000 — третья.

```
SELECT student id, name, studentship,  
       CASE WHEN studentship < 10000 THEN 1  
            WHEN studentship BETWEEN 10000 AND 20000 THEN 2 ELSE 3  
       END AS category  
FROM students;
```

Поисковая форма CASE

```
CASE выражение  
  WHEN значение THEN результат  
  WHEN ... ELSE результат  
END
```

- вычисляется первое выражение
- результат сравнивается со значениями в WHEN, пока не будет найдено равное ему
- если не нашлось, то возвращается результат предложения ELSE или NULL

👉 Эта форма больше похожа на оператор switch, существующий в языке C.



Агрегатные функции в простых выборках

COUNT

SUM(столбец)

MIN/MAX(столбец)

AVG(столбец)

Генерация последовательностей

DISTINCT в агрегациях

COUNT — подсчитывает количество строк

SUM — суммирует вместе значения в указанном столбце

MIN и MAX — возвращает минимальное и максимальное значение в указанном столбце

AVG — подсчитывает среднее по столбцу

В отличие от арифметических операций, которые допускают выполнение операций только для одной строки, агрегатные выполняют **операции для столбцов** и могут вовлекать огромные объемы данных.

При использовании агрегатных функций для простых выборок в select-списке нельзя указывать ничего кроме агрегатных функций.

COUNT

```
SELECT COUNT(*) FROM ...
```

Это агрегатная функция для подсчета количества строк.

☺ Наверное, это самая простая и одна из наиболее используемых агрегатных функций.

COUNT можно выполнять также со столбцом. В таком случае идет подсчет только ненулевых значений в столбце:

```
SELECT COUNT(столбец) FROM ...
```

Пример

Нужно написать запрос, который выводит количество ненулевых значений в поле low.

```
SELECT COUNT(low) AS low FROM stocks;
```

Важно дать осмысленное название для поля, так как название полей не наследуются. Например, в Postgres название этого поля скорее всего будет count.

SUM(столбец)

```
SELECT SUM(столбец) FROM ...
```

SUM считает сумму по столбцу.

В отличие от COUNT, который работает с любыми типами данных, SUM можно использовать только с числовыми типами.

📌 **Агрегатные функции не учитывают нулевые значения в столбце.**

В случае с NULL можно их рассматривать как 0.

Агрегатные функции агрегируют только по столбцам. Можно найти несколько сумм по нескольким столбцам, а потом произвести с этими суммами арифметические операции.

Пример

Нужно написать запрос, который найдет сумму дельт между наибольшей и наименьшей ценой.

 Ошибка:

```
select
```

```
    sum(high) — sum(low) as v1,
```

```
from stocks;
```

Этот вариант не учитывает нулевые значения в одном из столбцов. Такой запрос даст неверный результат, потому что дельта между неизвестностью и каким-то значением — это неизвестность.

Правильный ответ:

```
SELECT
    SUM(high-low) AS v2
FROM stocks;
```


При выполнении операции с NULL будет тоже NULL, а сумма его не учитывает.

MIN/MAX(столбец)

```
SELECT MIN(столбец) FROM ...
SELECT MAX(столбец) FROM ...
```

MIN/MAX используется, чтобы получить минимальное/максимальное значение по столбцу.

Эти функции похожи на COUNT тем, что могут использоваться не только по числовым полям. Главное условие — тип должен быть сравним.

 Если это множество, на котором не определена операция сравнения, найти минимальное и максимальное значение не получится.

AVG(столбец)

```
SELECT AVG(столбец) FROM ...
```

Функция AVG используется для вычисления среднего значения по столбцу.

AVG имеет ограничения:

- может использоваться только по числовым столбцам, как и SUM
- полностью игнорирует NULL, как и другие агрегатные функции

Когда это не нужно, применяются условные выражения.

Пример

Нужно найти средний объем дневных торгов.


```
SELECT
  AVG(volume) AS avg_volume
FROM stocks;
```

Генерация последовательностей

```
SELECT * FROM generate_series(start,end)
```

или

```
SELECT generate_series(start,end)
```

 Можно использовать даже без FROM:

Запрос `SELECT generate_series(start, end)` тоже будет работать.

Генерация случайных значений

```
SELECT RANDOM()
```

Это функция для получения случайных значений в диапазоне.

DISTINCT в агрегациях

```
SELECT COUNT(DISTINCT столбец) FROM ...
```

DISTINCT может использоваться при выполнении агрегаций — когда нужно учитывать **только уникальные значения**.

Часто применяется вместе с функцией COUNT.

Для MIN/MAX не применяется, поскольку MIN/MAX для одинаковых значений будет тем же, что и без применения DISTINCT.



GROUP BY

```
SELECT столбец, агрегаты FROM ...  
GROUP BY столбец
```

- Разбивает данные на группы
- Агрегаты работают по группам

Агрегатные функции, такие как **COUNT**, **AVG** и **SUM** агрегируют данные целой таблицы. В случае, когда нужно агрегировать таблицу по частям (например, подсчитать количество записей для каждого года), нужно выражение **GROUP BY**.

GROUP BY позволяет разбить данные на группы, которые могут быть агрегированы независимо друг от друга.

Пример

Нужно подсчитать количество записей для каждого года.

```
SELECT yr, COUNT(*) FROM (  
  SELECT  
    EXTRACT (year from date) AS yr  
  FROM stocks  
) AS stocks  
GROUP BY yr;
```

Связь GROUP BY и ORDER BY

📌 Нельзя использовать поле для упорядочивания, которого нет в группе, но можно сортировать по агрегатам. Ситуация аналогична DISTINCT.

👉 С помощью **GROUP BY** можно группировать по нескольким полям.

```
SELECT столбец1, столбец2, ..., агрегаты FROM ...  
GROUP BY столбец1, столбец2, ...
```

GROUP BY по номерам столбцов

```
SELECT столбец1, столбец2, ..., агрегаты FROM ...  
GROUP BY 1, 2, ...
```

💬 Как и в ORDER BY, можно указывать номера столбцов вместо колонок. Но так лучше делать, если у вас слишком большое количество колонок.