**Практическая работа №**

**Оконные функции в SQL**

1. Функция **OVER**

1.1 Функцияранжирования **ROW\_NUMBER**

1. Функция **PARTITION BY**
2. Функция **ROWS**
3. Функция **RANGE**
4. Функции смещения **LAG | LEAD**
5. Функция **WITH**

Оконные функции – это функции, которые позволяют осуществлять вычисления в заданном диапазоне строк внутри предложения Select. Окно – значит набор строк, в рамках которого происходит вычисление. Оконная функция позволяет разбивать весь набор данных на такие окна.

Основное преимущество оконных функций перед регулярными агрегатными функциями в том, что *оконные функции не приводят к группированию строк в одну строку вывода, строки сохраняют свои отдельные идентификаторы, а агрегированное значение добавляется к каждой строке*.

Оконные функции не изменяют выборку, а только добавляют некоторую дополнительную информацию о ней. Для простоты понимания можно считать, что SQL сначала выполняет весь запрос (кроме сортировки и limit), а уже потом считает значения окна.

1) Функция **OVER()**

Окно определяется с помощью функции **OVER()**.

Синтаксис этой функции:

**Оконная функция** (столбец для вычислений)

**OVER**

**(**

[**PARTITION BY** столбец для группировки]

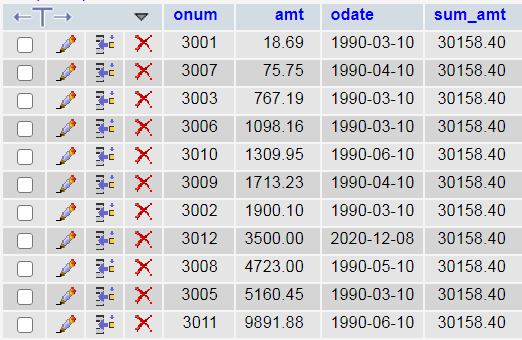
[**ORDER BY** столбец для сортировки]

[**ROWS** или **RANGE** выражение для ограничения строк в пределах группы]

**)**

Пример 1: Показать общую сумму заказов по всей таблице не исключая вывода номера заказа, его стоимости и даты.

[SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) `onum`,`amt`, `odate, [SUM](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/group-by-functions.html#function_sum)(`amt`) OVER() as sum\_amt FROM `orders`

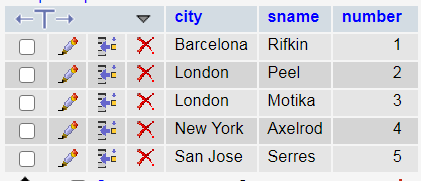


Пример 2: Пронумеруем уникальное сочетание столбцов CITY и SNAME.

Для этого задания необходимо использовать функцию **ROW\_NUMBER,** котораяприсваивает номер каждой выбранной записи.

Обратите внимание (два примера ниже), какой столбец первым будет указан после SELECT, по тому и будет проходить упорядочивание строк, которое и будет нумероваться функцией ROW\_NUMBER.

[SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) `city`, `sname`, ROW\_NUMBER() OVER() AS number FROM sal



[SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) `sname`, `city`, ROW\_NUMBER() OVER() AS number FROM sal

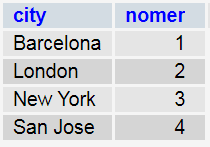


Пример 3: Выберем имеющиеся в таблице города (т.е вывести города исключая их повторение) и пронумеруем эту уникальность.

// Подзапрос, который после FROM выберет се города без повторений и передаст их в основной запрос, функция даст выбранным столбцам сквозную нумерацию. //

[SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) `city`, ROW\_NUMBER() OVER()as nomer

FROM ([SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) DISTINCT(`city`) from sal) as CITY



2) Функция **PARTITION BY**

Функция **PARTITION BY** определяет столбец, по которому будет производиться группировка строк по значению определённого столбца, и он является ключевым в разбиении набора строк на окна. Это полезно, если данные логически делятся на какие-то категории и нужно что-то сделать с данной строкой с учётом других строк той же группы (скажем, сравнить теннисиста с остальными теннисистами, но не с бегунами или пловцами). Этот оператор работает только с оконными функциями типа LAG, LEAD, RANK и т. д.

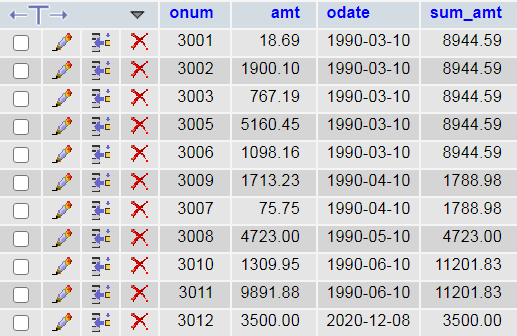
Пример 4: Возьмём задание из примера 1 и расширим условие.

Показать общую сумму заказов по каждой дате не исключая вывода номера заказа, его стоимости и даты.

// Вывелась вся исходная информация из указанных столбцов таблицы с созданием окна sum\_amt (используя over), содержащего сумму заказов по каждой дате (prtition by сгруппировало строки по odate), имеющейся в таблице. //

SELECT onum, amt, odate, SUM(amt) over(PARTITION BY odate) as sum\_amt

FROM orders



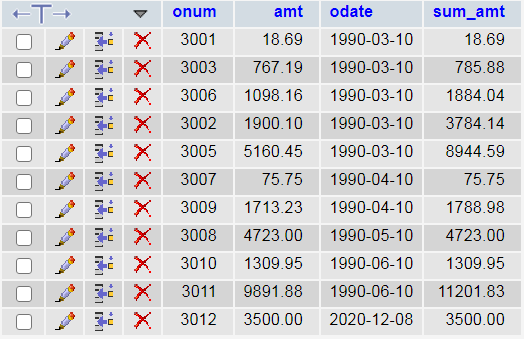
Оператор ORDER BY в OVER будет вносить изменения следующего вида как в примере 5 (продолжаем модифицировать пример 4).

Пример 5: Показать общую сумму заказов по каждой дате не исключая вывода номера заказа, его стоимости и даты. При этом показывать поэтапное суммирование стоимости заказа в каждой группе – это называется нарастающий/накопительный итог. // Сравните результаты работы запроса 4 и 5 примеров //

SELECT `onum`, `amt`, `odate`,

SUM(`amt`) over (PARTITION BY `odate` ORDER BY `amt`) as sum\_amt

FROM `orders`



3) Функция **ROWS**

Еще две функции ROWS и RANGE используются вместе с ORDER BY и применяются в инструкции OVER

Функция **ROWS** ограничивает строки в окне, указывая фиксированное количество строк, предшествующих или следующих за текущей.

Предложение ROWS может быть задано с помощью методов:

* CURRENT ROW – отображение текущей строки;
* UNBOUNDED FOLLOWING – все записи после текущей;
* UNBOUNDED PRECEDING – все предыдущие записи;
* <целое число> PRECEDING – заданное число предыдущих строк;
* <целое число> FOLLOWING – заданное число последующих записей.

Вы можете комбинировать эти функции для достижения желаемого результата.

Пример 6 (модифицируем пример 5): Показать общую сумму заказов по каждой дате не исключая вывода номера заказа, его стоимости и даты. При этом показывать суммирование текущей записи и одной (1) следующей записи продвигаясь по всей группе конкретной даты и так по всем группам дат.

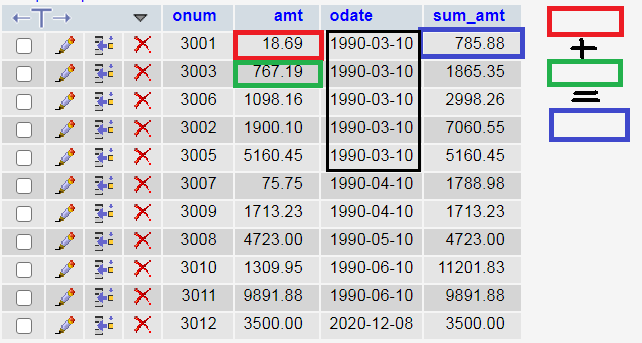
// Сравните результаты работы запроса 4 и 5 примеров //

SELECT `onum`, `amt`, `odate`,

SUM(`amt`) over(PARTITION BY `odate` ORDER BY `amt`

ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) as sum\_amt

FROM`orders`



4) Функция **RANGE**

Функция **RANGE** предназначено также для ограничения набора строк. В отличие от ROWS, оно работает не с физическими строками, а с диапазоном строк в предложении ORDER BY. Это означает, что одинаковые по рангу строки в контексте предложения ORDER BYбудут считаться как одна текущая строка для функции CURRENT ROW**.** А в предложении ROWSтекущая строка – это одна, текущая строка набора данных.

Предложение RANGE может использовать опции:

* CURRENT ROW – отображение текущей строки;
* UNBOUNDED FOLLOWING – все записи после текущей;
* UNBOUNDED PRECEDING – все предыдущие записи.

И не может:

* <целое число> PRECEDING – заданное число предыдущих строк;
* <целое число> FOLLOWING – заданное число последующих записей.

5) Функция **LAG | LEAD**

Функция **LAG** берёт строку и возвращает ту, которая шла перед текущей.

Функция **LEAD** берёт строку и возвращает ту, которая идёт после текущей.

**LAG | LEAD** **(< скалярное выражение >**

[,< сдвиг >] [, < значение по умолчанию >]**)**

**OVER** ( [< предложение partition BY >] < предложение ORDER BY > )

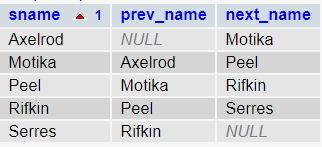
Пример 7: Вывести имена продавцов, с указанием имен тех, кто в таблице находится до и после текущего продавца.

SELECT `sname`,

LAG(`sname`) OVER(ORDER BY `sname`) prev\_name,

LEAD(`sname`) OVER(ORDER BY `sname`) next\_name

FROM `sal



6) Функция **WITH**

Функция **WITH** используется для определения CTE, его можно использовать как в обычных запросах, так и в функциях, хранимых процедурах, триггерах и представлениях.

Общее табличное выражение (CTE) – важная функция MySQL, которая используется для генерации временного набора результатов, который не сохраняется в базе данных в виде объектов, но к нему можно обращаться. Его можно использовать с любым оператором SQL, таким как SELECT, INSERT, UPDATE и так далее.

Сложные запросы можно упростить с помощью CTE. Набор результатов любого запроса сохраняется как объект для производной таблицы во время выполнения запроса. Но CTE может ссылаться на себя, что означает, что на один и тот же запрос можно ссылаться несколько раз с помощью CTE. По этой причине производительность CTE лучше, чем у производной таблицы.

Главной особенностью СТЕ является то, что с помощью них можно писать **рекурсивные запросы**.

Назначение СТЕ:

* основной целью СТЕ является написание рекурсивных запросов, можно сказать для этого они, и были созданы;
* СТЕ можно использовать также и для замены представлений (VIEW), например, в тех случаях, когда нет необходимости сохранять в базе SQL запрос представления, т.е. его определение;
* СТЕ повышают читаемость кода путем разделения запроса на логические блоки, и тем самым упрощают работу со сложными запросами;
* СТЕ предназначены и для многократных ссылок на результирующий набор из одной и той же SQL инструкции.

Синтаксис:

**WITH [** ( column\_name [ ,...n ] ) **]**

**AS**

**( CTE\_query\_definition )**

где: **common\_table\_expression\_name** – это псевдоним или можно сказать идентификатор обобщенного табличного выражения;

**column\_name** – имя столбца, который будет определен в обобщенном табличном выражении. Количество столбцов должно совпадать с количеством столбцов возвращаемых запросом CTE\_query\_definition. Указывать имена столбцов необязательно, но только в том случае, если всем столбцам в запросе CTE\_query\_definition присвоены уникальные псевдонимы;

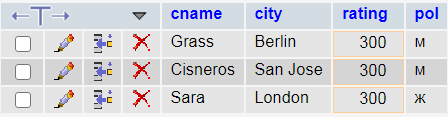
**CTE\_query\_definition** — запрос SELECT, к результирующему набору которого, мы и будем обращаться через обобщенное табличное выражение, т.е. common\_table\_expression\_name.

После обобщенного табличного выражения, т.е. сразу за ним должен идти одиночный запрос SELECT, INSERT, UPDATE, MERGE или DELETE.

Пример 8: Вывести всех заказчиков с рейтингом 300 (мужчины и женщины отдельно) и для каждого вывести заказчика с предыдущим рейтингом.

На первом этапе найдём всех заказчиков с рейтингом 300.

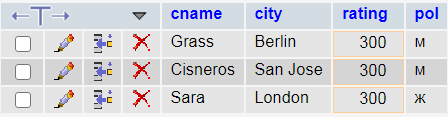
SELECT cname,city,rating, pol FROM cust WHERE rating=300



Результат запроса нам нужен для дальнейшей работы, поэтому далее, используя WITH, создадим табличное выражение (временную, но самостоятельную таблицу) «cust300», с которой будет работать основной запрос.

WITH cust300 as

(SELECT cname,city,rating, pol FROM cust WHERE rating=300)



Для того, что бы вывести имя предыдущего заказчика (cname) от текущего используем функцию LAG. Для того, что бы разделить выбранных заказчиков по полу (pol) используем PARTITION BY, а для упорядочивание по полу используем ORDER BY. В итоге получаем строку

LAG(`cname`) OVER(PARTITION BY pol  ORDER BY ` pol  ` ASC

Теперь собираем весь запрос.

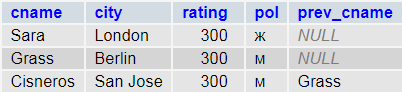
WITH cust300  as

([SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) cname,city,rating, pol FROM cust WHERE rating=300)

[SELECT](http://127.0.0.1/openserver/phpmyadmin/url.php?url=https://dev.mysql.com/doc/refman/5.5/en/select.html) cname,city,rating, pol,

LAG(`cname`) OVER(PARTITION BY pol  ORDER BY pol ASC) AS prev\_cname

FROM cust300 ORDER BY `rating`



// В таблице cust300 перед Sara есть Cisneros, но он не выведен (в ячейке значение NULL), т.к. он мужского пола. Перед Grass так же никого нет, т.к. он первый в таблице. //

**Практическое задание:**

1) Показать самые большие комиссионные продавцов в каждом город, не исключая вывода исходных данных о имени продавца, его исходных комиссионных и городе его проживания. Столбец с результатом назовите 'Максимальная комиссия в каждом городе'. Пронумеруйте уникальность строк, для результата используйте столбец 'Номер уникальности', который разместите перед столбцом со значением максимальной комиссии.

2) Покажите суммарный накопительный итог всех оформленных заказов для каждого продавца. В выводе должны участвовать столбцы: дата оформления заказа, стоимость заказа, имя(!!!) продавца, 'Накопительная сумма стоимости заказов'.

3) Показать максимальную стоимость заказа для каждого заказчика и заказ, стоимость которого предшествовала максимальному. В выводе должны участвовать столбцы: имя(!!!) продавца, стоимость заказа, 'Максимальная стоимость заказа для каждого заказчика', 'Стоимость заказа, предшествующая максимальной для данного Заказчика'.

4) Показать максимальную суммарную стоимость заказа Заказчиков (т.е по каждому заказчику найти его суммарную стоимость заказов и среди этих суммарных стоимостей найти максимальную).

4.1) Показать имя Заказчика у которого оказалась эта максимальная суммарная стоимость заказов.

4.2) Показать имя Заказчика который на втором месте по максимальной суммарной стоимости заказов. Т.е. дополнить информацию по запросу 4.1 – к выведенной информации по лидеру, добавить информацию по второму месту.