<https://console.cloud.google.com/bigquery>

Тренировка SQL в Google Cloud Console

**Порядок написания и выполнения операторов:**

SELECT 6

DISTINCT 7

FROM 1

JOIN 2

WHERE 3

GROUP BY 4

HAVING 5

ORDER BY 8

LIMIT OFFSET 9

|  |  |  |
| --- | --- | --- |
| **Синтаксис** | **Теория и примеры** | |
| Комментарии | -- Так выглядит однострочный комментарий  /\* Строка 1 многострочного комментария  Строка 2  Строка 3 \*/ | |
| ; | Конец запроса | |
| FROM | Описывает из какой таблицы выгружать данные | |
| SELECT | Выгрузка определенных полей  SELECT поле\_1,  поле\_2,  поле\_3  ...  FROM таблица; | |
| LIMIT | Ограничение строк вывода  SELECT поле\_1,  поле\_2,  поле\_3  ...  FROM таблица  LIMIT 10; | |
| OFFSET | Пропускает определённое число записей  SELECT поле\_1,  поле\_2,  поле\_3  ...  FROM таблица  OFFSET 5;  SELECT поле\_1,  поле\_2,  поле\_3  ...  FROM таблица  LIMIT 10 OFFSET 5; | |
| WHERE | Задает условие для среза  SELECT track\_id, -- поля для выгрузки  album\_id  ...  FROM track -- таблица, из которой выгружают данные  WHERE track\_id = 5; -- условие для среза данных  Задавать условия можно с помощью операторов сравнения. Равенство значений указывают знаком =, неравенство — знаками <>.  Обозначения неравенства !=, в PostgreSQL тоже сработает.  Остальные операторы сравнения выглядят так:  > — больше;  < — меньше;  >= — больше или равно;  <= — меньше или равно.  Сравнение с числом записывают просто:  WHERE поле < 5.  Сравнение с символьным типом, набор символов нужно взять в одинарные кавычки:  WHERE поле = 'Иванов'.  Это правило также касается даты и времени:  WHERE поле = '2013-07-01'. | |
| ORDER BY | Сортирует выводимые данные  DESC – по убыванию  SELECT \*  FROM invoice  ORDER BY total DESC-- впишите условие для сортировки  LIMIT 5-- добавьте ограничение;  ORDER BY 2 – можно не писать название столбца, а писать его номер.  Сортировка по нескольким полям. Поля сортируются поочереди. Второе поле сортируется внутри отсортированных значений первого поля и т.д.  SELECT billing\_country,  customer\_id,  total  FROM invoice  WHERE billing\_country = 'India' OR billing\_country = 'Portugal'  ORDER BY billing\_country,  customer\_id,  total DESC; | |
| GROUP BY | SELECT billing\_city,  ROUND(AVG(total))  FROM invoice  GROUP BY billing\_city -- здесь указывают поле, по которому группируют данные  LIMIT 5;  Выгрузится средняя сумма заказов для каждого города:  billing\_city round  Porto 5  Budapest 7  Reno 5  Группировать можно по нескольким полям. Тогда выгрузятся все уникальные сочетания группируемых данных  SELECT CAST(invoice\_date AS date),  billing\_country,  COUNT(total)  FROM invoice  GROUP BY CAST(invoice\_date AS date),  billing\_country  ORDER BY CAST(invoice\_date AS date) DESC  LIMIT 10;  При использовании GROUP BY нужно агрегировать все не-группируемые поля | |
| HAVING | Задает условие для группирования  Используется если нужно получить срез данных после группировки  Можно сочетать с агрегирующими функциями  SELECT customer\_id,  SUM(total)  FROM invoice  GROUP BY customer\_id  HAVING SUM(total) > 41; | |
| DISTINCT | Возвращает уникальные значения  SELECT DISTINCT billing\_country  FROM invoice  LIMIT 5;  DISTINCT - ключевое слово, которое применяется ко всем полям, указанным после него.  SELECT DISTINCT customer\_id,  billing\_country  FROM invoice  LIMIT 10;  В получившейся таблице в поле billing\_country некоторые страны будут повторяться.  Ключевое слово DISTINCT можно сочетать с агрегирующими функциями.  SELECT COUNT(DISTINCT billing\_country)  FROM invoice; | |
| IN | Значения, по которым нужно фильтровать данные, перечисляют в скобках.  Ниже два равнозначных запроса | |
| SELECT total,  customer\_id  FROM invoice  WHERE billing\_city IN ('Dublin',  'London',  'Paris',  'Boston',  'Berlin',  'Stuttgart'); | SELECT total,  customer\_id  FROM invoice  WHERE billing\_city = 'Dublin'  OR billing\_city = 'London'  OR billing\_city = 'Paris'  OR billing\_city = 'Boston'  OR billing\_city = 'Berlin'  OR billing\_city = 'Stuttgart'; |
| Тип данных в условии и в сравниваемом поле может не совпадать.  Произойдет автоматические преобразование строк '23' и '36' в числа 23 и 36.  SELECT total,  customer\_id  FROM invoice  WHERE customer\_id IN (3,  '23',  '36'); | |
| LIKE | Поиск по конкретному значению  Можно применять только к строковым значениям  % показывают, какую позицию в строке занимает шаблон:  'text%', '%text', '%text%'б 'te%xt'  SELECT \*  FROM playlist\_track  WHERE CAST(track\_id AS varchar) LIKE '43'; | |
| BETWEEN | Найти значения из диапазона  SELECT \*  FROM invoice  WHERE customer\_id BETWEEN 23 AND 48; | |
| SELECT last\_name  FROM staff  WHERE birth\_date BETWEEN '1970-01-01' AND '1970-12-31'; | |
| CAST | Изменение типа данных при выгрузке  SELECT CAST(поле AS тип данных)  FROM таблица;  Предпочтительно использовать CAST, т.к. он совместим со стандартом SQL, но  В PostgreSQL можно так:  поле::тип\_данных | |
| NULL | Специальное значение NULL обозначает отсутствие данных  Для работы со специальными значениями используют отдельные операторы:  IS NULL и IS NOT NULL  SELECT \*  FROM client  WHERE fax IS NULL  LIMIT 5; | |
| CASE | Выведется отдельный столбец с названием case в котором будет написано 'информация есть' или 'информации нет'.  SELECT fax,  CASE  WHEN fax IS NULL THEN 'информации нет'  ELSE 'информация есть' – можно не использовать  END  FROM client; | |
|  |  | |
| **Логические операторы** |  | |
| AND | WHERE city = 'Тула'  AND age > 43; | |
| OR | WHERE city = 'Тула'  OR city = 'Пермь'; | |
| NOT | WHERE NOT city = 'Тула';  WHERE NOT city = 'Тула'  AND NOT city = 'Пермь'; | |
|  | Приоритет логических операторов:  1 NOT,  2 AND,  3 OR.  Управлять порядком можно с помощью скобок.  Ниже два равнозначных запроса, но второй понятнее | |
| WHERE age > 43  AND city = 'Тула'  OR city = 'Пермь'  AND age < 43; | WHERE (age > 43  AND city = 'Тула')  OR (city = 'Пермь'  AND age < 43); |
|  |  | |
| **Работа со временем** | Еcли столбец со временем хранит данные типа varchar (строка), то тип данных лучше сразу преобразовывать в date  CAST(invoice\_date AS date) | |
| CURRENT\_DATE  CURRENT\_TIME  CURRENT\_TIMESTAMP | В PostgreSQL есть несколько операторов, которые выводят текущую дату и  время  SELECT CURRENT\_DATE -- вернет текущую дату  CURRENT\_TIME -- вернет текущее время  CURRENT\_TIMESTAMP; -- вернет текущие дату и время,  -- обычно в формате YYYY-MM-DD HH:MM:SS. | |
| DATE\_TRUNC | Функция DATE\_TRUNC «усекает» дату и время до необходимого значения: года, месяца или дня.  DATE\_TRUNC('отрезок времени', поле).  Отрезок времени может быть разным, главное, не забыть одинарные кавычки:  'microseconds' — микросекунды;  'milliseconds' — миллисекунды;  'second' — секунда;  'minute' — минута;  'hour' — час;  'day' — день;  'week' — неделя;  'month' — месяц;  'quarter' — квартал;  'year' — год;  'decade' — десятилетие;  'century' — век.  Возвращает **timestamp**  при необходимости получения **date** нужно делать преобразование типов  CAST(DATE\_TRUNC('month', some\_timestamp) AS date) | |
| SELECT DATE\_TRUNC('year', birth\_date)  FROM staff  LIMIT 1;  --(01.01.1962 00:00:00) округление до года  SELECT DATE\_TRUNC('month', birth\_date)  FROM staff  LIMIT 1;  --( 01.02.1962 00:00:00) округление до месяца | SELECT birth\_date  FROM staff  LIMIT 1;  --( 1962-02-18 00:00:00) не всегда удобный формат |
| EXTRACT | Когда нужно получить конкретную часть даты: год, месяц или минуту  Функция EXTRACT меняет тип данных — она выводит значения типа **numeric**.  EXTRACT(отрезок времени FROM поле).  CENTURY — век;  DAY — день;  DOY (от англ. day of the year) — день года, выраженный числом от 1 до 365 или 366, если год високосный;  DOW (от англ. day of the week) — день недели, выраженный числом от 0 до 6, где понедельник — 1, воскресенье — 0.  ISODOW (от англ. day of the week и ISO 8601) — день недели, выраженный числом от 1 до 7, где понедельник — 1, воскресенье — 7.  HOUR — час;  MILLISECOND — миллисекунда;  MINUTE — минута;  MONTH — месяц;  SECOND — секунда;  QUARTER — квартал;  WEEK — неделя в году;  YEAR — год.  SELECT EXTRACT(YEAR FROM birth\_date)  FROM staff  LIMIT 1;  -- (1962)  SELECT email  FROM staff  WHERE city = 'Calgary'  AND EXTRACT (YEAR FROM CAST(hire\_date **AS timestamp**)) = 2002 | |
| Разница в днях: | SELECT CURRENT\_DATE - date\_column AS days\_difference  FROM table; | |
| Прибавить к дате: | SELECT date\_column + INTERVAL '3 days'  FROM table; | |
| Вычесть из даты: | SELECT date\_column - INTERVAL '3 days'  FROM table; | |
|  |  | |
| **Математические операции:** |  | |
| +, -, \*, /. | SELECT invoice\_line\_id,  invoice\_id,  invoice\_line\_id + invoice\_id  FROM invoice\_line  LIMIT 5; | |
| ABS возвращает модуль числа | SELECT number,  ABS(number)  FROM table\_1; | |
| FLOOR округляет в меньшую сторону | SELECT number,  FLOOR(number)  FROM table\_1; | |
| CEILING округляет в большую сторону | SELECT number,  CEILING(number)  FROM table\_1; | |
| ROUND округляет до ближайшего числа | SELECT number,  ROUND(number)  FROM table\_1;  можно округлить число до определённого количества знаков после запятой  SELECT ROUND(21.5595743, 2);  Можно округлять и до десятков/сотен с отрицательным аргументом:  ROUND(value, -2) | |
| POWER возводит в степень | SELECT number,  POWER(number, 2)  FROM table\_1; | |
| SQRT извлекает квадратный корень | SELECT number,  SQRT(ABS(number))  FROM table\_1; | |
|  |  | |
| **Агрегирующие функции:**  SUM(поле) возвращает сумму значений в поле;  AVG(поле) находит среднее арифметическое для значений в поле;  MIN(поле) возвращает минимальное значение в поле;  МАХ(поле) возвращает максимальное значение в поле;  COUNT(\*) выводит количество записей в таблице, а COUNT(поле) — количество записей в поле. | SELECT SUM(total),  MIN(total),  MAX(total)  FROM invoice;  SELECT ROUND(AVG(total), 2)  FROM invoice;  Функцию COUNT, в отличие от других, можно использовать и для всей таблицы.  -- так считают записи в таблице  SELECT COUNT(\*)  FROM invoice;  -- так считают записи в поле  SELECT COUNT(billing\_postal\_code)  FROM invoice; | |
|  |  | |
| **Объединение таблиц** | image | |
| INNER JOIN  JOIN – можно в PostgreSQL | Возвращает только те строки, у которых есть совпадения в обеих таблицах  SELECT \*  FROM employees e  INNER JOIN departments d ON e.dept\_id = d.id;  Покажет только тех сотрудников, у которых есть отдел.  Объединение нескольких таблиц сразу  ...  FROM таблица\_1  INNER JOIN таблица\_2 ON таблица\_1.поле=таблица\_2.поле -- первое присоединение  INNER JOIN таблица\_3 ON таблица\_1.поле=таблица\_3.поле -- второе присоединение  ... | |
| LEFT OUTER JOIN  LEFT JOIN – можно в PostgreSQL | Возвращает все строки из левой таблицы и совпадения из правой. Если совпадения нет — подставляются NULL  SELECT \*  FROM employees e  LEFT JOIN departments d ON e.dept\_id = d.id;  Покажет всех сотрудников, даже если у них нет отдела (будет NULL в полях из departments). | |
| RIGHT OUTER JOIN  RIGHT JOIN – можно в PostgreSQL | Возвращает все строки из правой таблицы и совпадения из левой. Если совпадения нет — NULL  SELECT \*  FROM employees e  RIGHT JOIN departments d ON e.dept\_id = d.id;  Покажет все отделы, даже если в них нет сотрудников. | |
| FULL OUTER JOIN | Возвращает все строки из обеих таблиц. Если совпадений нет — подставляется NULL с одной из сторон.  SELECT \*  FROM employees e  FULL OUTER JOIN departments d ON e.dept\_id = d.id;  Покажет всех сотрудников и все отделы, даже если не связаны друг с другом. | |
| UNION  UNION ALL  вертикальное соединение таблиц | Извлекать поля из одной таблицы нужно в том же порядке, что и из другой. Число извлекаемых полей тоже должно совпадать.  Типы данных в полях должны соответствовать друг другу. Объединить поле типа integer с полем типа varchar не получится, а поля с типами integer и real — можно.  Оператор UNION: Абсолютные дубликаты в итоговую таблицу не входят: останется только одна запись.  Оператор UNION ALL: Все записи входят в итоговую таблицу, даже полные дубликаты.  SELECT i.billing\_country,  COUNT(i.total) AS total\_purchases  FROM invoice AS i  WHERE i.billing\_country IN ('USA',  'Germany',  'Brazil')  AND EXTRACT(YEAR FROM cast(invoice\_date AS date)) = 2009  GROUP BY i.billing\_country  UNION  SELECT i.billing\_country,  COUNT(i.total) AS total\_purchases  FROM invoice AS i  WHERE i.billing\_country IN ('USA',  'Germany',  'Brazil')  AND EXTRACT(YEAR FROM cast(invoice\_date AS date)) = 2013  GROUP BY i.billing\_country; | |
|  |  | |
| **VIEW** | Представление (VIEW) — это виртуальная таблица, которая создаётся на основе запроса SELECT. Оно не хранит данные, а показывает результат запроса при каждом обращении (создается как бы новая таблица из имеющихся в базе данных) | |
| CREATE VIEW | CREATE VIEW active\_users AS  SELECT id, name  FROM users  WHERE is\_active = true; | |
| DROP VIEW | DROP VIEW view\_name; | |
|  |  | |
| **Подзапросы** |  | |
| Подзапросы во FROM | Подзапрос во FROM всегда требует alias (псевдоним).  Можно использовать SELECT \* для отладки.  Удобен для поэтапного решения задач (декомпозиции).  Хорошо работает с агрегатами: AVG(), SUM(), COUNT() и др.  Пример: Посчитать среднюю стоимость аренды по рейтингам, а затем — среднее значение этих средних.  SELECT AVG(best\_rating.average\_rental)  FROM (SELECT m.rating,  AVG(m.rental\_rate) AS average\_rental  FROM movie AS m  GROUP BY m.rating  ORDER BY average\_rental DESC  LIMIT 5) AS best\_rating; | |
| Подзапросы в WHERE | Подзапрос в WHERE используется для фильтрации данных на основе результатов другого запроса.  Это удобно, когда:  Нужно уточнить срез данных  Есть список значений, полученный из другого запроса  Запрос нужно автоматизировать и переиспользовать  Пример: Найти, сколько потратили в этом году клиенты, у которых в прошлом году была самая высокая выручка.  Подход:  В подзапросе выбрать id клиентов с наибольшей выручкой за прошлый год.  В основном запросе ограничить выборку этими id и текущим годом.  SELECT customer\_id, SUM(amount) AS yearly\_spent  FROM sales  WHERE customer\_id IN (  SELECT customer\_id  FROM sales  WHERE year = 2024  GROUP BY customer\_id  ORDER BY SUM(amount) DESC  LIMIT 20  ) AND year = 2025  GROUP BY customer\_id; | |
| CTE | CTE (от англ. common table expressions) общие табличные выражения  CTE — это временные именованные подзапросы, которые определяются с помощью оператора WITH и могут использоваться в основном SQL-запросе.  Помогают сделать код более читаемым и структурированным, особенно если подзапросов много.  Важно: Нельзя использовать CTE после WHERE  WITH  alias1 AS (  подзапрос\_1  ),  alias2 AS (  подзапрос\_2  )  -- Без запятой после последнего подзапроса!  SELECT ...  FROM alias1  JOIN alias2 ON ...  Пример: Найти топ-5 стран по количеству заказов и показать, сколько клиентов из этих стран зарегистрировано.  WITH  i AS (  SELECT billing\_country AS country,  COUNT(total) AS total\_invoice  FROM invoice  GROUP BY billing\_country  ORDER BY total\_invoice DESC  LIMIT 5  ),  c AS (  SELECT country,  COUNT(customer\_id) AS total\_clients  FROM client  GROUP BY country  )  SELECT i.country,  i.total\_invoice,  c.total\_clients  FROM i  LEFT OUTER JOIN c ON i.country = c.country  ORDER BY i.total\_invoice DESC; | |
|  |  | |
| **DML** | DML — это подмножество SQL-команд, предназначенное для работы с данными в таблицах: добавления, изменения, удаления и выборки. | |
| INSERT | INSERT INTO table\_name (column1, column2, … )  VALUES (value1, value2, … ); | |
| UPDATE | UPDATE table\_name  SET column1 = value1, column2 = value2, …  WHERE condition;  Без WHERE обновятся все строки, что может привести к потере данных. | |
| DELETE | DELETE FROM table\_name  WHERE condition;  Без WHERE будут удалены все строки из таблицы | |
|  |  | |
| Транзакция | Транзакция — это единый блок операций, который должен выполниться целиком. Если одна из операций внутри транзакции завершается с ошибкой, все предыдущие изменения откатываются.  BEGIN  UPDATE …  UPDATE …  UPDATE …  COMMIT  Если что-то идёт не так (ошибка, не тот результат, неверное условие), то выполняется ROLLBACK, чтобы отменить все изменения, сделанные в этой транзакции.  -- Если всё хорошо:  COMMIT;  -- Если ошибка:  ROLLBACK; | |
|  |  | |
| **DDL** | **DDL** команды используются для создания, изменения и удаления структуры объектов базы данных | |
| ALTER TABLE  Уточнить для PostgreSQL | --Добавить столбец  ALTER TABLE table\_name  ADD column\_name data\_type; | |
| --Изменить столбец  ALTER TABLE table\_name  ALTER COLUMN column\_name SET DATA TYPE new\_data\_type; | |
| -- Удалить столбец  ALTER TABLE table\_name  DROP COLUMN column\_name; | |
| --Переименовать столбец  ALTER TABLE table\_name  RENAME COLUMN old\_name TO new\_name; | |
| TRUNCATE TABLE – удаляет все строки из таблицы | TRUNCATE TABLE table\_name; | |
| CREATE TABLE – удаляет таблицу из БД | CREATE TABLE table\_name (  column1 datatype,  column2 datatype,  column3 datatype,  …  );  CREATE TABLE country (  id INT NOT NULL,  ccode CHAR(2),  name VARCHAR(60),  PRIMARY KEY (id)  ); | |
| DROP TABLE | DROP TABLE table\_name; | |
|  |  | |
| **Функции для работы со строками:**  LENGTH  INITCAP  LOWER  UPPER  LTRIM  RTRIM  REPLACE  CONCAT | |  |  |  |  | | --- | --- | --- | --- | | Функция | Описание | Пример | Результат | | LENGTH | Возвращает длину строки | LENGTH('Привет!') | 7 | | INITCAP | Меняет первую букву каждого слова в строке на заглавную, а остальные буквы меняет на строчные | INITCAP('иванов александр') | Иванов Александр | | LOWER | Возвращает строку со всеми символами в нижнем регистре | LOWER('ПРИвет') | привет | | UPPER | Возвращает строку со всеми символами в верхнем регистре | UPPER('ПРИвет') | ПРИВЕТ | | LTRIM | Удаляет последние символы слева, которые передают функции | LTRIM('г. Санкт-Петербург', 'г.') | Санкт-Петербург | | RTRIM | Удаляет последние символы справа, которые передают функции | RTRIM('Санкт-Петербург г. ', 'г.') | Санкт-Петербург | | REPLACE | Возвращает строку, в которой все совпадения символов заменены на символы, которые передают функции | REPLACE('Санкт/Петербург', '/', '-') | Санкт-Петербург | | CONCAT | Объединяет несколько значений в одно | CONCAT('Петров', ' ', 'Александр', ' ', 'Александрович') | Петров Александр Александрович | | |
|  |  | |
| PG\_TYPEOF() | Проверка типа данных  SELECT PG\_TYPEOF(100) вернёт значение 'integer', а  SELECT PG\_TYPEOF(CAST(100 as numeric)) вернёт 'numeric'. | |
|  |  | |
| **Оконные функции**  OVER () | Оконные функции выполняют вычисления для набора записей, объединённых по какому-либо признаку. Такой набор называют окном — отсюда и название функций.  Вместе с текущей записью оконная функция обрабатывает остальные записи, которые входят в то же окно. **Для каждой записи функция выводит одно значение.** Этим оконная функция отличается от агрегирующей: и та и другая вычисляются для набора записей, но **оконная функция не объединяет записи в одну как агрегирующая**, сохраняя независимость записей.  Пример:  SELECT AVG(revenue) AS total\_avg  FROM online\_store.orders;  -- total\_avg  -- 5.49986  Если к агрегирующей функции добавить выражение OVER , функция превратится в оконную:  SELECT  \*,  AVG(revenue) **OVER ()** AS total\_avg  FROM online\_store.orders  LIMIT 5;  -- user\_id event\_dt revenue total\_avg  -- 55519067947 2020-06-14 5.62 5.49986  -- 831753022495 2020-06-14 1.33 5.49986  Ограничение!: нельзя использовать DISTINCT в аргументе оконной функции | |
| PARTITION BY | Чтобы разделить данные на группы, используют параметр окна PARTITION BY.  Функция будет рассчитана не по всей таблице, как в случае с пустым выражением OVER, а для каждой группы отдельно.  SELECT \*,  AVG(revenue) OVER (PARTITION BY event\_dt) AS date\_avg  FROM online\_store.orders;  -- user\_id event\_dt revenue total\_avg  -- 55519067947 2020-06-14 5.62 5.49986  -- 831753022495 2020-06-14 1.33 5.49986  -- 345356355 2020-08-21 4.52 1.97994  -- 35345353453 2020-08-21 9.44 1.97994  total\_avg расчитан для каждой даты отдельно  После PARTITION BY, как и после GROUP BY, можно указать несколько полей, и тогда окно будет сформировано по нескольким полям сразу. В одно окно войдёт каждая уникальная комбинация значений из этих полей.  SELECT \*,  AVG(revenue) OVER (PARTITION BY user\_id, event\_dt)  FROM online\_store.orders | |
| ROW\_NUMBER () | С помощью оконных функций можно ранжировать записи, то есть нумеровать их по определённому правилу. Для ранжирования в SQL используют функцию ROW\_NUMBER(). Она возвращает порядковый номер записи в окне.  ROW\_NUMBER() не требует аргумента  SELECT  \*,  ROW\_NUMBER() OVER ()  FROM online\_store.orders  -- user\_id event\_dt revenue row\_number  -- 959432616 2020-06-17 7.59 1  -- 753434554 2020-06-19 5.81 2  -- 840452722 2020-06-21 3.17 3 | |
| ORDER BY | Чтобы задать другой порядок ранжирования, в выражение OVER добавляют оператор ORDER BY вместе с полем, по которому нужно ранжировать записи.  SELECT \*,  ROW\_NUMBER() OVER (ORDER BY user\_id)  FROM online\_store.orders  ORDER BY user\_id;  -- user\_id event\_dt revenue row\_number  -- 753434554 2020-06-19 5.81 1  -- 840452722 2020-06-21 3.17 2  -- 959432616 2020-06-17 7.59 3  DESC – ранжирование по убыванию | |
| RANK() | Функция RANK() присваивает одинаковым значениям одинаковый ранг.  SELECT \*,  RANK() OVER (ORDER BY user\_id)  FROM online\_store.orders  -- user\_id event\_dt revenue rank  -- 59432616ы 2020-06-17 7.59 1  -- 300768196 2020-06-25 8.74 2  -- 300768196 2020-06-27 5.26 2  -- 840452722 2020-06-19 5.81 **4 -- !** | |
| DENSE\_RANK() | В отличие от функции RANK(), DENSE\_RANK() не учитывает количество записей и назначает ранги последовательно.  SELECT \*,  DENSE\_RANK() OVER (ORDER BY user\_id)  FROM online\_store.orders  -- user\_id event\_dt revenue rank  -- 594326161 2020-06-17 7.59 1  -- 300768196 2020-06-25 8.74 2  -- 300768196 2020-06-27 5.26 2  -- 840452722 2020-06-19 5.81 **3 -- !** | |
| NTILE() | Позволяет назначать записям фиксированное количество рангов — в зависимости от аргумента (в этом случае — числа), который передают функции.  Пример: Функция NTILE() с аргументом 3 разделит записи на три группы в зависимости от значения revenue:  SELECT \*,  NTILE(3) OVER (ORDER BY revenue)  FROM online\_store.orders  -- user\_id event\_dt revenue ntile  -- 840452722 2020-06-21 3.17 1  -- 300768196 2020-06-27 5.26 1  -- 840452722 2020-06-19 5.81 2  -- 840452722 2020-06-24 6.27 2  -- 59432616 2020-06-17 7.59 3  -- 300768196 2020-06-25 8.74 3 | |
| Расчёт кумулятивных значений | Оператор ORDER BY в выражении OVER позволяет не только сортировать значения. Если сочетать его с агрегирующими функциями, можно производить вычисления кумулятивно.  Пример 1.  SELECT \*,  SUM(costs) OVER (ORDER BY dt) AS costs\_cum  FROM online\_store.costs  WHERE channel = 'Yandex';  dt channel costs costs\_cum  2020-06-14 Yandex 848 848  2020-06-15 Yandex 734 1582  2020-06-16 Yandex 833 2415  Пример 2. Без фильтра channel = 'Yandex'  В поле dt, по которому сортируют данные, встречается несколько одинаковых значений. В таких записях значения costs будут складываться. При этом сумма будет также рассчитана кумулятивно.  SELECT \*,  SUM(costs) OVER (ORDER BY dt) AS costs\_cum  FROM online\_store.costs;  dt channel costs costs\_cum  2020-06-14 AnotherSource 1826 2674  2020-06-14 Yandex 848 2674  2020-06-15 AnotherSource 1443 4851  2020-06-15 Yandex 734 4851  2020-06-16 AnotherSource 1508 7192  2020-06-16 Yandex 833 7192  Сумму с накоплением можно рассчитать и для каждого окна отдельно. Для этого нужно добавить оператор PARTITION BY и указать нужное поле.  SUM(session\_duration) OVER (PARTITION BY user\_id ORDER BY session\_start) | |
| Функции смещения:  LEAD()  LAG() | Функции смещения возвращают данные из других записей в зависимости от их расстояния от текущего значения. Функция LAG() позволяет возвращать предыдущие записи, а LEAD() — следующие.  LEAD(поле, смещение, значение по умолчанию) OVER (определение окна);  LAG(поле, смещение, значение по умолчанию) OVER (определение окна). | |
| WINDOW | Когда одну оконную функцию нужно использовать много раз, её можно сохранить отдельно. Для этого используют конструкцию WINDOW.  В основном запросе конструкцию WINDOW указывают после оператора WHERE и до оператора ORDER BY.  Синтаксис у конструкции такой: WINDOW название окна AS определение окна.  SELECT \*,  ROW\_NUMBER() OVER my\_window,  RANK() OVER my\_window,  DENSE\_RANK() OVER my\_window  FROM online\_store.orders  WINDOW my\_window AS (ORDER BY revenue); | |