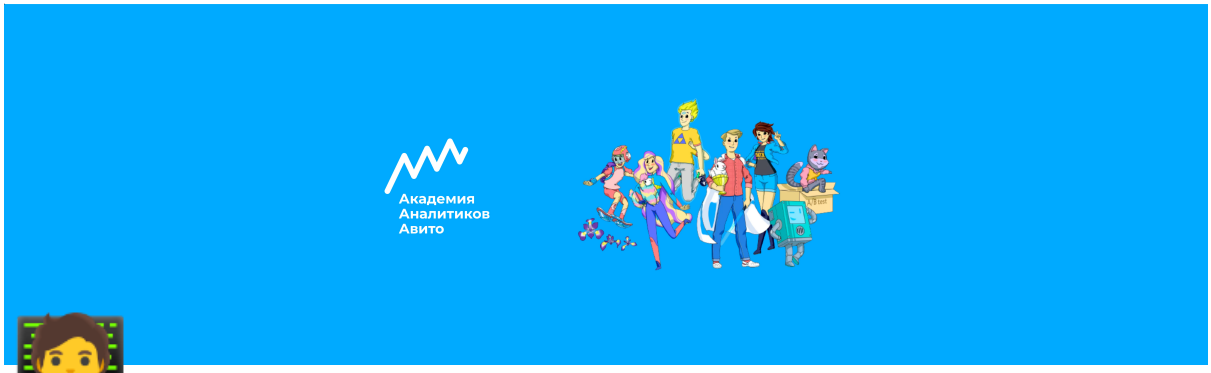


Сравнение табличных строк и массивов

ALL, ANY/SOME: квантификаторы с предикатами сравнения

```
<выражение> <оператор сравнения> ALL | { ANY | SOME } (<выражение массива>)
```

Квантификаторы ALL, ANY/SOME берут выражение, которое возвращает список строк, или массив, и начинают последовательно сравнивать элементы слева от оператора с элементами этого массива.

💻 Пример

Нужно проверить утверждения в таблице:

Сравнение	ANY/SOME	ALL
Результат выполнения подзапроса не содержит строк для сравнения	FALSE	TRUE
Результат сравнения конструктора значений строки с каждой строкой из набора строк, полученных как подзапрос, равен TRUE	TRUE	TRUE
Результат сравнения конструктора значений строки с каждой строкой из набора строк, полученных как подзапрос, равен FALSE	FALSE	FALSE
Хотя бы один из результатов сравнения конструктора значений строки со строкой из набора строк, полученных как подзапрос, равен TRUE	TRUE	UNKNOWN
Ни один из результатов сравнения конструктора значений строки со строкой из набора строк, полученных как подзапрос, не равен TRUE	UNKNOWN	UNKNOWN

Можно применить такой запрос, чтобы определить истинность или ложность предиката. Когда истина оказывается в WHERE — запрос возвращает ту строку, для которой предикат истинный.

```
SELECT true WHERE 10 <= ALL (VALUES (1), (2), (3))
```

Предикат EXISTS

```
[NOT] EXISTS (<подзапрос>)
```

Предикат EXISTS принимает значение TRUE, если подзапрос содержит любое количество строк, иначе его значение равно FALSE. Для NOT EXISTS все наоборот.

📌EXISTS никогда не принимает значение UNKNOWN.

👉**Сравнения = и <> отличаются от других**

С этими операторами две строки считаются:

- равными, если все их соответствующие поля не равны NULL и равны между собой,
- неравными, если какие-либо соответствующие поля не равны между собой, и ни одно поле не равно NULL

В противном случае результатом сравнения будет неопределённость (NULL).

```
select
  (1, 2, 3) = (1, 2, 3), --true
  (1, 2, 3) = (1, 2, 4), -- false
  (1, 2, 3) = (1, 2, null), --null
  (1, 2, null) = (1, 2, null) --null
```

Связанные подзапросы

```
SELECT …, <связанный запрос>, … FROM …
WHERE (<связанный подзапрос>)
```

Связанный подзапрос — это подзапрос, который содержит **ссылку на столбцы** из включающего его запроса (назовем его основным).

Связанный подзапрос будет выполняться для каждой строки основного запроса, так как значения столбцов основного запроса будут меняться.

💬Связанные подзапросы позволяют иногда очень кратко написать запросы, которые могут выглядеть громоздко при использовании других языковых средств.

Полусоединение (SEMI-JOIN)

SEMI [пер. «половина»]

В реляционной алгебре существует операция полуобъединения (semi join), которая не имеет синтаксического представления в SQL.

💬Если бы синтаксис для данной операции существовал, вероятно, он имел бы следующий вид: LEFT SEMI JOIN и RIGHT SEMI JOIN. Такой синтаксис реализован в Cloudera Impala.

В SQL используются два варианта альтернативного синтаксиса, чтобы реализовать операцию «SEMI» JOIN:

- EXISTS
- IN

Альтернативный синтаксис: EXISTS

💻 Пример

Нужно извлечь всех работников, для которых существует (exists) поездка, то есть работников, ездивших хотя бы в одну поездку.

```
SELECT *
FROM employee e
WHERE EXISTS (
  SELECT *
  FROM employee_trip et
  WHERE e.id = et.employee_id
)
```

Код, реализующий SEMI JOIN, помещен в предложении WHERE.

В результате этого запроса каждый работник будет указан максимум один раз.

💬 Несмотря на то, что в данном синтаксисе отсутствует ключевое слово JOIN, большинство СУБД способны распознать, что данный запрос выполняет именно SEMI JOIN и особым образом его реализует на низком уровне.

Соответственно, применение SEMI JOIN вместо INNER JOIN для решения поставленной задачи обеспечивает преимущество в производительности.

📌После того, как найдено **первое совпадение**, СУБД не будет искать другие совпадения.

Альтернативный синтаксис: IN

💻 Пример

Полностью аналогичный запрос с применением синтаксиса IN.

```
SELECT *
FROM employee e
WHERE id IN (
  SELECT employee_id
  FROM employee_trip
)
```

Антисоединение (ANTI-JOIN)

Операция «ANTI» JOIN является противоположностью операции «SEMI» JOIN. Она тоже не имеет специального синтаксиса в SQL, но реализуется с помощью **NOT EXISTS**.

NOT EXISTS — альтернативный синтаксис ANTI-JOIN

💻 Пример

```
SELECT *
FROM employee e
WHERE NOT EXISTS (
  SELECT *
  FROM employee_trip et
  WHERE e.id = et.employee_id
)
```

NOT IN

❌ Синтаксисы на основе NOT EXISTS и NOT IN не эквивалентны. Это связано со спецификой NULL-значений.

❗ Не стоит использовать предикат NOT IN в SQL,

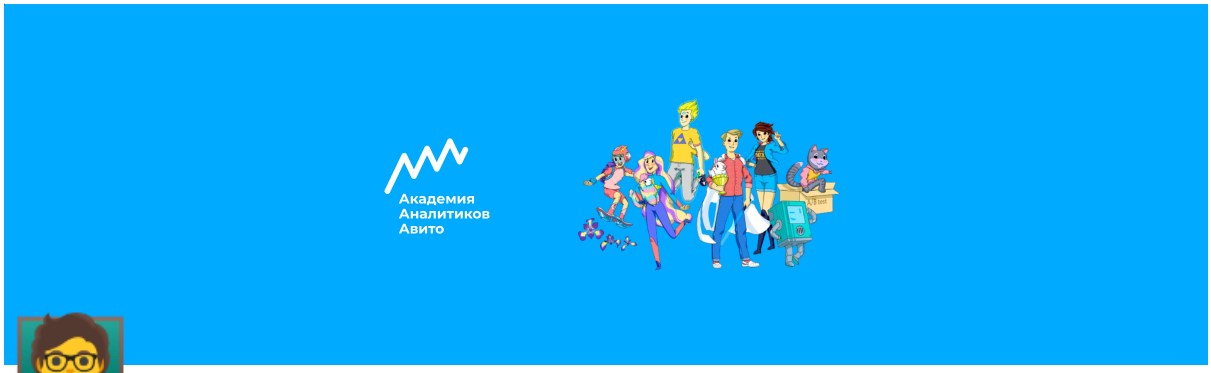
за исключением тех случаев, когда вы указали в нем константные,

не содержащие NULL значения.

LEFT JOIN / IS NULL: альтернативный синтаксис

Это корректный синтаксис, но он не выражает намерение выполнить ANTI JOIN.

💬Такой запрос будет медленнее аналогов, поскольку оптимизатор СУБД не сможет распознать, что программист хочет выполнить именно ANTI JOIN — и поэтому не подберет более быстрый алгоритм.



Ограничение выборки

Ограничение выборки

Ограничение выборки часто помогает провалидировать небольшие теории, которые не требуют всего набора данных.

👉Ограничения выборки можно использовать как простой способ выбора сэмпла данных — это экономит время ожидания ответа.

💻 Пример

```
SELECT * FROM companies LIMIT 10;
```

Для ограничения количества записей в конец запроса добавляется выражение LIMIT с необходимым количеством строк.

Порядок выполнения частей SELECT

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT

LIMIT будет применяться к результату агрегатной функции, которая без GROUP BY всегда вернет только одну запись.

❗ База данных не знает, какой порядок имеется в виду,

если не указана однозначная сортировка.

Детерминированность результата не гарантируется, если предложение ORDER BY не диктует выбор определённого подмножества. Планировщик запроса учитывает ограничение LIMIT, строя план выполнения запроса. Вероятнее всего, планы (а значит, и порядок строк) будут меняться при разных LIMIT.

📌**Если нужна идемпотентность, нужно включать сортировку.**

Идемпотентность — это свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при первом.

Предложение LIMIT

В общем виде предложение LIMIT в Potgres имеет такой вид:

```
LIMIT { число | ALL }
OFFSET начало
```

Оно состоит из двух независимых вложенных предложений.

Здесь «число» определяет максимальное количество строк, которое должно быть выдано, тогда как «начало» определяет, сколько строк нужно пропустить, прежде чем начать выдавать строки. Когда указаны оба значения, сначала строки пропускаются в количестве, заданном значением «начало», а затем следующие строки выдаются в количестве, не превышающем значения «число».

💻 Пример

```
SELECT id
FROM companies
ORDER BY id
OFFSET 10
LIMIT 100;
```

Альтернативный синтаксис для ограничения

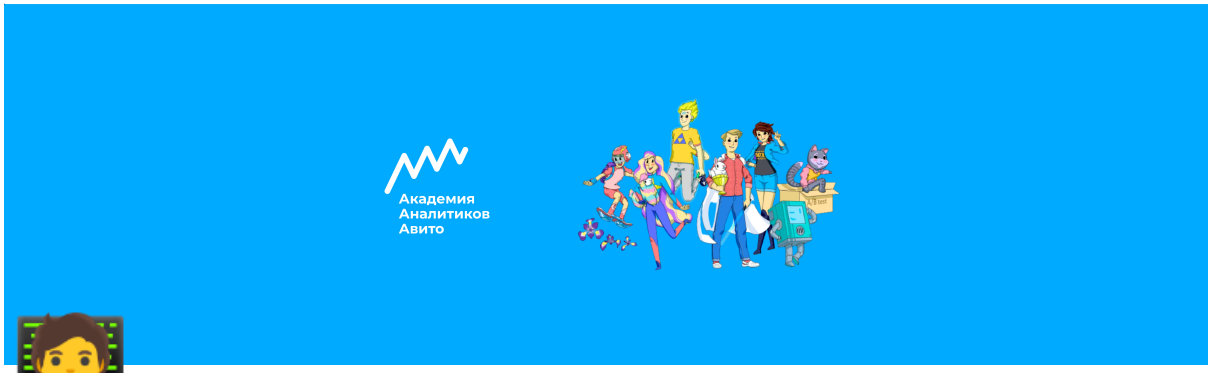
```
OFFSET начало { ROW | ROWS }
FETCH { FIRST | NEXT } [ число ] { ROW | ROWS } ONLY
```

В этом синтаксисе слова ROW и ROWS, а также FIRST и NEXT являются незначащими и не влияют на поведение этих предложений.

Синтаксис для ограничения выборки был введен в стандарте SQL:2008 и похож на DB2.

В других базах данных может отличаться:

DB2	select * from table fetch first 10 rows only
Informix	select first 10 * from table
MS SQL and Access	select top 10 with ties * from table
MySQL and PostgreSQL	select * from table limit 10
Oracle	select * from (select * from table) where rownum <= 10
DB2	select * from table fetch first 10 rows only



Связанные подзапросы и JOIN

LATERAL JOIN

```
A JOIN LATERAL (<связанный подзапрос>) ON …
```

LATERAL [пер. «боковой»]

📌LATERAL JOIN позволяет подзапросам ссылаться на предшествующие во FROM объекты.

Для каждой строчки из объекта FROM, на которые существуют кросс-ссылки, зависимая часть LATERAL JOIN выполняется каждый раз. Результирующие строки этого процесса соединяются как обычно.

💻 Пример

Даны таблицы lateral_table1(rowcount) и lateral_table2(id, value). Нужно написать запрос для выбора rowcount строк из lateral_table2, отсортировав результат по id. Для решения нужно использовать

```
SELECT id, value
FROM lateral_table1 t1,
LATERAL (
  SELECT *
  FROM lateral_table2
  LIMIT t1.rowcount
) t2
ORDER BY id
```

💬В этом случае значение в LIMIT не переменное, как может показаться на первый взгляд.

Для каждой строчки из объекта FROM, на которые существуют кросс-ссылки, зависимая часть LATERAL JOIN выполняется каждый раз. Это означает, что для запроса t2 значение в LIMIT подставляется еще до начала его выполнения — то есть является константой, а не параметром.