ClickHouse для инженеров и архитекторов БД

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Джоины и агрегации

Александра Гроховская

Senior Data Analyst / Team Lead
Ph.D.

*Преподаватель курса **ClickHouse для инженеров и архитекторов БД** в OTUS*

LinkedIn

Правила вебинара

Активно
участвуем

Задаем вопрос
в чат или голосом

Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения

Индивидуально

Время, необходимое
на активность

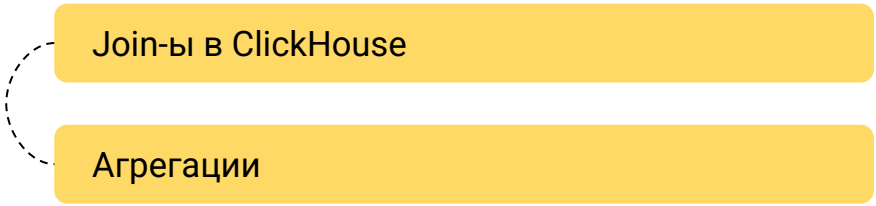
Пишем в чат

Говорим голосом

Документ

Ответьте себе или
задайте вопрос

Маршрут вебинара



Цели вебинара

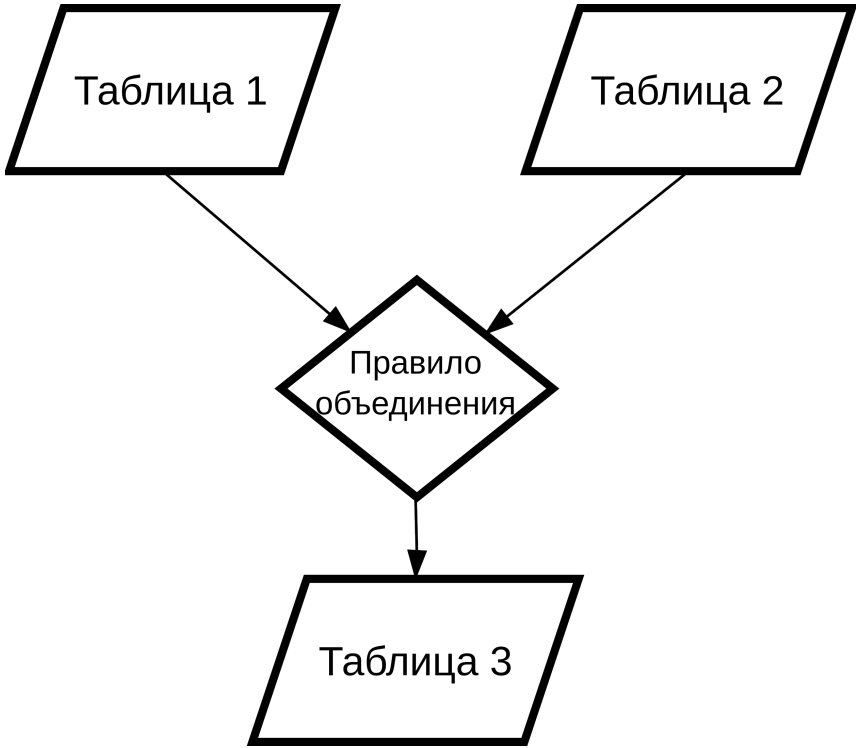
К концу занятия вы сможете

1. Изучить базовые и дополнительные типы соединений, доступные в ClickHouse;
2. Узнать о продвинутых методах агрегации и агрегатных функциях.

JOIN в ClickHouse

Что такое JOIN?

JOIN – это одна из самых популярных команд в SQL. Она позволяет объединять информацию из разных таблиц по определенным параметрам и получать нужные данные.



Джоины в ClickHouse

1. Базовые соединения
2. Дополнительные соединения

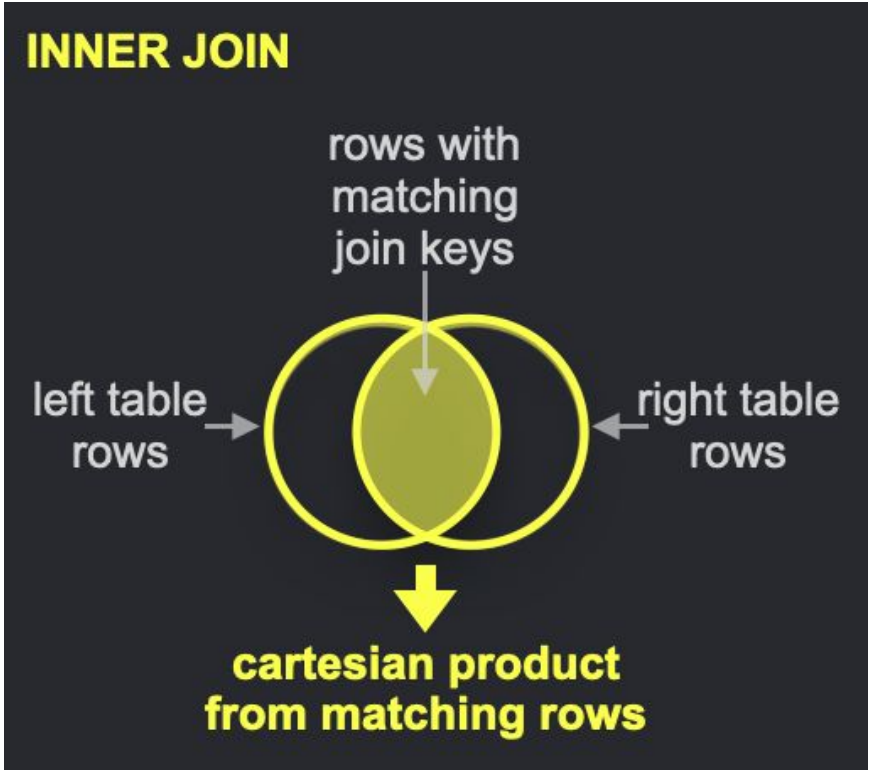
Базовые соединения

Базовые джоины в ClickHouse

- INNER JOIN
- LEFT OUTER JOIN
- RIGHT OUTER JOIN
- FULL OUTER JOIN
- CROSS JOIN

INNER JOIN

INNER JOIN – тип джоина, который возвращает данные, совпадающие по ключам соединения.



INNER JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

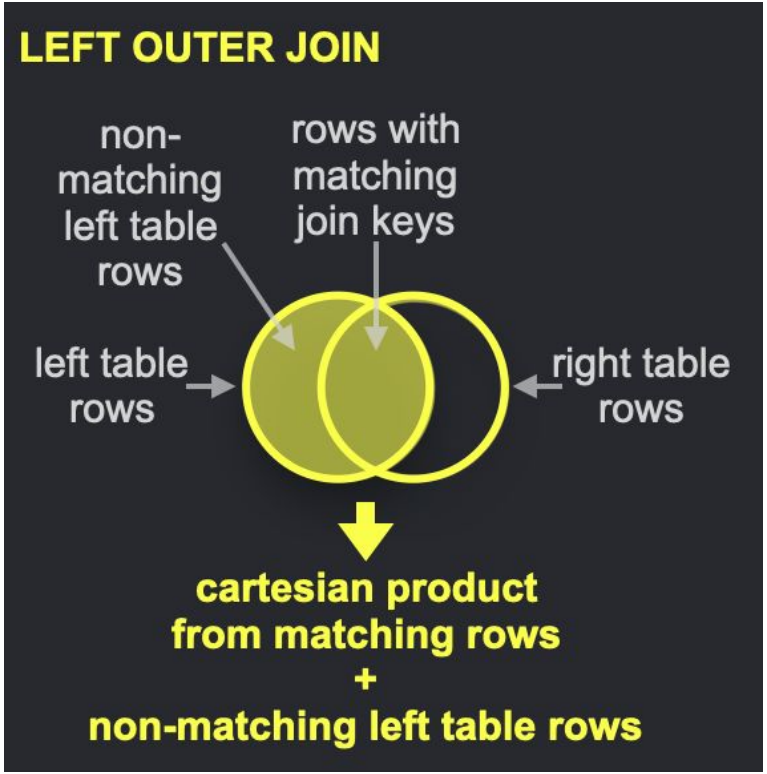
```
SELECT * FROM TABLE_1 JOIN TABLE_2 ON TABLE_1.id = TABLE_2.id
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:06:00	50	0	2020-01-01 10:05:05	purchase

LEFT OUTER JOIN

LEFT OUTER JOIN – тип джоина, который возвращает данные из левой таблицы и совпадающие по ключам соединения. Если не будет ни одного совпадения с правой таблицей, будут выведены данные из левой таблицы а поля правой будут иметь Null значения



LEFT OUTER JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

```
SELECT * FROM TABLE_1 LEFT JOIN TABLE_2 USING(ID)
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:06:00	50	0	2020-01-01 10:05:05	purchase
1	2020-01-01 12:01:07	12	0	1970-01-01 00:00:00	

LEFT OUTER JOIN

```
--создание таблицы с Nullable()
CREATE TABLE table2
(
    Nullable(id) UInt64,
    Nullable(date) DateTime,
    Nullable(description) String
)
ENGINE = Log
```

LEFT OUTER JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

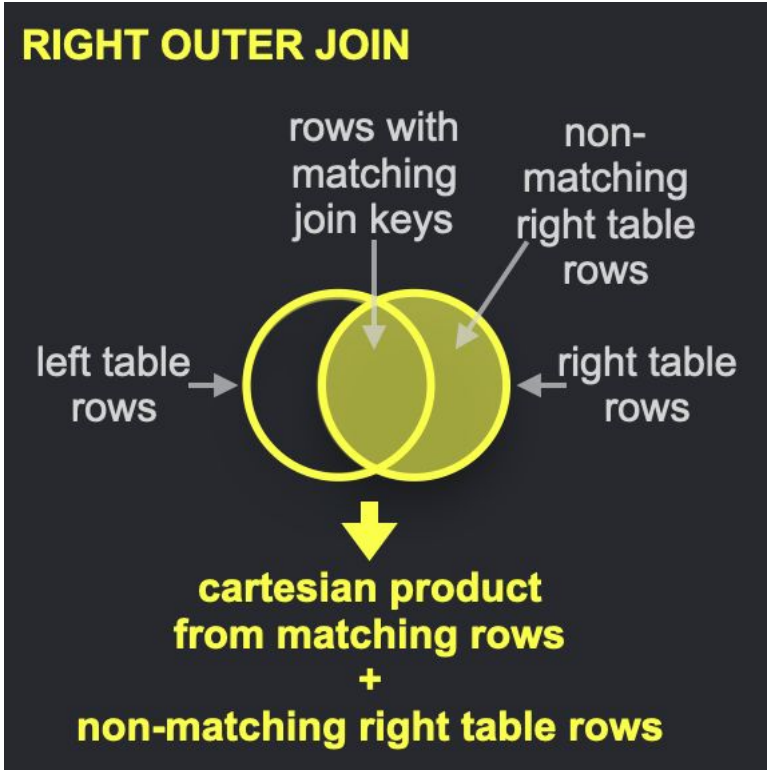
```
SELECT * FROM TABLE_1 LEFT JOIN TABLE_2 USING(ID)
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:06:00	50	0	2020-01-01 10:05:05	purchase
1	2020-01-01 12:01:07	12	NULL	NULL	NULL

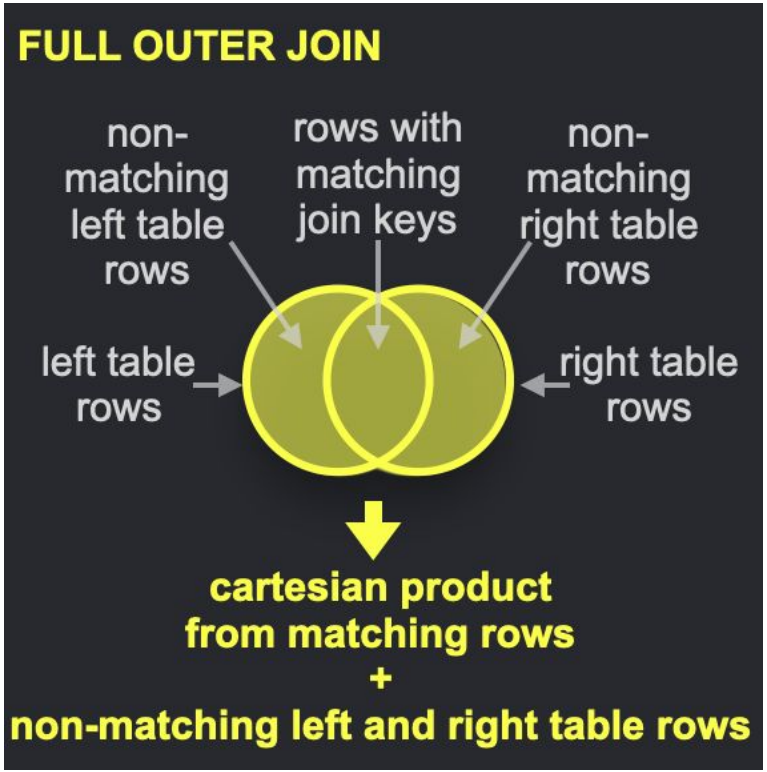
RIGHT OUTER JOIN

RIGHT JOIN – тип джоина, который возвращает данные из правой таблицы и совпадающие по ключам соединения. Если не будет ни одного совпадения с правой таблицей, будут выведены данные из левой таблицы а поля правой будут иметь Null значения



FULL OUTER JOIN

FULL JOIN – тип джоина, который объединяет в себе LEFT и RIGHT OUTER JOIN. Результат будет возвращать значения из обеих таблиц, а в случае отсутствия совпадений, отсутствующие поля будут содержать значение Null



FULL OUTER JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

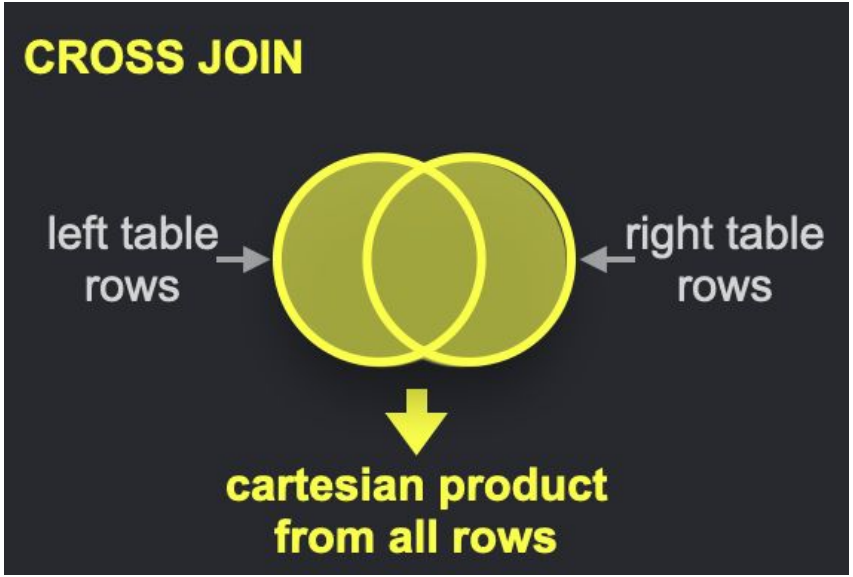
```
SELECT * FROM TABLE_1 FULL JOIN TABLE_2 ON TABLE_1.ID = TABLE_2.ID
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:06:00	50	0	2020-01-01 10:05:05	purchase
1	2020-01-01 12:01:07	12	0	1970-01-01 00:00:00	
0	1970-01-01 00:00:00	0	3	2020-01-01 12:01:07	purchase

CROSS JOIN

CROSS JOIN – тип джоина, который создает полное декартово произведение двух таблиц без учета ключей соединения. Каждая строка левой таблицы объединяется с каждой строкой правой таблицы.



CROSS JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

```
SELECT * FROM TABLE_1 CROSS JOIN TABLE_2
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:06:00	50	0	2020-01-01 10:05:05	purchase
0	2020-01-01 10:06:00	50	3	2020-01-01 12:01:07	purchase
1	2020-01-01 12:01:07	12	0	2020-01-01 10:00:05	cancel
1	2020-01-01 12:01:07	12	0	2020-01-01 10:05:05	purchase
1	2020-01-01 12:01:07	12	3	2020-01-01 12:01:07	purchase

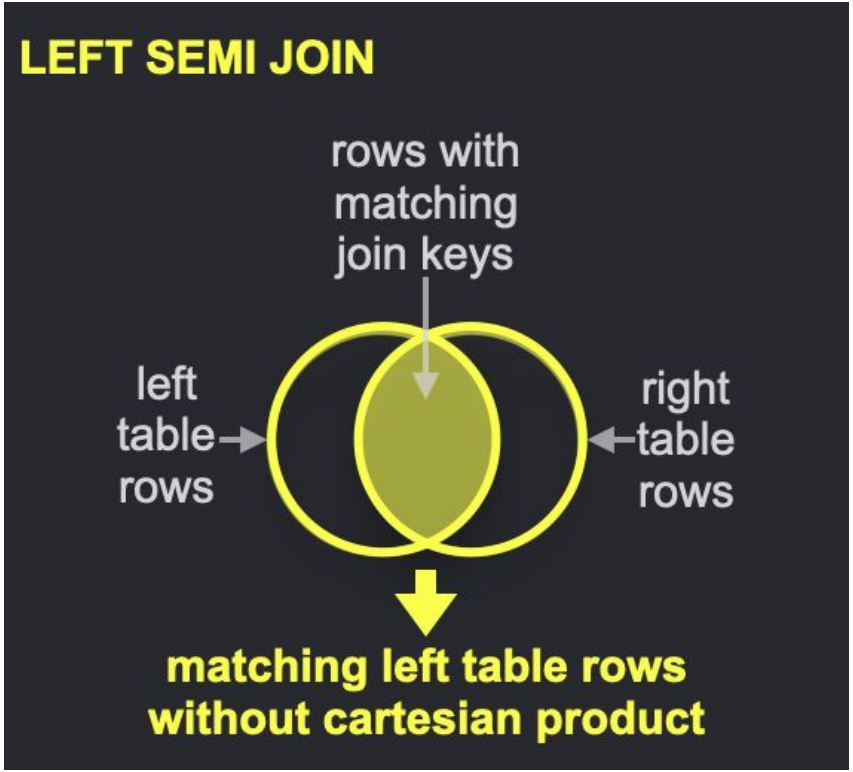
Дополнительные соединения

Дополнительные джоины в ClickHouse

- (LEFT / RIGHT) SEMI JOIN
- (LEFT / RIGHT) ANTI JOIN
- (LEFT / RIGHT / INNER) ANY JOIN
- ASOF JOIN
- PASTE JOIN

LEFT / RIGHT SEMI JOIN

SEMI JOIN – тип джоина, который возвращает значения для всех строк главной таблицы, имеющих хотя бы одно совпадение в другой таблице. Возвращается только первое найденное совпадение



LEFT / RIGHT SEMI JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

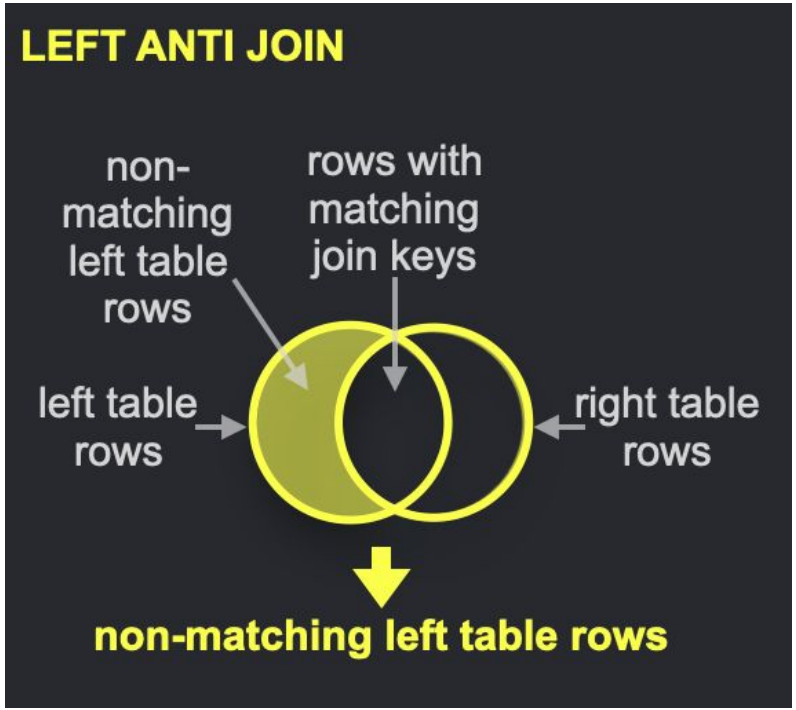
```
SELECT * FROM TABLE_1 LEFT SEMI JOIN TABLE_2 ON TABLE_1.ID = TABLE_2.ID
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel

LEFT / RIGHT ANTI JOIN

LEFT ANTI JOIN – тип джоина, который возвращает значения для всех несовпадающих строк из левой таблицы. Аналогично, RIGHT ANTI JOIN возвращает значения столбцов для всех несовпадающих правых строк таблицы.



LEFT / RIGHT ANTI JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

```
SELECT * FROM TABLE_1 LEFT ANTI JOIN TABLE_2 ON TABLE_1.ID = TABLE_2.ID
```

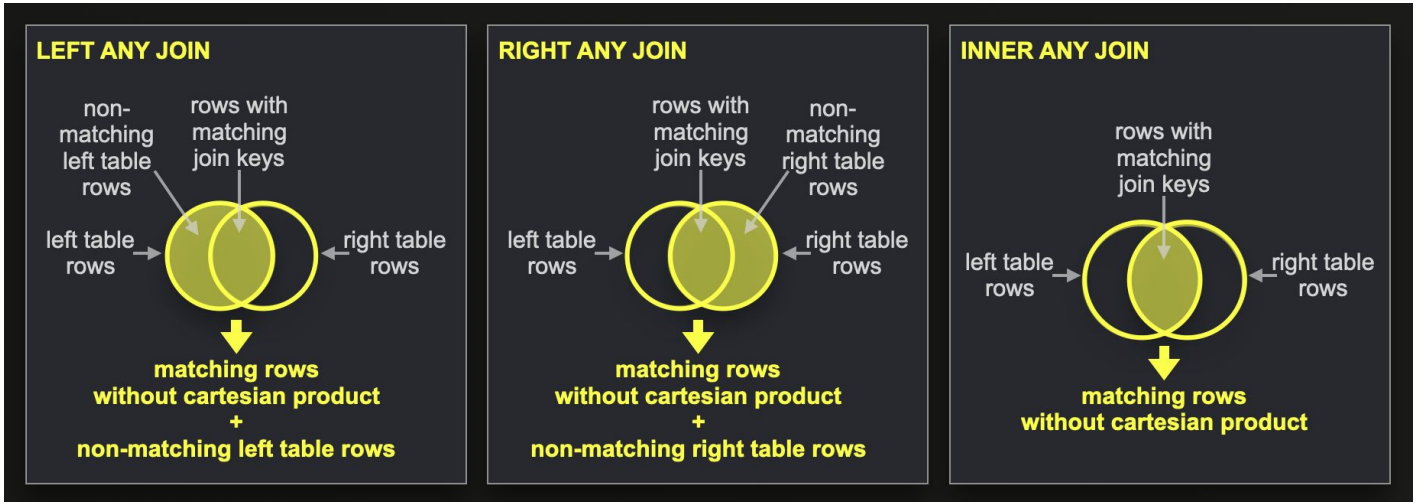

TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
1	2020-01-01 12:01:07	12	0	1970-01-01 00:00:00	

LEFT / RIGHT / INNER ANY JOIN

LEFT ANY JOIN - тип джоина, который работает как LEFT OUTER JOIN, но с отключенным декартовым произведением.

INNER ANY JOIN — это INNER JOIN с отключенным декартовым произведением.



LEFT / RIGHT / INNER ANY JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

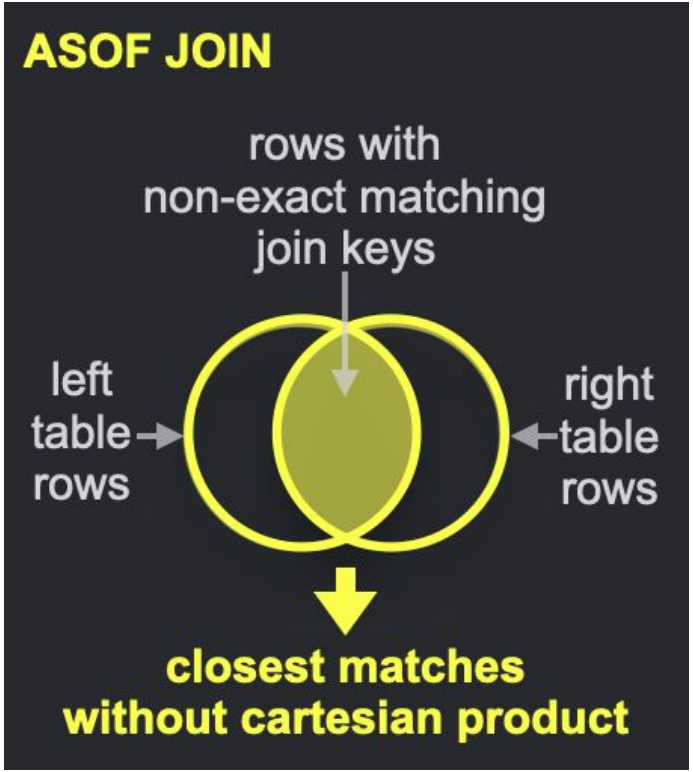
```
SELECT * FROM TABLE_1 LEFT ANY JOIN TABLE_2 ON TABLE_1.ID = TABLE_2.ID
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
1	2020-01-01 12:01:07	12	0	1970-01-01 00:00:00	

ASOF JOIN

ASOF JOIN – уникальный джоин, который предоставляет возможности неточного сопоставления. Если строка не имеет точного соответствия, то вместо нее в качестве совпадения используется ближайшее совпадающее значение. Декартовое произведение отсутствует.



ASOF JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

```
SELECT * FROM TABLE_1 ASOF LEFT JOIN TABLE_2 ON TABLE_1.date = TABLE_2.date
```


TABLE_3

id	date	value	TABLE_2.description
0	2020-01-01 10:06:00	50	purchase

PASTE JOIN

Результатом PASTE JOIN является таблица, содержащая все столбцы из левого подзапроса, а затем все столбцы из правого подзапроса. Строки сопоставляются на основе их позиций в исходных таблицах (порядок следования строк должен быть определен). Если подзапросы возвращают разное количество строк, лишние строки будут вырезаны.

PASTE JOIN

TABLE_1

id	date	value
0	2020-01-01 10:06:00	50
1	2020-01-01 12:01:07	12

TABLE_2

id	date	description
0	2020-01-01 10:00:05	cancel
0	2020-01-01 10:05:05	purchase
3	2020-01-01 12:01:07	purchase

```
SELECT * FROM TABLE_1 PASTE JOIN TABLE_2
```


TABLE_3

id	date	value	TABLE_2.id	TABLE_2.date	TABLE_2.description
0	2020-01-01 10:06:00	50	0	2020-01-01 10:00:05	cancel
1	2020-01-01 12:01:07	12	0	2020-01-01 10:05:05	purchase

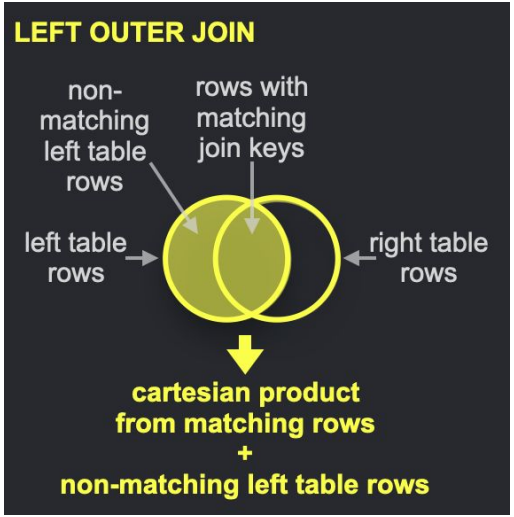
Вопросы?

Ставим “+”,
если вопросы есть

Ставим “-”,
если вопросов нет

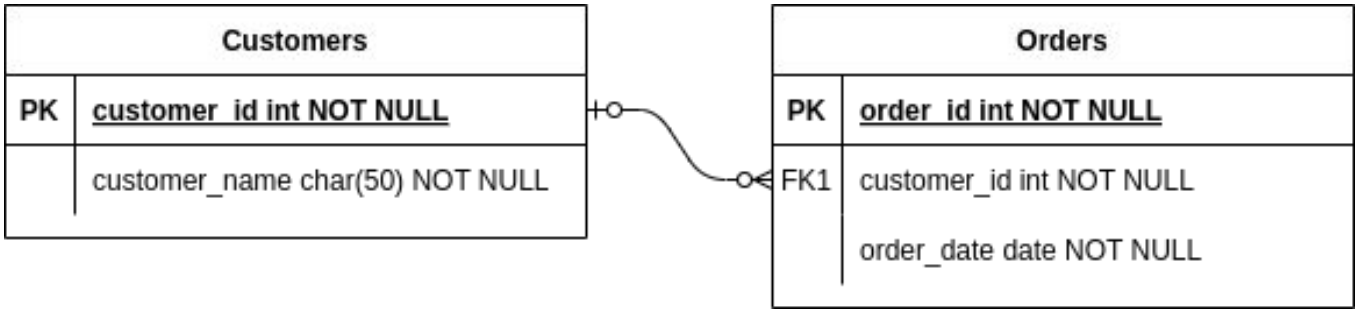
Пришлите в чат ответ, какое max и min количество строк может быть в результирующей таблице?

Дано 2 таблицы. В первой содержатся 10 строк, во второй 100 строк. Сколько строк будет в результирующей таблице, если к первой присоединить вторую с помощью LEFT JOIN при условии, что никаких сведений об уникальности ключей ни в одной из таблиц?



Пришлите в чат ответ, какое max и min количество строк может быть в результирующей таблице?

Дано 2 таблицы. В первой содержатся 10 строк, во второй 100 строк. Сколько строк будет в результирующей таблице, если к первой присоединить вторую с помощью LEFT JOIN при условии, что в первой таблице customer_id имеет тип primary key, а во второй customer_id - это foreign key из первой таблицы?



Что же не так с джоинами в ClickHouse?

JOIN и CLICKHOUSE

Говорят, что ClickHouse плохо джоинит. Почему?

- Столбцовая структура хранения данных
- Распределенная природа - накладные расходы на сетевую связь и передачу данных.
- Отсутствие индексов
    - ClickHouse не поддерживает традиционные индексы типа B-tree или bitmap, обычно используемые в базах данных, основанных на строках, для ускорения операций объединения. Вместо этого он полагается на сортировку и сжатие первичных ключей, которые не всегда оптимальны для запросов с большим количеством соединений.

JOIN и CLICKHOUSE

- Не оптимизирует порядок JOIN-ов
- Не фильтрует по ключу соединения *
- Не поддерживает сравнение значений (>, <) в качестве условий соединения
- Не выбирает алгоритм JOIN-а, основываясь на собранной статистике
- Не обрабатывает исключения по памяти

Как же быть?

- Денормализация, избыточность
- Материализованные представления
- Выбор алгоритма объединения
- Оптимизация структуры таблиц - минимизация объема данных для сканирования во время соединения
- Распределенные таблицы - минимизация межузловых соединений (по возможности)
- Использование массивов и вложенных данных

Создаем таблицы для экспериментов

```
-- Создаем таблицы с 2 млн. и 2 млрд. строк

CREATE TABLE 2billion (idx Int64) ENGINE = Log
CREATE TABLE 2million (idx Int64) ENGINE = Log

INSERT INTO 2billion (idx) select * from numbers(1, 2000000000)
INSERT INTO 2million (idx) select * from numbers(1, 2000000000, 1000)
```

LEFT JOIN

```
SET send_logs_level='trace';
select count(*)
from 2billion
left join 2million using(idx)
```

```
executeQuery: Read 2002000000 rows, 14.92 GiB in 190.654906 sec., 10500647.699042164
rows/sec., 80.11 MiB/sec.
HashJoin: Join data is being destroyed, 134217728 bytes and 2000000 rows in hash table
MemoryTracker: Peak memory usage (for query): 148.85 MiB.
```

LEFT JOIN

```
SET send_logs_level='trace';
select count(*)
from 2million
join 2billion using(idx)
```

```
<Error> executeQuery: Code: 241. DB::Exception: Memory limit (total) exceeded: would use 16.46 GiB (attempt to allocate chunk of 17179869184 bytes), maximum: 11.41 GiB. OvercommitTracker decision: Query was selected to stop by OvercommitTracker.: While executing FillingRightJoinSide. (MEMORY_LIMIT_EXCEEDED) (version 24.3.5.47.altinitystable (altinity build)) (from 127.0.0.1:33602) (in query: select count(*) from 2million left join 2billion using(idx)), Stack trace (when copying this message, always include the lines below):
```

RIGHT JOIN

```
SET send_logs_level='trace';
select count(*)
from 2billion
right join 2million using(idx)
```

```
executeQuery: Read 2002000000 rows, 14.92 GiB in 238.264494 sec., 8402426.926439153
rows/sec., 64.11 MiB/sec.
HashJoin: Join data is being destroyed, 152491104 bytes and 2000000 rows in hash table
MemoryTracker: Peak memory usage (for query): 169.35 MiB.
```

IN

```
SET send_logs_level='trace';
select count(*)
from 2million as 2m
join (select *
          from 2billion
          where idx IN (select idx
                        from 2million)) as 2m
using(idx)
```

```
executeQuery: Read 2004000000 rows, 14.93 GiB in 141.81466 sec., 14131120.153586378
rows/sec., 107.81 MiB/sec.
HashJoin: Join data is being destroyed, 134217728 bytes and 2000000 rows in hash table
MemoryTracker: Peak memory usage (for query): 189.99 MiB.
```

Настройки

- Тип соединения по умолчанию можно переопределить с помощью join_default_strictness
- Поведение ClickHouse для ANY JOIN зависит от настройки any_join_distinct_right_table_keys
- join_algorithm
- join_any_take_last_row
- join_use_nulls
- partial_merge_join_optimizations
- partial_merge_join_rows_in_right_blocks
- join_on_disk_max_files_to_merge
- any_join_distinct_right_table_keys

merge join

```
SET send_logs_level='trace';
select count(*)
from 2million
join 2billion using(idx)
SETTINGS join_algorithm = 'full_sorting_merge'
```

```
executeQuery: Read 2004000000 rows, 14.93 GiB in 141.81466 sec., 14131120.153586378
rows/sec., 107.81 MiB/sec.
HashJoin: Join data is being destroyed, 134217728 bytes and 2000000 rows in hash table
MemoryTracker: Peak memory usage (for query): 189.99 MiB.
```

Внешние словари

```
dictGet('dict_name', attr_names, id_expr)
dictGetOrDefault('dict_name', attr_names, id_expr, default_value_expr)
dictGetOrNull('dict_name', attr_name, id_expr)
```

Аргументы:

- `dict_name` - Имя словаря, строка;
- `attr_names` - Имя столбца словаря, строка, или кортеж имен столбцов, Tuple(String literal);
- `id_expr` - Ключ;
- `default_value_expr` - Значения, возвращаемые, если словарь не содержит строки с `id_expr` ключом.

Predicate Pushdown

Переопределяет условия фильтрации (предикаты) ближе к операторам, которые сканируют данные, что снижает объем обрабатываемых данных и улучшает использование индексов.

- Доступен с версии 24.4
- Задается параметром optimize_move_to_prewhere (Default =1)
- Работает только для таблиц с движком *MergeTree
- Разное поведение для INNER и LEFT/RIGHT JOIN-ов

Predicate Pushdown

LEFT JOIN BEFORE AND AFTER

```
SELECT * FROM test_table_1 AS lhs
LEFT JOIN test_table_2 AS rhs ON lhs.id = rhs.id
WHERE lhs.id = 5;

Before:

Elapsed: 22.680 sec.
Processed 150.01 million rows, 3.79 GB (6.61 million rows/s., 167.06 MB/s.)
Peak memory usage: 18.42 GiB.

After:

Elapsed: 0.005 sec.
Processed 16.38 thousand rows, 131.19 KB (3.30 million rows/s., 26.45 MB/s.)
Peak memory usage: 579.28 KiB.
```

OUTER JOIN –> INNER JOIN

Доступен с версии 24.4

- Автоматическое преобразование: Если условие фильтрации после (LEFT/RIGHT) OUTER JOIN отсекает ненужные строки, то OUTER JOIN преобразуется в INNER JOIN.

- Это приводит к возможностям для оптимизации, включая применение predicate pushdown в большем количестве сценариев. После замены JOIN наблюдаются улучшения в производительности запросов.

Вопросы?

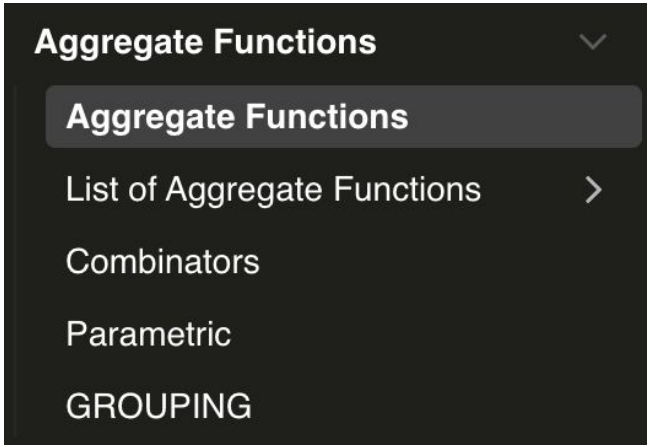
Ставим “+”,
если вопросы есть

Ставим “-”,
если вопросов нет

Агрегатные функции

Что такое Агрегатные функции?

Агрегация относится к таким операциям, когда больший набор строк свёртывается в меньший. Типичные агрегатные функции - COUNT, MIN, MAX, SUM и AVG.



Комбинаторы

Комбинатор — специальный суффикс, который добавляется к названию агрегатной функции и модифицирует логику работы этой функции. Для одной функции можно использовать несколько комбинаторов одновременно.

-If

<aggr_func_name>If(<expr>, <if_condition>)

```
TABLE_2
| id | date                | description |
|----|---------------------|-------------|
| 0  | 2020-01-01 10:00:05 | cancel      |
| 0  | 2020-01-01 10:05:05 | purchase    |
| 3  | 2020-01-01 12:01:07 | purchase    |
```



```
SELECT countIf(toHour(date)>10) AS after_10 FROM TABLE_2;
```



```
|after_10|
|--------|
| 1      |
```

-Distinct

<aggr_func_name>Distinct(<expr>) / <aggr_func_name>(DISTINCT <expr>)

```
TABLE_2
| id | date                | description |
|----|---------------------|-------------|
| 0  | 2020-01-01 10:00:05 | cancel      |
| 0  | 2020-01-01 10:05:05 | purchase    |
| 3  | 2020-01-01 12:01:07 | purchase    |
```



```
SELECT countDistinct(description) AS uniq_events FROM TABLE_2;
```



```
|uniq_events|
|-----------|
| 2         |
```

-ForEach

<aggr_func_name>ForEach(<arr>)

```
TABLE_2
| id | arr     |
|----|---------|
| 1  | [1,2,3] |
| 2  | [2,3,7] |
| 3  | [9,1,1] |
```

↓

```
SELECT maxForEach(arr) FROM TABLE_2
```

↓

```
| maxForEach(arr)   |
|-------------------|
| [9,3,7]           |
```

-State

-Merge — доагрегирует данные и возвращает готовое значение;

-MergeState — выполняет слияние промежуточных состояний агрегации аналогично комбинатору Merge, но возвращает не готовое значение, а промежуточное состояние агрегации аналогично комбинатору State.

-Resample

<aggr_func_name>Resample(<start>, <end>, <step>)(<aggr_func_params>, <resample_key>)

```
TABLE_2
| id | date                | description |
|----|---------------------|-------------|
| 0  | 2020-01-01 10:00:05 | cancel      |
| 0  | 2020-01-01 10:05:05 | purchase    |
| 3  | 2020-01-01 12:01:07 | purchase    |
```



```
SELECT groupArrayResample(0, 4, 2)(description, id) FROM TABLE_2;
```



```
|groupArrayResample(0, 4, 2)(description, id)|
|---------------------------------------------|
| [['cancel','purchase'],['purchase']]        |
```

Обработка NULL

```
TABLE_2
| id | date                | description |
|----|---------------------|-------------|
| 0  | 2020-01-01 10:00:05 | cancel      |
| 0  | 2020-01-01 10:05:05 | purchase    |
| 3  | 2020-01-01 12:01:07 | NULL        |
```



```
SELECT groupArray(description) FROM TABLE_2;
```



```
|groupArray(description)|
|-----------------------|
| ['cancel','purchase'] |
```

Обработка NULL

```
TABLE_2
| id | date                | description |
|----|---------------------|-------------|
| 0  | 2020-01-01 10:00:05 | cancel      |
| 0  | 2020-01-01 10:05:05 | purchase    |
| 3  | 2020-01-01 12:01:07 | NULL        |
```



```
SELECT count() FROM TABLE_2;
```



```
|count()|
|-------|
| 3     |
```

Обработка NULL

```
TABLE_2
| id | date                 | description |
|----|----------------------|-------------|
| 0  | 2020-01-01 10:00:05  | cancel      |
| 0  | 2020-01-01 10:05:05  | purchase    |
| 3  | 2020-01-01 12:01:07  | NULL        |
```



```
SELECT count(description) FROM TABLE_2;
```



```
|count(description)|
|------------------|
| 2                |
```

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Домашнее задание

Рефлексия

Будут ли полезны специальные виды JOIN в вашей работе?

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо за внимание!

Приходите на следующие вебинары

Александра Гроховская

Senior Data Analyst / Team Lead
Ph.D.

*Преподаватель курса **ClickHouse для инженеров и архитекторов БД** в OTUS*

LinkedIn