



Курс базы данных и SQL. Лекция 5



На пятой лекции будем рассматривать: оконные функции(агрегирующие, ранжирующие, функции смещения, аналитические функции) и узнаем про представления

Терминология

Оконная функция в SQL - функция, которая работает с выделенным набором строк (окном, партицией) и выполняет вычисление для этого набора строк в отдельном столбце.

Партиции (окна из набора строк) - это набор строк, указанный для оконной функции по одному из столбцов или группе столбцов таблицы. Партиции для каждой оконной функции в запросе могут быть разделены по различным колонкам таблицы.

Агрегатные функции (агрегации) — это функции, которые вычисляются от группы значений и объединяют их в одно результирующее.

Функции смещения – это функции, которые позволяют перемещаться и обращаться к разным строкам в окне, относительно текущей строки, а также обращаться к значениям в начале или в конце окна.

Аналитические функции — это функции которые возвращают информацию о распределении данных и используются для статистического анализа.

Предикат в SQL это:

в широком смысле, — любое выражение, результатом которого являются значения булевого типа — TRUE, FALSE, а так же UNKNOWN

в узком смысле, — некий уточняющий фильтр. Самым явным примером предиката служит оператор WHERE

Что мы узнаем:

- Оконные функции
- Ранжирующие оконные функции
- Агрегирующие оконные функции
- Функции смещения в оконных функциях
- Арифметические операции
- Логические операторы (and, or, between, not, in)
- Оператор CASE, IF

Доброго времени суток, уважаемые студенты!

На сегодняшней лекции особое внимание хотелось бы уделить мощному инструменту аналитика - оконным функциям. “Окошки” помогают делать различные аналитические отчеты без участия стороннего ПО (“экселя”). Давайте попробуем рассмотреть основную массу задач, которые могут решать оконные функции, затем - разберемся, как же воплотить наши ожидания с помощью кода:

- Ранжирование. Составление всевозможных рейтингов: топ-100 популярных треков на какой-то радиостанции
- Просмотр разницы между близкими элементами (2 и 3 человек в топ-5 самых богатых людей мира)
- “Скользящие” агрегаты
- Агрегация (топ самых высоких зарплат)

Классы оконных функций

Множество оконных функций можно поделить на 3 группы:

- Агрегирующие (Aggregate)
- Ранжирующие (Ranking)
- Функции смещения (Value)
- Аналитические функции.

Составление рейтинга по ЗП

```
-- Создание таблицы  
CREATE TABLE IF NOT EXISTS staff
```

```
(
    id INT PRIMARY KEY,
    first_name VARCHAR(30),
    post VARCHAR(30),
    discipline VARCHAR(30),
    salary INT
);

-- Заполнение таблицы данными
INSERT staff (id, first_name, post, discipline, salary)
VALUES
    (100, 'Антон', 'Преподаватель', 'Программирование', 50),
    (101, 'Василий', 'Преподаватель', 'Программирование', 60),
    (103, 'Александр', 'Ассистент', 'Программирование', 25),
    (104, 'Владимир', 'Профессор', 'Математика', 120),
    (105, 'Иван', 'Профессор', 'Математика', 120),
    (106, 'Михаил', 'Доцент', 'Физика', 70),
    (107, 'Анна', 'Доцент', 'Физика', 70),
    (108, 'Вероника', 'Доцент', 'ИКТ', 30),
    (109, 'Григорий', 'Преподаватель', 'ИКТ', 25),
    (110, 'Георгий', 'Ассистент', 'Программирование', 30);
```

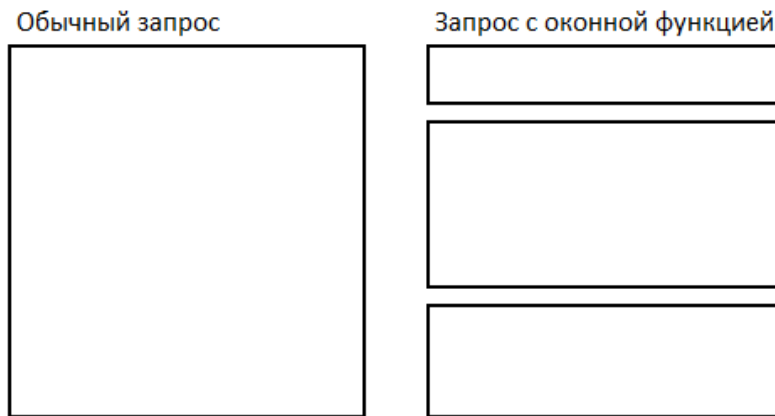
	id	first_name	post	discipline	salary
►	100	Антон	Преподаватель	Программирование	50
	101	Василий	Преподаватель	Программирование	60
	103	Александр	Ассистент	Программирование	25
	104	Владимир	Профессор	Математика	120
	105	Иван	Профессор	Математика	120
	106	Михаил	Доцент	Физика	70
	107	Анна	Доцент	Физика	70
	108	Вероника	Доцент	ИКТ	30
	109	Григорий	Преподаватель	ИКТ	25
	110	Георгий	Ассистент	Программирование	30

Как мы видим, одинаковый ранг будет у сотрудников с одинаковой ЗП (допусти, Владимир и Иван в правой таблице, Иван и Михаил, Вероника и Григорий).

	rank	first_name	discipline	salary
►	6	Владимир	Математика	120
	6	Иван	Математика	120
	5	Михаил	Физика	70
	5	Анна	Физика	70
	4	Василий	Программирование	60
	3	Антон	Программирование	50
	2	Вероника	ИКТ	30
	2	Георгий	Программирование	30
	1	Александр	Программирование	25
	1	Григорий	ИКТ	25

Давайте попробуем пройти по всем строчкам. Начнем с 1 и будем проставлять ранг. Для простановки ранга необходимо учитывать только значение конкретного столбца - **salary**. В контексте оконной функции - окно. При обычном запросе все множество строк обрабатывается единым потоком, для которого считаются агрегаты. При работе с оконными функциями наш запрос

делится на небольшие части (окна) и уже для каждой из отдельных частей считаются свои агрегаты.



Пример для абстрактной таблицы:

Исходная таблица				Оконные функции			Окна
ID	PRODUCT	TYPE	PRICE	Пример оконных функций			
				func_sum	func_count	func_row_number	
1	Монитор LG	Монитор	2,0	2,0	1	1	окно 1 (partition)
2	Ноутбук HP	Ноутбук	4,0	8,0	2	1	окно 2 (partition)
3	Ноутбук SONY	Ноутбук	4,0	8,0	2	2	
4	Принтер Xerox	Принтер	3,0	9,0	3	1	окно 3 (partition)
5	Принтер Canon	Принтер	3,0	9,0	3	2	
6	Принтер Brother	Принтер	3,0	9,0	3	3	
7	Смартфон Huawei	Смартфон	2,0	8,0	4	1	окно 4 (partition)
8	Смартфон Samsung	Смартфон	2,0	8,0	4	2	
9	Смартфон Apple	Смартфон	2,0	8,0	4	3	
10	Смартфон Xiaomi	Смартфон	2,0	8,0	4	4	

Наше окно - содержимое столбца salary, которое мы отсортировали по убыванию.

Как же записать это в контексте SQL?

MySQL :: MySQL 8.0 Reference Manual :: 12.21.2 Window Function Concepts and Syntax

<https://dev.mysql.com/doc/refman/8.0/en/window-functions-usage.html>

Для открытия окна используется обязательная инструкция OVER:

```
SELECT Название_функции (столбец для вычислений)
OVER
```

```
(  
    PARTITION BY столбец для группировки  
    ORDER BY столбец для сортировки  
    ROWS или RANGE выражение для ограничения строк в пределах группы  
)
```

Нашему окну можно так же присвоить имя, которое можно будет использовать в запросе. Наше окно должно проставлять ранг по убыванию столбца `salary`:

```
WINDOW w AS (ORDER BY salary desc)
```

- `window` — определение окна;
- `w` — название окна, указываем через пробел, как и псевдоним без AS;
- `order by salary desc` — сортировка столбца `salary`.

Внутри конкретного окна нужно посчитать ранг (ранг по окну `w`).

Пусть имеется таблица:

	shopping_day	department	count
►	2022-12-21	Бытовая техника	3
	2022-12-21	Свежая выпечка	4
	2022-12-21	Напитки	4
	2022-12-22	Бытовая техника	1
	2022-12-22	Свежая выпечка	3
	2022-12-22	Напитки	3
	2022-12-22	Замороженные продукты	1
	2022-12-23	Бытовая техника	2
	2022-12-23	Свежая выпечка	4
	2022-12-23	Замороженные продукты	4

Начнем с примера: откроем окно `over` и найдем сумму и количество покупок:

```
SELECT shopping_day, department, count,  
       SUM(count) OVER() AS 'Sum',  
       COUNT(count) OVER() AS 'Count'  
FROM shop;
```

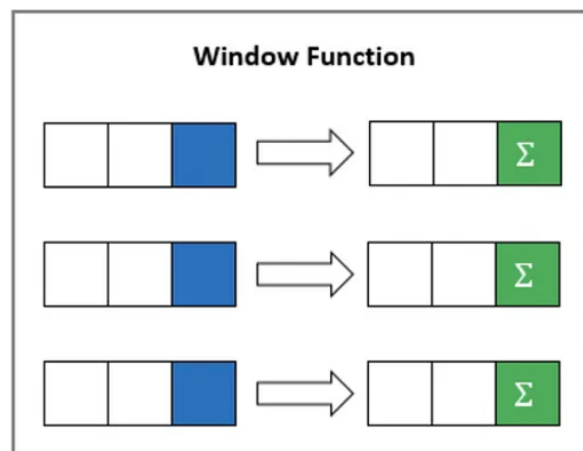
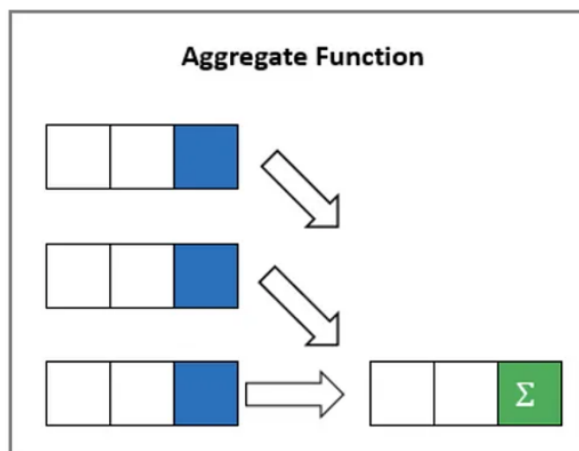
	shopping_day	department	count	Sum	Count
►	2022-12-21	Бытовая техника	3	29	10
	2022-12-21	Свежая выпечка	4	29	10
	2022-12-21	Напитки	4	29	10
	2022-12-22	Бытовая техника	1	29	10
	2022-12-22	Свежая выпечка	3	29	10
	2022-12-22	Напитки	3	29	10
	2022-12-22	Замороженные продукты	1	29	10
	2022-12-23	Бытовая техника	2	29	10
	2022-12-23	Свежая выпечка	4	29	10
	2022-12-23	Замороженные продукты	4	29	10

В нашем примере сумма, равная 29 - это общее количество покупок(суммирование всего столбца `count` и подсчет общего количества строк). Если бы мы использовали агрегатную функцию в чистом виде, то получим сужение выборки до 1 строчки.

В оконной функции исходное количество строк не уменьшается по сравнению с исходной таблицей. При использовании агрегирующих функций предложение `GROUP BY` сокращает количество строк в запросе с помощью их группировки.

```
SELECT
  SUM(count) AS 'Sum',
  COUNT(count) AS 'Count'
FROM shop;
```

	Sum	Count
►	29	10



Внутри окна так же возможно применить сортировку и группировку.

- **Группировка**

Для группировки используется `PARTITION BY`, которая определяет столбец для группировки в окне:

	shopping_day	department	count
►	2022-12-21	Бытовая техника	3
	2022-12-21	Свежая выпечка	4
	2022-12-21	Напитки	4
	2022-12-22	Бытовая техника	1
	2022-12-22	Свежая выпечка	3
	2022-12-22	Напитки	3
	2022-12-22	Замороженные продукты	1
	2022-12-23	Бытовая техника	2
	2022-12-23	Свежая выпечка	4
	2022-12-23	Замороженные продукты	4

```
SELECT shopping_day, department, count,
       SUM(count) OVER(PARTITION BY shopping_day) AS 'Sum'
FROM shop;
```

	shopping_day	department	count	Sum
►	2022-12-21	Бытовая техника	3	11
	2022-12-21	Свежая выпечка	4	11
	2022-12-21	Напитки	4	11
	2022-12-22	Бытовая техника	1	8
	2022-12-22	Свежая выпечка	3	8
	2022-12-22	Напитки	3	8
	2022-12-22	Замороженные продукты	1	8
	2022-12-23	Бытовая техника	2	10
	2022-12-23	Свежая выпечка	4	10
	2022-12-23	Замороженные продукты	4	10

Инструкция **PARTITION BY** сгруппировала строки по полю «**shopping_day**» и каждой группы рассчитывается сумма значений по столбцу «**count**».

- **Сортировка**

	shopping_day	department	count
►	2022-12-21	Бытовая техника	3
	2022-12-21	Свежая выпечка	4
	2022-12-21	Напитки	4
	2022-12-22	Бытовая техника	1
	2022-12-22	Свежая выпечка	3
	2022-12-22	Напитки	3
	2022-12-22	Замороженные продукты	1
	2022-12-23	Бытовая техника	2
	2022-12-23	Свежая выпечка	4
	2022-12-23	Замороженные продукты	4

Добавим к **PARTITION BY** сортировку по столбцу «**department**» (**ORDER BY department**). В данной ситуации мы будем считать нарастающий итог: для каждого значения «**count**» мы ищем сумму всех значений с предыдущими.

```
SELECT shopping_day, department, count,
       SUM(count) OVER(PARTITION BY shopping_day ORDER BY department) AS 'Sum'
FROM shop;
```

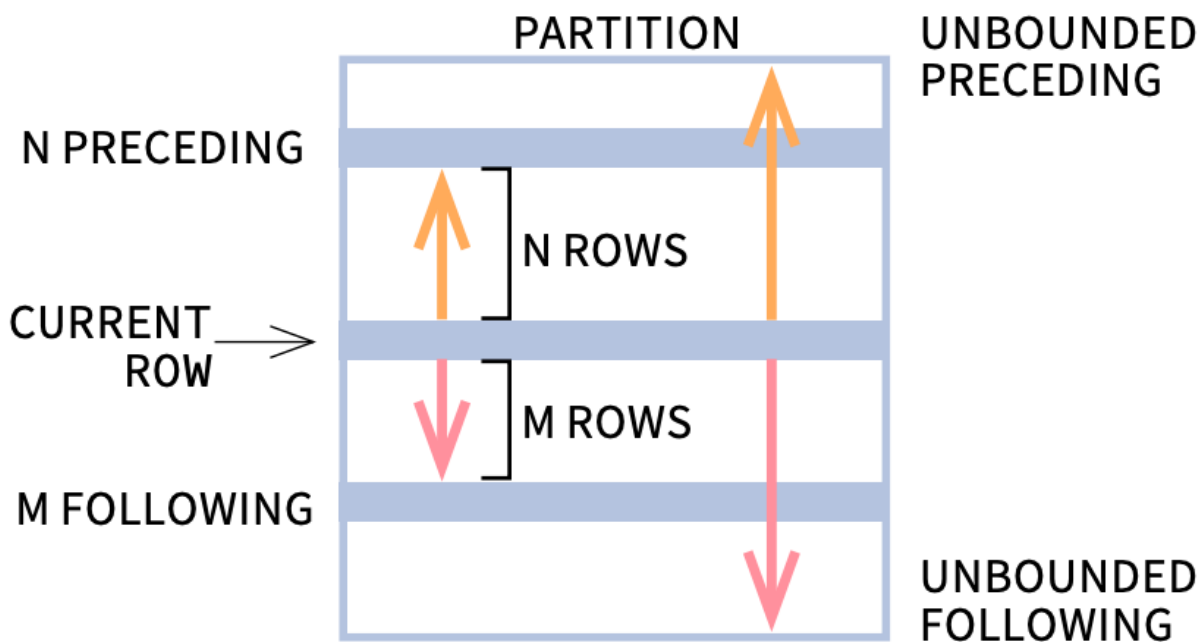
	shopping_day	department	count	Sum
▶	2022-12-21	Бытовая техника	3	3
	2022-12-21	Напитки	4	7
	2022-12-21	Свежая выпечка	4	11
	2022-12-22	Бытовая техника	1	1
	2022-12-22	Замороженные продукты	1	2
	2022-12-22	Напитки	3	5
	2022-12-22	Свежая выпечка	3	8
	2022-12-23	Бытовая техника	2	2
	2022-12-23	Замороженные продукты	4	6
	2022-12-23	Свежая выпечка	4	10

И вишенка на тортике - ограничение строк в окне :)

Инструкция `rows` ограничивает строки в окне, указывается определенное количество строк, предшествующих или следующих за текущей.

Инструкция `range`, в отличие от `rows`, работает не со строками, а с диапазоном строк в инструкции ORDER BY.

Обе инструкции `rows` и `range` всегда используются вместе с ORDER BY.



MySQL :: MySQL 8.0 Reference Manual :: 12.21.3 Window Function Frame Specification

<https://dev.mysql.com/doc/refman/8.0/en/window-functions-frames.html>

- **UNBOUNDED PRECEDING** – указывает, что окно начинается с первой строки группы;

- **n PRECEDING** – n строк перед текущей строкой.
- **CURRENT ROW** – окно начинается или заканчивается на текущей строке;
- **n FOLLOWING** – определяет число строк после текущей строки (не допускается в предложении RANGE). n строк после текущей строки.
- **UNBOUNDED FOLLOWING** – С помощью данной инструкции можно указать, что окно заканчивается на последней строке группы

Сумма рассчитывается по текущей и следующей ячейке в окне. А последняя строка в окне имеет то же значение, что и столбец «count»: больше не с чем складывать.

```
SELECT shopping_day, department, count,
       SUM(count) OVER(PARTITION BY shopping_day ORDER BY count ROWS BETWEEN CURRENT ROW AND 1 FOLLOWING) AS 'Sum'
FROM shop;
```

	shopping_day	department	count	Sum	
▶	2022-12-21	Бытовая техника	3	7	=3+4
	2022-12-21	Свежая выпечка	4	8	=4+4
	2022-12-21	Напитки	4	4	=4
	2022-12-22	Бытовая техника	1	2	
	2022-12-22	Замороженные продукты	1	4	
	2022-12-22	Свежая выпечка	3	6	
	2022-12-22	Напитки	3	3	
	2022-12-23	Бытовая техника	2	6	
	2022-12-23	Свежая выпечка	4	8	
	2022-12-23	Замороженные продукты	4	4	

Ранжирующие функции

Эти функции позволяют использовать для указания ранга или порядкового номера, который у нас на руке 😊

- **row_number** – задаем номер строки, используется для нумерации;
- **rank** – функция возвращает ранг каждой строки, в случае нахождения одинаковых, возвращается одинаковый ранг с пропуском следующего значения;
- **dense_rank** – функция возвращает так же ранг каждой строки. Но в отличие от функции **rank**, она для одинаковых значений возвращает ранг, не пропуская следующий;
- **ntitle** – это функция, которая позволяет поделить вашу выборку на группы, количество групп задается в скобках.

Пример для тестовой таблицы:

	shopping_day	department	count
►	2022-12-21	Бытовая техника	3
	2022-12-21	Свежая выпечка	4
	2022-12-21	Напитки	4
	2022-12-22	Бытовая техника	1
	2022-12-22	Свежая выпечка	3
	2022-12-22	Напитки	3
	2022-12-22	Замороженные продукты	1
	2022-12-23	Бытовая техника	2
	2022-12-23	Свежая выпечка	4
	2022-12-23	Замороженные продукты	4

```
SELECT shopping_day, department, count,
       ROW_NUMBER() OVER(PARTITION BY shopping_day ORDER BY count) AS 'Row_number',
       RANK() OVER(PARTITION BY shopping_day ORDER BY count) AS 'Rank' ,
       DENSE_RANK() OVER(PARTITION BY shopping_day ORDER BY count) AS 'Dense_Rank' ,
       NTILE(3) OVER(PARTITION BY shopping_day ORDER BY count) AS 'Ntile'
FROM shop;
```

	shopping_day	department	count	Row_number	Rank	Dense_Rank	Ntile
►	2022-12-21	Бытовая техника	3	1	1	1	1
	2022-12-21	Свежая выпечка	4	2	2	2	2
	2022-12-21	Напитки	4	3	2	2	3
	2022-12-22	Бытовая техника	1	1	1	1	1
	2022-12-22	Замороженные продукты	1	2	1	1	1
	2022-12-22	Свежая выпечка	3	3	3	2	2
	2022-12-22	Напитки	3	4	3	2	3
	2022-12-23	Бытовая техника	2	1	1	1	1
	2022-12-23	Свежая выпечка	4	2	2	2	2
	2022-12-23	Замороженные продукты	4	3	2	2	3

Вернемся к задаче: проставить ранги по убыванию ЗП (самая маленькая ЗП - самый маленький ранг).

Для подсчета ранга в нашем случае используем функцию `dense_rank()`. В нашем случае пропуск ранга не нужен: начинаем с 1 и увеличиваем ранг при отличии от предыдущего. Слово `rank` выделяем в экранированные кавычки: название столбца совпадает с функцией ранжирования, ``rank`` будет считаться как имя столбца:

```
SELECT
  DENSE_RANK() OVER w AS `rank`, -- w = OVER(ORDER BY salary DESC)
  first_name, discipline,
  salary
FROM staff
WINDOW w AS (ORDER BY salary DESC)
ORDER BY `rank`, id;
```

	rank	first_name	discipline	salary
▶	6	Владимир	Математика	120
	6	Иван	Математика	120
	5	Михаил	Физика	70
	5	Анна	Физика	70
	4	Василий	Программирование	60
	3	Антон	Программирование	50
	2	Вероника	ИКТ	30
	2	Георгий	Программирование	30
	1	Александр	Программирование	25
	1	Григорий	ИКТ	25

Обратите внимание, что `order by` в окне задает сортировку окна, а `order by` в основном запросе — сортировку результатов всего запроса после обработки окна. Пусть целью моего запроса будет проставить ранг по возрастанию зарплаты, а сортировка - по убыванию:

```
SELECT
  DENSE_RANK() OVER W AS `rank`,
  first_name, discipline,
  salary
FROM staff
WINDOW w AS (ORDER BY salary ASC)
ORDER BY salary DESC;
```

	rank	first_name	discipline	salary
▶	6	Владимир	Математика	120
	6	Иван	Математика	120
	5	Михаил	Физика	70
	5	Анна	Физика	70
	4	Василий	Программирование	60
	3	Антон	Программирование	50
	2	Вероника	ИКТ	30
	2	Георгий	Программирование	30
	1	Александр	Программирование	25
	1	Григорий	ИКТ	25

Агрегирующие оконные функции

Следующий вид оконных функций - агрегирующие. Они используются для учета средней зарплаты, подсчета количества сотрудников, максимальную и минимальную ЗП и сумму по чеку. Эти функции выполняют манипуляции с набором данных и возвращают итоговое значение.

<code>sum()</code>	сумма значений в столбце;
<code>count()</code>	количество значений в столбце (<i>NULL не учитываются</i>);
<code>avg()</code>	среднее значение в столбце
<code>max()</code>	определяет максимальное значение в столбце;
<code>min()</code>	определяет минимальное значение в столбце.

	shopping_day	department	count
►	2022-12-21	Бытовая техника	3
	2022-12-21	Свежая выпечка	4
	2022-12-21	Напитки	4
	2022-12-22	Бытовая техника	1
	2022-12-22	Свежая выпечка	3
	2022-12-22	Напитки	3
	2022-12-22	Замороженные продукты	1
	2022-12-23	Бытовая техника	2
	2022-12-23	Свежая выпечка	4
	2022-12-23	Замороженные продукты	4

```
SELECT shopping_day, department, count,
       SUM(count) OVER(PARTITION BY shopping_day) AS 'Sum' ,
       COUNT(count) OVER(PARTITION BY shopping_day) AS 'Count' ,
       AVG(count) OVER(PARTITION BY shopping_day) AS 'Avg' ,
       MAX(count) OVER(PARTITION BY shopping_day) AS 'Max' ,
       MIN(count) OVER(PARTITION BY shopping_day) AS 'Min'
FROM shop;
```

	shopping_day	department	count	Sum	Count	Avg	Max	Min
►	2022-12-21	Бытовая техника	3	11	3	3.6667	4	3
	2022-12-21	Свежая выпечка	4	11	3	3.6667	4	3
	2022-12-21	Напитки	4	11	3	3.6667	4	3
	2022-12-22	Бытовая техника	1	8	4	2.0000	3	1
	2022-12-22	Свежая выпечка	3	8	4	2.0000	3	1
	2022-12-22	Напитки	3	8	4	2.0000	3	1
	2022-12-22	Замороженные продукты	1	8	4	2.0000	3	1
	2022-12-23	Бытовая техника	2	10	3	3.3333	4	2
	2022-12-23	Свежая выпечка	4	10	3	3.3333	4	2
	2022-12-23	Замороженные продукты	4	10	3	3.3333	4	2

Давайте попробуем применить знания на практике. Каждая группа преподавателей по конкретному предмету получают зарплату. Давайте узнаем, какой процент от суммарной ЗП по отделу получает каждый педагог.

	id	first_name	post	discipline	salary
►	100	Антон	Преподаватель	Программирование	50
	101	Василий	Преподаватель	Программирование	60
	103	Александр	Ассистент	Программирование	25
	104	Владимир	Профессор	Математика	120
	105	Иван	Профессор	Математика	120
	106	Михаил	Доцент	Физика	70
	107	Анна	Доцент	Физика	70
	108	Вероника	Доцент	ИКТ	30
	109	Григорий	Преподаватель	ИКТ	25
	110	Георгий	Ассистент	Программирование	30

```
SELECT
  first_name, discipline, salary,
  SUM(salary) OVER w AS payment_fund,
  ROUND(salary * 100.0 / SUM(salary) OVER w) AS percentage
FROM staff
```

```
WINDOW w AS (PARTITION BY discipline)
ORDER BY discipline, salary, id;
```

Пройдемся по всей таблице. Считаем:

- `payment_fund` — показывает суммарную ЗП отдела (она одинакова для всех сотрудников департамента);
- `percentage` — процент зарплаты от этой суммы.

Окно состоит из секций по дисциплинам:

```
WINDOW w AS (PARTITION BY discipline)
```

Чтобы найти суммарную зарплату, мы используем функцию агрегатную функцию `SUM(salary)` по столбцу `salary`. Для расчета процента ЗП будем использовать формулу: `salary/sum(salary)`

	first_name	discipline	salary	payment_fund	percentage
▶	Григорий	ИКТ	25	55	45
	Вероника	ИКТ	30	55	55
	Владимир	Математика	120	240	50
	Иван	Математика	120	240	50
	Александр	Программирование	25	165	15
	Георгий	Программирование	30	165	18
	Антон	Программирование	50	165	30
	Василий	Программирование	60	165	36
	Михаил	Физика	70	140	50
	Анна	Физика	70	140	50

Функции смещения

Это функции, которые позволяют перемещаться и обращаться к разным строкам в окне, относительно текущей строки, а также обращаться к значениям в начале или в конце окна.

<code>lag(value)</code>	функция <code>lag()</code> обращается к данным из предыдущей строки окна	
<code>lead(value)</code>	функция <code>lead()</code> обращается к данным из следующей строки	
<code>first_value()</code>	с помощью функции <code>first_value()</code> можно получить первое значение в окне, параметр - столбец, который нужно вернуть	
<code>last_value()</code>	с помощью функции <code>last_value()</code> можно получить последнее значение в окне, параметр - столбец, который нужно вернуть	

```
SELECT shopping_day, department, count,
       LAG(count) OVER(PARTITION BY shopping_day ORDER BY shopping_day) AS 'Lag' ,
       LEAD(count) OVER(PARTITION BY shopping_day ORDER BY shopping_day) AS 'Lead' ,
```


```
FIRST_VALUE(count) OVER(PARTITION BY shopping_day ORDER BY shopping_day) AS 'First_Value' ,
LAST_VALUE(count) OVER(PARTITION BY shopping_day ORDER BY shopping_day) AS 'Last_Value'
FROM shop;
```

	shopping_day	department	count	Lag	Lead	First_Value	Last_Value
►	2022-12-21	Бытовая техника	3	NULL	4	3	4
	2022-12-21	Свежая выпечка	4	3	4	3	4
	2022-12-21	Напитки	4	4	NULL	3	4
	2022-12-22	Бытовая техника	1	NULL	3	1	1
	2022-12-22	Свежая выпечка	3	1	3	1	1
	2022-12-22	Напитки	3	3	1	1	1
	2022-12-22	Замороженные продукты	1	3	NULL	1	1
	2022-12-23	Бытовая техника	2	NULL	4	2	4
	2022-12-23	Свежая выпечка	4	2	4	2	4
	2022-12-23	Замороженные продукты	4	4	NULL	2	4

Аналитические функции

Аналитические функции — это функции которые возвращают информацию о распределении данных и используются для статистического анализа. Показывать математику без уточнения деталей будет не очень, уместно:

MySQL :: MySQL 8.0 Reference Manual :: 12.21.1 Window Function Descriptions

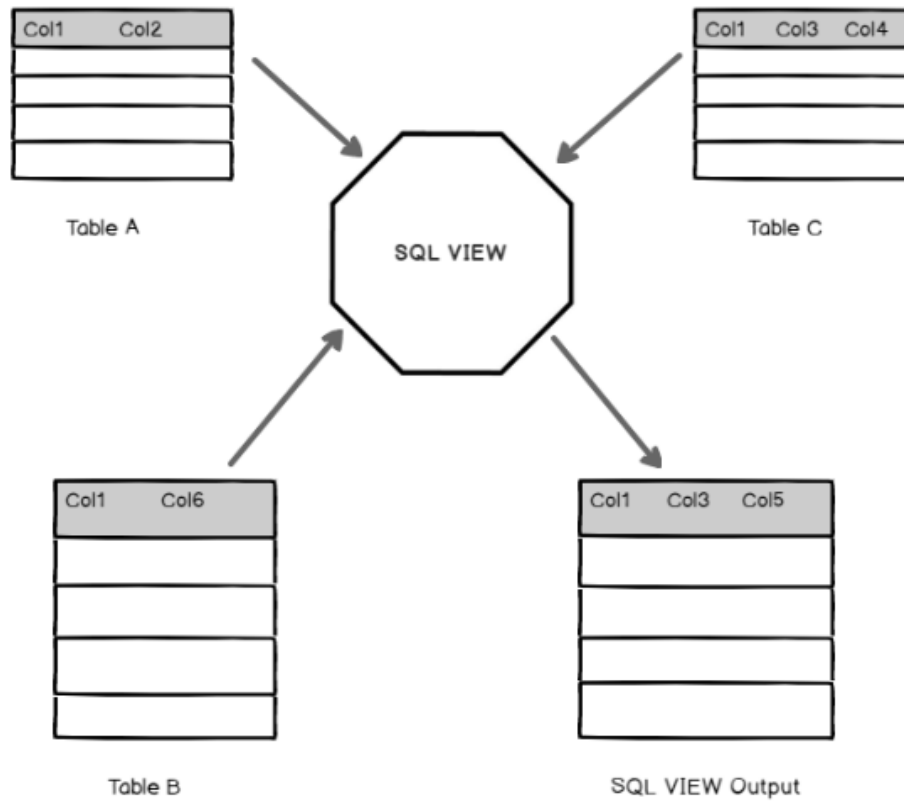
 <https://dev.mysql.com/doc/refman/8.0/en/window-function-descriptions.html?ff=nopfls>

Представления (VIEW)

MySQL :: MySQL 8.0 Reference Manual :: 25.5 Using Views

 <https://dev.mysql.com/doc/refman/8.0/en/views.html>

При работе с БД требуется многократно запускать запросы, которые могут быть сложными и требовать обращения к нескольким таблицам. Чтобы не использовать многократно запросы, имеет смысл обратиться к так называемым представлениям (view). Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению. Представления иногда называют «виртуальными таблицами».



Синтаксис представления:

```

CREATE VIEW name_view
AS
(
  ...
  query
  ...
);
  
```

В этой команде обязательно имя представления и сам запрос к БД. Попробуем реализовать простое представление. Попробуем реализовать запрос, который выводит дисциплины и количество преподавателей, которые эти предметы ведут:

```

SELECT
  discipline, count(first_name)
FROM staff
GROUP BY discipline
ORDER BY count(first_name) DESC;
  
```

Сделаем этот запрос представлением:

```

CREATE OR REPLACE VIEW count_teacher AS
SELECT
  
```

```

discipline, count(first_name) AS res
FROM staff
GROUP BY discipline
ORDER BY count(first_name) DESC;
-- "OR REPLACE" заменяет представление, если оно существует

```

Теперь вместо указания какого - либо сложного запроса можно вызвать представление:

```
SELECT * FROM count_teacher;
```

	discipline	res
►	Программирование	4
	Математика	2
	Физика	2
	ИКТ	2

Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу; кэширования результатов выборки из таблицы при работе представлений не производится. При этом, механизм кэширования запросов (query cache) работает на уровне запросов пользователя безотносительно к тому, обращается ли пользователь к таблицам или представлениям. Представления могут основываться как на таблицах, так и на других представлениях, т.е. могут быть вложенными (до 32 уровней вложенности).

Операции с представлениями

- **DROP** : представление/виртуальную таблицу можно удалить с помощью команды **DROP VIEW**
- Объединение: мы также можем создать представление, объединив несколько таблиц. Это соединение будет извлекать совпадающие записи из обеих таблиц. (
- Создание рассмотрено выше (**CREATE VIEW**)
- Изменение существующего представления - **ALTER VIEW**

```

-- Исключим математиков
ALTER VIEW count_teacher AS
SELECT
    discipline, count(first_name) AS res
FROM staff
WHERE discipline!="Математика"
GROUP BY discipline
ORDER BY count(first_name) DESC;
-- Показ
SELECT * FROM count_teacher;

```


	discipline	res
►	Программирование	4
	Физика	2
	ИКТ	2

Преимущества использования представлений:

1. Дает возможность гибкой настройки прав доступа к данным за счет того, что права даются не на таблицу, а на представление. Это очень удобно в случае если пользователю нужно дать права на отдельные строки таблицы или возможность получения не самих данных, а результата каких-то действий над ними.
2. Позволяет разделить логику хранения данных и программного обеспечения. Можно менять структуру данных, не затрагивая программный код, нужно лишь создать представления, аналогичные таблицам, к которым раньше обращались приложения. Это очень удобно когда нет возможности изменить программный код или к одной базе данных обращаются несколько приложений с различными требованиями к структуре данных.
3. Удобство в использовании за счет автоматического выполнения таких действий как доступ к определенной части строк и/или столбцов, получение данных из нескольких таблиц и их преобразование с помощью различных функций.

Полезные ссылки и рекомендации:

- <https://habr.com/ru/company/oleg-bunin/blog/348172/> - DDL, DCL на **MS SQL Server (1 часть)**.
- <https://habr.com/ru/post/255523/> - DDL, DCL на **MS SQL Server (2 часть)**.
- Руководство по стилю написания SQL: <https://www.sqlstyle.guide/ru/>
- <https://tproger.ru/translations/sql-window-functions/> - оконные функции
- Старайтесь не использовать русские буквы при работе с СУБД

Книги:

- “Изучаем SQL”, книга Бейли Л.
 - Алан Бьюли "Изучаем SQL" (2007)
 - Энтони Молинаро "SQL. Сборник рецептов" (2009)
1. Виктор Гольцман “MySQL 5.0. Библиотека программиста”
 2. “Изучаем SQL”, книга Бейли Л.
 3. <https://www.webmasterwiki.ru/MySQL>
 4. Поль Дюбуа “MySQL. Сборник рецептов”