### Базы данных и SQL. Обучение в записи

Урок 10. Семинар: SQL - оконные функции

Ссылка на penoзиторий, с выполненным Д/3: https://github.com/olgashenkel/Databases-and-SQL

### **ЧАСТЬ І.**

		mysql> SELECT * FROM Cars;						
1.	Создайте представление, в которое попадут	+-						-+
автомо	автомобили стоимостью до 25 000 долларов				Name		Cost	-+
		1	1	I	Audi	1	52642	1
2.	Изменить в существующем представлении	1	2	1	Mercedes	1	57127	1
порог д	ля стоимости: пусть цена будет до 30 000	- 1	3		Skoda	1	9000	1
лоллар:	ларов (используя оператор ALTER VIEW)		4	1	Volvo	1	29000	1
Harris Ch.	()	- 1	5		Bentley	-	350000	1
_		- 1	6		Citroen		21000	1
3.	Создайте представление, в котором будут	1	7		Hummer	1	41400	1
олько автомобили марки "Шкода" и "Ауди"		1	8	I	Volkswagen	1	21600	1

#### 1. Ход выполнения задания 1.1:

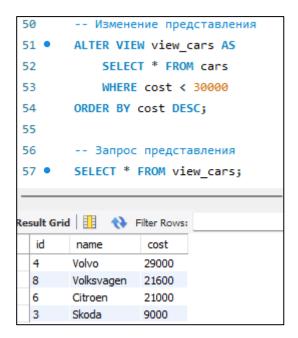
Создайте представление, в которое попадут автомобили стоимостью до 25 000 долларов.

```
USE seminar_5;
2
3
 4
       1. Ззадание 1:
       Создайте представление, в которое попадут
 6
       автомобили стоимостью до 25 000 долларов.
 7
8
9
       -- Создание и наполнение данными таблицы cars
10 •
      DROP TABLE IF EXISTS cars;
      CREATE TABLE cars
12
           id INT PRIMARY KEY AUTO_INCREMENT,
13
14
           name VARCHAR(20),
           cost INT
15
16
17
18 •
    INSERT INTO cars(name, cost)
19
       VALUES
      ('Audi', 52642),
20
21
      ('Mercedes', 57127),
22
      ('Skoda', 9000),
23
    ('Volvo', 29000),
      ('Bentley', 350000),
      ('Citroen', 21000),
26
       ('Hummer', 41400),
       ('Volksvagen', 21600);
```

```
31
            -- Создание представления
    32 •
           CREATE OR REPLACE VIEW view_cars AS
    33
                SELECT *
    34
                FROM cars
    35
                WHERE cost < 25000
                ORDER BY cost DESC;
    36
    37
            -- Запрос созданного представления
           SELECT * FROM view_cars;
    esult Grid 🔢 🔷 Filter Rows:
                                              Export
                       cost
     8
           Volksvagen
                      21600
     6
           Citroen
                      21000
     3
           Skoda
2)
```

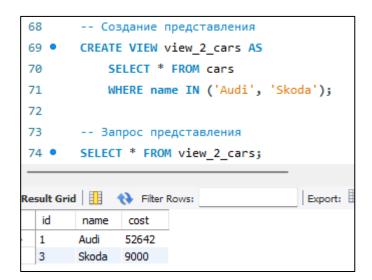
#### 2. Ход выполнения задания 1.2:

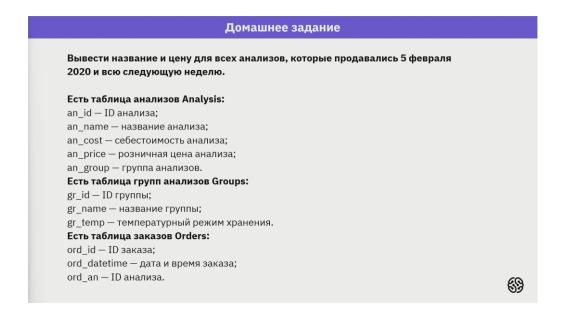
Изменить в существующем представлении порог стоимости: пусть цена будет до 30000 долларов (используя оператор ALTER VIEW).



### 3. Ход выполнения задания 1.3:

Создайте представление, в котором будут только автомобили марки Skoda и Audi.





#### 1. Ход выполнения задания 2:

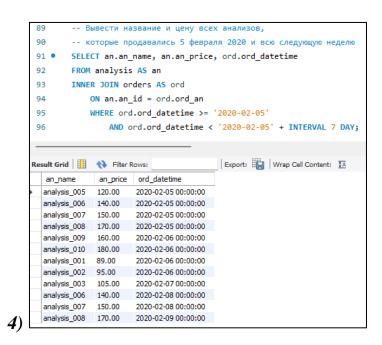
Вывести название и цену всех анализов, которые продавались 5 февраля 2020 и всю следующую неделю.

```
USE semnar_5;
       -- Создание и заполнение таблицы analysis
      DROP TABLE IF EXISTS analysis;
5 •
      CREATE TABLE analysis
    ⊖ (
6
           an_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
           an name VARCHAR(45) NOT NULL,
8
9
           an_cost DECIMAL(10,2),
10
           an price DECIMAL(10,2),
           an_group VARCHAR(10)
11
12
       );
13
      INSERT INTO analysis(an_name, an_cost, an_price, an_group)
14 •
15
       ('analysis_001', 50, 89, 'A'),
16
17
      ('analysis_002', 60, 95, 'B'),
      ('analysis_003', 70, 105, 'C'),
18
19
       ('analysis_004', 80, 110, 'D'),
20
      ('analysis_005', 90, 120, 'A'),
21
      ('analysis_006', 100, 140, 'B'),
      ('analysis_007', 110, 150, 'C'),
22
23
      ('analysis_008', 120, 170, 'D'),
       ('analysis_009', 130, 160, 'A'),
24
25
       ('analysis_010', 140, 180, 'B');
26
27 •
      SELECT * FROM analysis;
 sult Grid 🔢 🙌 Filter Rows:
                                       Edit: 🚣 🖶 Export/Import:
 an_id an_name
                  an cost
                          an_price an_group
       analysis_001
                   50.00
                           89.00
       analysis_002 60.00 95.00
                                  В
       analysis 003
                   70.00
                           105.00
       analysis 004 80.00
                           110.00
                                  D
       analysis_005
                   90.00
                           120.00
       analysis_006 100.00
                           140.00
       analysis_007
                   110.00
                           150.00
       analysis_008 120.00
                           170.00 D
       analysis 009
                  130.00
                           160.00
       analysis_010 140.00
                           180.00
```

```
-- Создание и заполнение таблицы grups
31 •
      DROP TABLE IF EXISTS grups;
32 •
      CREATE TABLE grups
33
           gr_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
34
35
           gr_name VARCHAR(45) NOT NULL,
36
           gr_temp DECIMAL(5,2)
37
38
39 •
      INSERT INTO grups(gr_name, gr_temp)
       VALUE
40
       ('group_001', 10),
41
42
      ('group 002', 11),
43
      ('group_003', 12),
44
      ('group_004', 13),
45
      ('group_005', 14),
46
      ('group_006', 15),
47
      ('group_007', 16),
      ('group_008', 17),
48
49
      ('group_009', 18),
       ('group_010', 19);
50
51
52 •
      SELECT * FROM grups;
esult Grid 🔢 🙌 Filter Rows:
                                       Edit: 🚄 🖶 🖶 Exp
 gr_id gr_name gr_temp
                 10.00
       group 001
     group 002 11,00
 3
       group_003
                 12.00
     group_004 13.00
       group_005
                 14.00
 6
      group_006
                 15.00
       group_007
 8
      group_008
                 17.00
       group_009
                 18.00
      group_010 19.00
```

2)

```
Создание и заполнение таблицы orders
       56 •
             CREATE TABLE orders
       57
       58
                  ord_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
       59
                  ord_datetime DATETIME,
       60
                  ord an INT NOT NULL,
       61
                  FOREIGN KEY (ord_an) REFERENCES analysis (an_id)
       62
       63
       64 •
            INSERT INTO orders(ord_datetime, ord_an)
       65
              VALUE
       66
             ('2020-02-04 00:00:00', 1),
       67
              ('2020-02-04 00:01:00', 2),
       68
             ('2020-02-04 00:02:00', 3),
       69
             ('2020-02-04 12:00:00', 4),
       70
             ('2020-02-05 00:00:00', 5),
       71
              ('2020-02-05 00:00:00', 6),
       72
             ('2020-02-05 00:00:00', 7),
       73
              ('2020-02-05 00:00:00', 8),
       74
             ('2020-02-06 00:00:00', 9),
       75
              ('2020-02-06 00:00:00', 10),
       76
             ('2020-02-06 00:00:00', 1),
       77
              ('2020-02-06 00:00:00', 2),
       78
             ('2020-02-07 00:00:00', 3),
       79
              ('2020-02-14 00:00:00', 4),
       80
             ('2020-02-24 00:00:00', 5),
       81
             ('2020-02-08 00:00:00', 6),
       82
             ('2020-02-08 00:00:00', 7),
       83
             ('2020-02-09 00:00:00', 8),
       84
             ('2020-02-14 00:00:00', 9),
      85
              ('2020-02-15 00:00:00', 10);
       86
              SELECT * FROM orders:
       87 •
       esult Grid 📗 💎 Filter Rows:
                                            Edit: 🚄 🖶 🖶 Export
        ord_id ord_datetime
               2020-02-04 00:00:00
               2020-02-04 00:01:00 2
               2020-02-04 00:02:00
               2020-02-04 12:00:00 4
               2020-02-05 00:00:00 5
3)
```



### Домашнее задание

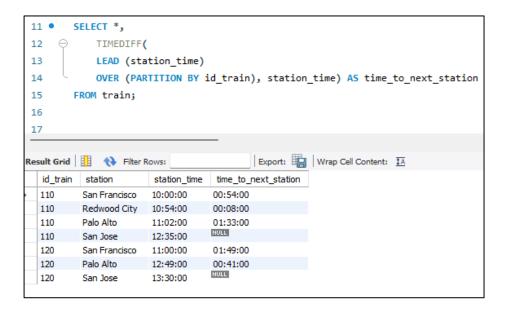
Добавьте новый столбец под названием «время до следующей станции». Чтобы получить это значение, мы вычитаем время станций для пар смежных станций. Мы можем вычислить это значение без использования оконной функции SQL, но это может быть очень сложно. Проще это сделать с помощью оконной функции LEAD . Эта функция сравнивает значения из одной строки со следующей строкой, чтобы получить результат. В этом случае функция сравнивает значения в столбце «время» для станции со станцией сразу после нее.

train_id integer		station_time time without time zone	time_to_next_station interval			
110	San Francisco	10:00:00	00:54:00			
110	Redwood City	10:54:00	00:08:00			
110	Palo Alto	11:02:00	01:33:00			
110	San Jose	12:35:00				
120	San Francisco	11:00:00	01:49:00			
120	Palo Alto	12:49:00	00:41:00			
120	San Jose	13:30:00				



### 2. Ход выполнения задания 3:

Добавьте новый столбец под названием «время до следующей станции».



#### ЧАСТЬ II.

# 1. Ход выполнения задания 0: Запрос на создание начальной таблицы, выполнять задания на основе этих данных

Вы можете воспользоваться заготовкой fiddle <a href="https://dbfiddle.uk/oQi939Ap">https://dbfiddle.uk/oQi939Ap</a> Или использовать код ниже для создания таблиц

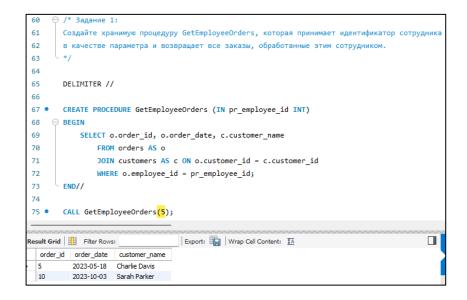
```
USE seminar_5;
-- Запрос на создание начальной таблицы, выполнять задания на основе этих данных
-- Создание таблицы Employees
DROP TABLE IF EXISTS Employees;
CREATE TABLE Employees (
employee_id INT PRIMARY KEY,
employee_name VARCHAR(255)
);
-- Создание таблицы Orders
DROP TABLE IF EXISTS Orders;
CREATE TABLE Orders (
order_id INT PRIMARY KEY,
employee_id INT,
customer_id INT,
order_date DATE,
total_amount DECIMAL(10, 2),
FOREIGN KEY (employee_id) REFERENCES Employees(employee_id)
);
-- Создание таблицы Customers
DROP TABLE IF EXISTS Customers;
CREATE TABLE Customers (
customer id INT PRIMARY KEY,
customer_name VARCHAR(255)
-- Наполнение таблиц данными
INSERT INTO Employees (employee_id, employee_name)
VALUES
(1, 'Alice Johnson'), (2, 'Bob Brown'), (3, 'Charlie Davis'),
(4, 'David Wilson'), (5, 'Emily Clark');
INSERT INTO Customers (customer_id, customer_name)
VALUES
(1, 'John Doe'), (2, 'Jane Smith'), (3, 'Alice Johnson'),
(4, 'Bob Brown'), (5, 'Charlie Davis'), (6, 'Emily Clark'),
(7, 'David Wilson'), (8, 'Laura Adams'), (9, 'Michael Thompson'),
(10, 'Sarah Parker');
INSERT INTO Orders (order_id, employee_id, customer_id, order_date, total_amount)
VALUES
(1, 1, 1, '2023-01-15', 550.00),
(2, 2, 2, '2023-02-20', 600.00),
(3, 3, 3, '2023-03-10', 300.00),
(4, 4, 4, '2023-04-25', 750.00),
(5, 5, 5, '2023-05-18', 450.00),
(6, 1, 6, '2023-06-12', 500.00),
(7, 2, 7, '2023-07-21', 700.00),
(8, 3, 8, '2023-08-30', 620.00),
(9, 4, 9, '2023-09-14', 480.00),
(10, 5, 10, '2023-10-03', 510.00);
```

#### 2. Ход выполнения задания 1: Получение заказов по сотруднику

Создайте хранимую процедуру GetEmployeeOrders, которая принимает идентификатор сотрудника в качестве параметра и возвращает все заказы, обработанные этим сотрудником. В результате запроса будут столбцы:

- order id: идентификатор заказа
- order date: дата заказа
- customer name: имя клиента

Подсказка: Используйте оператор CREATE PROCEDURE для создания хранимой процедуры. Для объединения таблиц используйте оператор JOIN. Параметр передается в процедуру через IN.

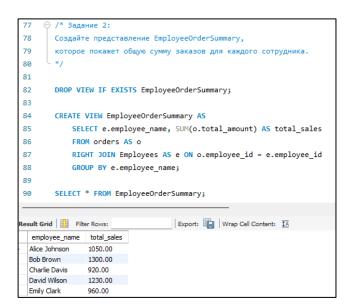


3. Ход выполнения задания 2: Создание представления для суммы заказов по сотрудникам Создайте представление EmployeeOrderSummary, которое покажет общую сумму заказов для каждого сотрудника.

В результате запроса будут столбцы:

- employee name: имя сотрудника
- total sales: общая сумма заказов

Подсказка: Используйте оператор CREATE VIEW для создания представления. Для агрегирования данных используйте функцию SUM и группировку GROUP BY.

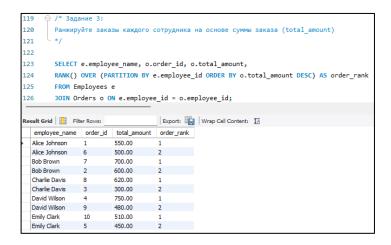


# **4. Ход выполнения задания 3: Использование оконных функций для ранжирования заказов** Ранжируйте заказы каждого сотрудника на основе суммы заказа (total amount).

В результате запроса будут столбцы:

- employee name: имя сотрудника
- order id: идентификатор заказа
- total amount: сумма заказа
- order rank: ранг заказа по сумме для каждого сотрудника

Подсказка: Используйте функцию RANK() с PARTITION BY для разделения данных по сотрудникам и ORDER BY для сортировки по сумме заказа. Объедините таблицы с помощью JOIN.



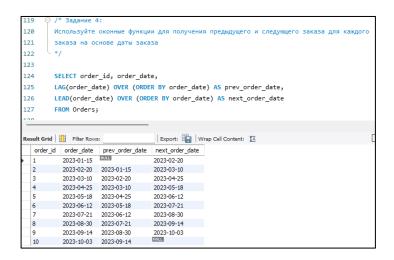
## 5. Ход выполнения задания 4: Получение предыдущего и следующего заказа для каждого заказа

Используйте оконные функции для получения предыдущего и следующего заказа для каждого заказа на основе даты заказа.

В результате запроса будут столбцы:

- order id: идентификатор заказа
- order date: дата заказа
- prev order date: дата предыдущего заказа
- next order date: дата следующего заказа

Подсказка: Используйте оконные функции LAG и LEAD для получения предыдущего и следующего значений. Примените ORDER BY для упорядочивания заказов по дате.



## 6. Ход выполнения задания 5: Аналитические функции для среднего, минимального и максимального значения заказов

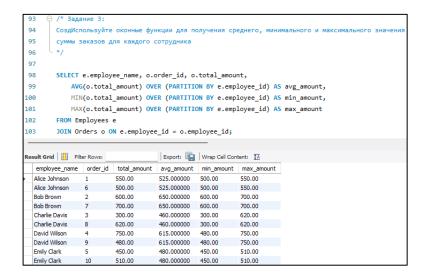
Используйте оконные функции для получения среднего, минимального и максимального значения суммы заказов для каждого сотрудника.

В результате запроса будут столбцы:

- employee name: имя сотрудника

- order id: идентификатор заказа
- total amount: сумма заказа
- avg amount: средняя сумма заказа для каждого сотрудника
- тіп amount: минимальная сумма заказа для каждого сотрудника
- тах amount: максимальная сумма заказа для каждого сотрудника

Подсказка: Используйте оконные функции AVG, MIN и MAX с PARTITION BY для разделения данных по сотрудникам. Объедините таблицы с помощью JOIN.



#### 7. Ход выполнения задания 6: Комплексный анализ заказов

Создайте представление OrderAnalysis, которое будет содержать информацию о каждом заказе, включая имя сотрудника, имя клиента, ранг заказа по сумме для каждого сотрудника, среднюю, минимальную и максимальную сумму заказов для каждого сотрудника, а также даты предыдущего и следующего заказов.

В результате запроса будут столбцы:

- employee\_name: имя сотрудника
- customer name: имя клиента
- order id: идентификатор заказа
- total amount: сумма заказа
- order rank: ранг заказа по сумме для каждого сотрудника
- avg amount: средняя сумма заказа для каждого сотрудника
- тіп amount: минимальная сумма заказа для каждого сотрудника
- тах amount: максимальная сумма заказа для каждого сотрудника
- prev order date: дата предыдущего заказа
- next\_order\_date: дата следующего заказа

Подсказка: Создайте представление с помощью CREATE VIEW. Используйте функции RANK, AVG, MIN, MAX, LAG и LEAD для анализа данных. Объедините данные из таблиц Employees, Orders и Customers с помощью JOIN.

```
Создайте представление OrderAnalysis, которое будет содержать информацию о каждом заказе,
131
        включая имя сотрудника, имя клиента, ранг заказа по сумме для каждого сотрудника, сред
132
        минимальную и максимальную сумму заказов для каждого сотрудника,
133
        а также даты предыдущего и следующего заказов
134
135
       CREATE VIEW OrderAnalysis AS
        SELECT e.employee_name, c.customer_name, o.order_id, o.total_amount,
      RANK() OVER (PARTITION BY e.employee_id ORDER BY o.total_amount DESC) AS order_rank,
138
139
           AVG(o.total amount) OVER (PARTITION BY e.employee id) AS avg amount,
140
          MIN(o.total_amount) OVER (PARTITION BY e.employee_id) AS min_amount,
141
          MAX(o.total_amount) OVER (PARTITION BY e.employee_id) AS max_amount,
142
          LAG(o.order_date) OVER (ORDER BY o.order_date) AS prev_order_date,
         LEAD(o.order_date) OVER (ORDER BY o.order_date) AS next_order_date
143
144
       FROM Employees e
145
       JOIN Orders o ON e.employee id = o.employee id
146
        JOIN Customers c ON o.customer id = c.customer id;
147
148
        SELECT * FROM OrderAnalysis;
```

employee_name	customer_name	order_id	total_amount	order_rank	avg_amount	min_amount	max_amount	prev_order_date	next_order_date
Alice Johnson	John Doe	1	550.00	1	525.000000	500.00	550.00	NULL	2023-02-20
Bob Brown	Jane Smith	2	600.00	2	650.000000	600.00	700.00	2023-01-15	2023-03-10
Charlie Davis	Alice Johnson	3	300.00	2	460.000000	300.00	620.00	2023-02-20	2023-04-25
David Wilson	Bob Brown	4	750.00	1	615.000000	480.00	750.00	2023-03-10	2023-05-18
Emily Clark	Charlie Davis	5	450.00	2	480.000000	450.00	510.00	2023-04-25	2023-06-12
Alice Johnson	Emily Clark	6	500.00	2	525.000000	500.00	550.00	2023-05-18	2023-07-21
Bob Brown	David Wilson	7	700.00	1	650.000000	600.00	700.00	2023-06-12	2023-08-30
Charlie Davis	Laura Adams	8	620.00	1	460.000000	300.00	620.00	2023-07-21	2023-09-14
David Wilson	Michael Thompson	9	480.00	2	615.000000	480.00	750.00	2023-08-30	2023-10-03
Emily Clark	Sarah Parker	10	510.00	1	480.000000	450.00	510.00	2023-09-14	NULL