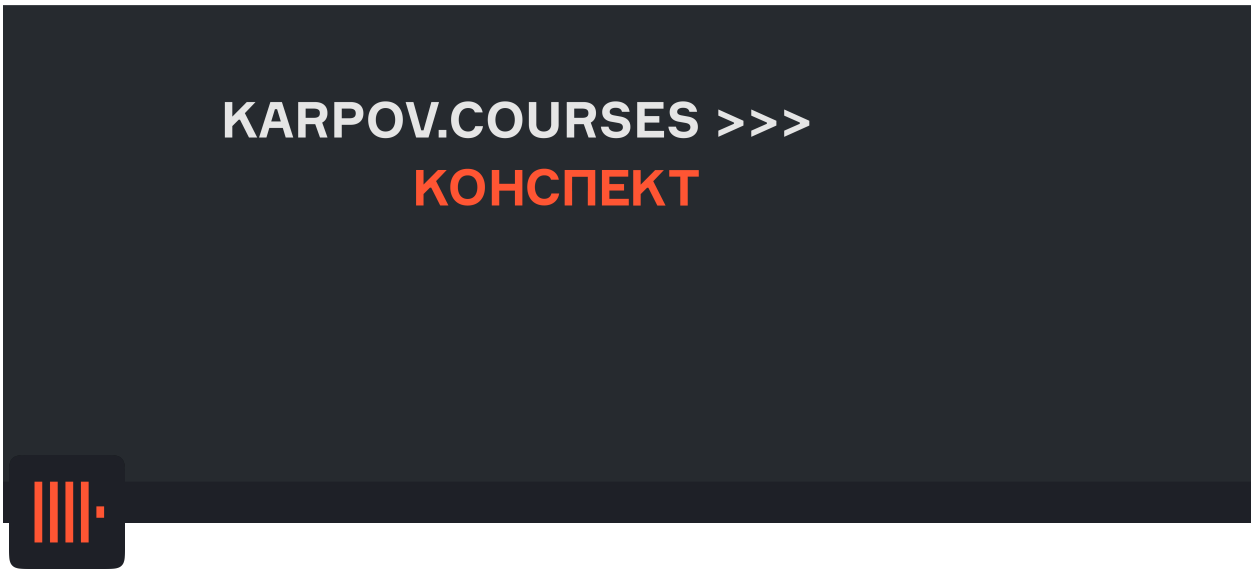


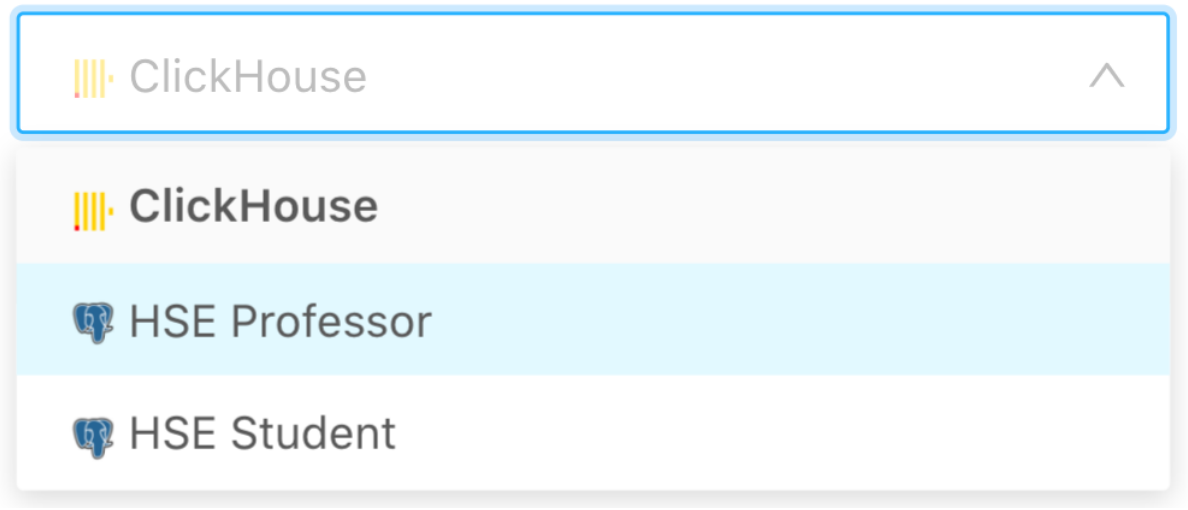
> Конспект > 8 урок > SQL

> Оглавление

> Подключаемся

В ClickHouse нет оконных функций, поэтому нужно подключиться к Postgres:

1. Redash → New Query → HSE Professor



Как зайти в Redash?

1. Переходим на https://redash.lab.karpov.courses/
2. Заполняем поля
   Логин: ваш email
   Пароль: пароль от личного кабинета

Как зайти в Redash? (подробнее)

> Что такое оконные функции

Пришло время узнать о том, как можно получить значения соседних ячеек, не меняя при этом группировку строк. Такая потребность может появиться, например, когда вам нужно в одной строке учитывать значения из соседних строк.Допустим, вы каждый день считаете, сколько вы заработали денег за сегодня. Помимо этого, вы хотите знать накопленную сумму заработанных денег от начала года и до текущей даты. И, в дополнение к этому, вы разбиваете данные по городам, и хотите видеть сумму за день и накопленную сумму по каждому городу отдельно.

Город	Дата	Выручка за сегодня	Накопленная выручка
Мск	01.01.2020	758 349,00 ₽	758 349,00 ₽
Мск	02.01.2020	622 781,00 ₽	1 381 130,00 ₽
Мск	03.01.2020	410 851,00 ₽	1 791 981,00 ₽
Мск	04.01.2020	957 158,00 ₽	2 749 139,00 ₽
Мск	05.01.2020	164 747,00 ₽	2 913 886,00 ₽
Мск	06.01.2020	741 391,00 ₽	3 655 277,00 ₽
Мск	07.01.2020	494 280,00 ₽	4 149 557,00 ₽
Мск	08.01.2020	285 008,00 ₽	4 434 565,00 ₽
Мск	09.01.2020	643 484,00 ₽	5 078 049,00 ₽
СПб	01.01.2020	530 844,30 ₽	530 844,30 ₽
СПб	02.01.2020	435 946,70 ₽	966 791,00 ₽
СПб	03.01.2020	287 595,70 ₽	1 254 386,70 ₽
СПб	04.01.2020	670 010,60 ₽	1 924 397,30 ₽
СПб	05.01.2020	115 322,90 ₽	2 039 720,20 ₽
СПб	06.01.2020	518 973,70 ₽	2 558 693,90 ₽
СПб	07.01.2020	345 996,00 ₽	2 904 689,90 ₽
СПб	08.01.2020	199 505,60 ₽	3 104 195,50 ₽
СПб	09.01.2020	450 438,80 ₽	3 554 634,30 ₽

Именно в таком случае вам на помощь приходят оконные функции. Концептуально, оконные функции можно описать как:

- некоторый выход на N строк вперед и/или назад с текущей строки
- сбор данных с этих строк
- операции над собранными данными
- запись в исходную строку
- повтор на следующей строке

Именно эта концепция и отражена в названии – с каждой строки создается некоторое окно, и по этому окну происходит подсчет.

> Синтаксис оконных функций

Оконные функции можно вызывать непосредственно в блоке `SELECT`, используя запрос вида

```
SELECT
    city,
    date,
    revenue,
    SUM(revenue) OVER w AS revenue_sum
FROM
    data
```

```
WINDOW w AS
    (
    PARTITION BY city
    ORDER BY date ASCROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    )
ORDER BY
    city ASC,
    date ASC
```

Результат выполнения запроса:

	city	date	revenue	revenue_sum
0	Москва	2019-01-01	100	100
1	Москва	2019-01-02	200	300
2	Москва	2019-01-03	150	450
3	Москва	2019-01-04	170	620
4	Москва	2019-01-05	140	760
5	Москва	2019-01-06	190	950
6	Москва	2019-01-07	210	1160
7	Москва	2019-01-08	210	1370
8	Москва	2019-01-09	190	1560
9	Москва	2019-01-10	200	1760
10	Москва	2019-01-11	220	1980
11	Москва	2019-01-12	190	2170
12	СПб	2019-01-01	50	50
13	СПб	2019-01-02	100	150
14	СПб	2019-01-03	75	225
15	СПб	2019-01-04	85	310
16	СПб	2019-01-05	95	405
17	СПб	2019-01-06	120	525
18	СПб	2019-01-07	95	620
19	СПб	2019-01-08	100	720
20	СПб	2019-01-09	110	830

В данном запросе мы объявили окно с помощью функции `OVER` в блоке `FROM`, передав в нее следующие параметры:

- `PARTITION BY city` – означает, что мы хотим работать с городами как с отдельными разделами
- `ORDER BY date ASC` – когда мы создали окно для каждого города, мы хотим отсортировать строчки в таком окне по возрастанию даты
- `ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` – означает, что мы хотим, чтобы для каждой строки внутри города в окно попадала таблица от начала окна до текущей строки. Иными словами, для первой строки это будет только одна строка, для второй это будет первая и вторая, для третьей

это будет первая + вторая + третья, и так далее. Порядок строк задается выше параметром `ORDER BY`.

Далее, с помощью этого окна, для каждого города и для каждой строчки мы имеем диапазон дат от начала периода до текущей даты, и по этому окну мы делаем сумму. Таким образом, на каждый день для каждого города у нас получается накопительная сумма.

Выше мы описывали окно в блоке `FROM`, и потом вызывали его в блоке `SELECT`. Существует возможность вызова окна сразу в блоке `SELECT`, без его именования:

```
SELECT
    id,
    section,
    header,
    score,
    row_number() OVER ()  AS num
FROM news;
```

```
+----+---------+-----------+-------+-----+
| id | section |  header   | score | num |
+----+---------+-----------+-------+-----+
|  1 |       2 | Заголовок |    23 |   1 |
|  2 |       1 | Заголовок |     6 |   2 |
|  3 |       4 | Заголовок |    79 |   3 |
|  4 |       3 | Заголовок |    36 |   4 |
|  5 |       2 | Заголовок |    34 |   5 |
|  6 |       2 | Заголовок |    95 |   6 |
|  7 |       4 | Заголовок |    26 |   7 |
|  8 |       3 | Заголовок |    36 |   8 |
+----+---------+-----------+-------+-----+
```

> Синтаксис оконных функций (продолжение)

При создании окна ему можно передать следующие параметры:

- `PARTITION BY [city]`
- `ORDER BY [date ASC]`

- `ROWS` [`BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW`]

– `UNBOUNDED PRECEDING` – указывает, что окно начинается с первой строки секции

– `UNBOUNDED FOLLOWING` – указывает, что окно заканчивается на последней строке секции

Давайте рассмотрим эти параметры подробнее.

PARTITION BY

Группирует строки запроса в разделы, которые потом обрабатываются оконной функцией независимо друг от друга. Работает аналогично блоку `GROUP BY` в запросе, только группирует данные для окна.

ORDER BY

Сортирует результаты внутри раздела, и тем самым определяет порядок, в котором оконная функция будет работать со строками. Работает аналогично блоку `ORDER BY` на уровне запроса.

ROWS/RANGE

Задает параметры рамки окна. Рамки используются в тех оконных функциях, которые работают с рамками, а не с разделом целиком. Первым аргументом задается начало рамки, вторым аргументом конец рамки, и дополнительно, третьим аргументом задается исключение из рамки.

Рамку можно задать в нескольких режимах, а именно:

- `ROWS` – Начало и конец рамки задаются в строках относительно текущей строки. Например `ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING` означает создание рамки на 3 строки вверх и вниз относительно текущей строки.
- `RANGE` – Начало и конец рамки задаются в разнице значений столбца из `ORDER BY`. Например, если в `ORDER BY` у вас столбец `event_date` с типом данных date, то определение окна можно задать следующим образом `RANGE BETWEEN '3 day' PRECEDING AND '3 days' FOLLOWING` что будет означать рамку на 3 дня назад и вперед.

При указании рамки через `RANGE` обязательным условием является только один столбец в `ORDER BY` окна.

> Синтаксис оконных функций (продолжение 2)

Любые агрегатные функции можно использовать с помощью окна. Это необходимо делать явно с указанием `OVER` и имени окна.
Например `AVG(revenue) OVER w AS avg_revenue` Помимо обычных агрегатных функций, существуют специальные функции для работы через окно.

- `row_number()` – номер текущей строки в разделе, начинается 1
- `lag(my_field, 1)` – получить значение столбца `my_field` из предыдущей строки
- `lead(my_field, 1)` – получить значение столбца `my_field` из следующей строки

Функция	Тип результата	Описание
`row_number()`	`bigint`	номер текущей строки в её разделе, начиная с 1
`rank()`	`bigint`	ранг текущей строки с пропусками; то же, что и `row_number` для первой родственной ей строки
`dense_rank()`	`bigint`	ранг текущей строки без пропусков; эта функция считает группы родственных строк
`percent_rank()`	`double precision`	относительный ранг текущей строки: (`rank` - 1) / (общее число строк - 1)
`cume_dist()`	`double precision`	относительный ранг текущей строки: (число строк, предшествующих или родственных текущей) / (общее число строк)
`ntile(`*число_групп* `integer)`	`integer`	ранжирование по целым числам от 1 до значения аргумента так, чтобы размеры групп были максимально близки
`lag(`*значение* `anyelement [,` *смещение* `integer [,` *по_умолчанию* `anyelement ]])`	тип аргумента *значение*	возвращает *значение* для строки, положение которой задаётся *смещением* от текущей строки к началу раздела; если такой строки нет, возвращается значение *по_умолчанию* (оно должно иметь тот же тип, что и *значение*). Оба параметра *смещение* и *по_умолчанию* вычисляются для текущей строки. Если они не указываются, то *смещение* считается равным 1, а *по_умолчанию* — NULL
`lead(`*значение* `anyelement [,` *смещение* `integer [,` *по_умолчанию* `anyelement ]])`	тип аргумента *значение*	возвращает *значение* для строки, положение которой задаётся *смещением* от текущей строки к концу раздела; если такой строки нет, возвращается значение *по_умолчанию* (оно должно иметь тот же тип, что и *значение*). Оба параметра *смещение* и *по_умолчанию* вычисляются для текущей строки. Если они не указываются, то *смещение* считается равным 1, а *по_умолчанию* — NULL
`first_value(`*значение* `any)`	тип аргумента *значение*	возвращает *значение*, вычисленное для первой строки в рамке окна
`last_value(`*значение* `any)`	тип аргумента *значение*	возвращает *значение*, вычисленное для последней строки в рамке окна
`nth_value(`*значение* `any,` *n* `integer)`	тип аргумента *значение*	возвращает *значение*, вычисленное в *n*-*ой* строке в рамке окна (считая с 1), или NULL, если такой строки нет

> Порядок построения запроса

```
SELECT
    [column_1],
    [column_2],
    [window_function]() OVER [window_name]
FROM
    [table_name]
WHERE
    [...]
GROUP BY
    [...]
HAVING
    [...]
WINDOW [window_name] AS (
    PARTITION BY [...]
    ORDER BY [...]
    [window_frame])
ORDER BY
    [...]
LIMIT []
```

Несколько примеров:

```
SELECT
    country,
    city,
    rank() OVER country_sold_avg
FROM
    sales
WHERE
    month BETWEEN 1 AND 6
GROUP BY
    country,
    city
HAVING
    sum(sold) > 10000
WINDOW
    country_sold_avg AS (
        PARTITION BY country
        ORDER BY avg(sold) DESC
    )
ORDER BY
    country,
    city
```

```
SELECT
    city,
    month,
```

```
    sum(sold) OVER (
        PARTITION BY city
        ORDER BY month
RANGE UNBOUNDED PRECEDING ) total
FROM
    sales
```
