Базы данных и SQL. Обучение в записи

Урок 10. Семинар: SQL – оконные функции

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

ЧАСТЬ I.

Домашнее задание

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

2. Изменить в существующем представлении порог для стоимости: пусть цена будет до 30 000 долларов (используя оператор ALTER VIEW)

3. Создайте представление, в котором будут только автомобили марки “Шкода” и “Ауди”

```
mysql> SELECT * FROM Cars;
+----+------------+--------+
| Id | Name       | Cost   |
+----+------------+--------+
|  1 | Audi       |  52642 |
|  2 | Mercedes   |  57127 |
|  3 | Skoda      |   9000 |
|  4 | Volvo      |  29000 |
|  5 | Bentley    | 350000 |
|  6 | Citroen    |  21000 |
|  7 | Hummer     |  41400 |
|  8 | Volkswagen |  21600 |
+----+------------+--------+
```

1. *Ход выполнения задания 1.1:*

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

1)

```sql
USE seminar_5;

/*
1.  Ззадание 1:
Создайте представление, в которое попадут
автомобили стоимостью до 25 000 долларов.
*/

-- Создание и наполнение данными таблицы cars
DROP TABLE IF EXISTS cars;
CREATE TABLE cars
(
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(20),
    cost INT
);

INSERT INTO cars(name, cost)
VALUES
('Audi', 52642),
('Mercedes', 57127),
('Skoda', 9000),
('Volvo', 29000),
('Bentley', 350000),
('Citroen', 21000),
('Hummer', 41400),
('Volksvagen', 21600);
```

2)

```sql
-- Создание представления
CREATE OR REPLACE VIEW view_cars AS
    SELECT *
    FROM cars
    WHERE cost < 25000
    ORDER BY cost DESC;

-- Запрос созданного представления
SELECT * FROM view_cars;
```

Result Grid | Filter Rows: | Export

id	name	cost
8	Volksvagen	21600
6	Citroen	21000
3	Skoda	9000

2. *Ход выполнения задания 1.2:*

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

```
50      -- Изменение представления
51      ALTER VIEW view_cars AS
52          SELECT * FROM cars
53          WHERE cost < 30000
54      ORDER BY cost DESC;
55
56      -- Запрос представления
57      SELECT * FROM view_cars;
```

Result Grid | Filter Rows:

id	name	cost
4	Volvo	29000
8	Volksvagen	21600
6	Citroen	21000
3	Skoda	9000

3. Ход выполнения задания 1.3:

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

```
68      -- Создание представления
69      CREATE VIEW view_2_cars AS
70          SELECT * FROM cars
71          WHERE name IN ('Audi', 'Skoda');
72
73      -- Запрос представления
74      SELECT * FROM view_2_cars;
```

Result Grid | Filter Rows: | Export:

id	name	cost
1	Audi	52642
3	Skoda	9000

Домашнее задание

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

Есть таблица анализов Analysis:
an_id — ID анализа;
an_name — название анализа;
an_cost — себестоимость анализа;
an_price — розничная цена анализа;
an_group — группа анализов.
Есть таблица групп анализов Groups:
gr_id — ID группы;
gr_name — название группы;
gr_temp — температурный режим хранения.
Есть таблица заказов Orders:
ord_id — ID заказа;
ord_datetime — дата и время заказа;
ord_an — ID анализа.

1. ***Ход выполнения задания 2:***

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

1)

```sql
USE semnar_5;

-- Создание и заполнение таблицы analysis
DROP TABLE IF EXISTS analysis;
CREATE TABLE analysis
(
    an_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    an_name VARCHAR(45) NOT NULL,
    an_cost DECIMAL(10,2),
    an_price DECIMAL(10,2),
    an_group VARCHAR(10)
);

INSERT INTO analysis(an_name, an_cost, an_price, an_group)
VALUE
('analysis_001', 50, 89, 'A'),
('analysis_002', 60, 95, 'B'),
('analysis_003', 70, 105, 'C'),
('analysis_004', 80, 110, 'D'),
('analysis_005', 90, 120, 'A'),
('analysis_006', 100, 140, 'B'),
('analysis_007', 110, 150, 'C'),
('analysis_008', 120, 170, 'D'),
('analysis_009', 130, 160, 'A'),
('analysis_010', 140, 180, 'B');

SELECT * FROM analysis;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

an_id	an_name	an_cost	an_price	an_group
1	analysis_001	50.00	89.00	A
2	analysis_002	60.00	95.00	B
3	analysis_003	70.00	105.00	C
4	analysis_004	80.00	110.00	D
5	analysis_005	90.00	120.00	A
6	analysis_006	100.00	140.00	B
7	analysis_007	110.00	150.00	C
8	analysis_008	120.00	170.00	D
9	analysis_009	130.00	160.00	A
10	analysis_010	140.00	180.00	B

2)

```sql
-- Создание и заполнение таблицы grups
DROP TABLE IF EXISTS grups;
CREATE TABLE grups
(
    gr_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    gr_name VARCHAR(45) NOT NULL,
    gr_temp DECIMAL(5,2)
);

INSERT INTO grups(gr_name, gr_temp)
VALUE
('group_001', 10),
('group_002', 11),
('group_003', 12),
('group_004', 13),
('group_005', 14),
('group_006', 15),
('group_007', 16),
('group_008', 17),
('group_009', 18),
('group_010', 19);

SELECT * FROM grups;
```

Result Grid | Filter Rows: | Edit: | Exp

gr_id	gr_name	gr_temp
1	group_001	10.00
2	group_002	11.00
3	group_003	12.00
4	group_004	13.00
5	group_005	14.00
6	group_006	15.00
7	group_007	16.00
8	group_008	17.00
9	group_009	18.00
10	group_010	19.00

3)

```sql
-- Создание и заполнение таблицы orders
CREATE TABLE orders
(
    ord_id INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    ord_datetime DATETIME,
    ord_an INT NOT NULL,
    FOREIGN KEY (ord_an) REFERENCES analysis (an_id)
);

INSERT INTO orders(ord_datetime, ord_an)
VALUE
('2020-02-04 00:00:00', 1),
('2020-02-04 00:01:00', 2),
('2020-02-04 00:02:00', 3),
('2020-02-04 12:00:00', 4),
('2020-02-05 00:00:00', 5),
('2020-02-05 00:00:00', 6),
('2020-02-05 00:00:00', 7),
('2020-02-05 00:00:00', 8),
('2020-02-06 00:00:00', 9),
('2020-02-06 00:00:00', 10),
('2020-02-06 00:00:00', 1),
('2020-02-06 00:00:00', 2),
('2020-02-07 00:00:00', 3),
('2020-02-14 00:00:00', 4),
('2020-02-24 00:00:00', 5),
('2020-02-08 00:00:00', 6),
('2020-02-08 00:00:00', 7),
('2020-02-09 00:00:00', 8),
('2020-02-14 00:00:00', 9),
('2020-02-15 00:00:00', 10);

SELECT * FROM orders;
```

Result Grid | Filter Rows: | Edit: | Export/

ord_id	ord_datetime	ord_an
1	2020-02-04 00:00:00	1
2	2020-02-04 00:01:00	2
3	2020-02-04 00:02:00	3
4	2020-02-04 12:00:00	4
5	2020-02-05 00:00:00	5

4)

```sql
-- Вывести название и цену всех анализов,
-- которые продавались 5 февраля 2020 и всю следующую неделю
SELECT an.an_name, an.an_price, ord.ord_datetime
FROM analysis AS an
INNER JOIN orders AS ord
    ON an.an_id = ord.ord_an
    WHERE ord.ord_datetime >= '2020-02-05'
        AND ord.ord_datetime < '2020-02-05' + INTERVAL 7 DAY;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

an_name	an_price	ord_datetime
analysis_005	120.00	2020-02-05 00:00:00
analysis_006	140.00	2020-02-05 00:00:00
analysis_007	150.00	2020-02-05 00:00:00
analysis_008	170.00	2020-02-05 00:00:00
analysis_009	160.00	2020-02-06 00:00:00
analysis_010	180.00	2020-02-06 00:00:00
analysis_001	89.00	2020-02-06 00:00:00
analysis_002	95.00	2020-02-06 00:00:00
analysis_003	105.00	2020-02-07 00:00:00
analysis_006	140.00	2020-02-08 00:00:00
analysis_007	150.00	2020-02-08 00:00:00
analysis_008	170.00	2020-02-09 00:00:00

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 character varying(20)	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».

```
11 •  SELECT *,
12        TIMEDIFF(
13        LEAD (station_time)
14        OVER (PARTITION BY id_train), station_time) AS time_to_next_station
15    FROM train;
16
17
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

id_train	station	station_time	time_to_next_station
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	NULL
120	San Francisco	11:00:00	01:49:00
120	Palo Alto	12:49:00	00:41:00
120	San Jose	13:30:00	NULL

ЧАСТЬ II.

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

Вы можете воспользоваться заготовкой fiddle https://dbfiddle.uk/oQi939Ap
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.

```
/* Задание 1:
Создайте хранимую процедуру GetEmployeeOrders, которая принимает идентификатор сотрудника
в качестве параметра и возвращает все заказы, обработанные этим сотрудником.
*/

DELIMITER //

CREATE PROCEDURE GetEmployeeOrders (IN pr_employee_id INT)
BEGIN
    SELECT o.order_id, o.order_date, c.customer_name
        FROM orders AS o
        JOIN customers AS c ON o.customer_id = c.customer_id
        WHERE o.employee_id = pr_employee_id;
END//

CALL GetEmployeeOrders(5);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	order_date	customer_name
5	2023-05-18	Charlie Davis
10	2023-10-03	Sarah Parker

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.

```
/* Задание 2:
Создайте представление EmployeeOrderSummary,
которое покажет общую сумму заказов для каждого сотрудника.
*/

DROP VIEW IF EXISTS EmployeeOrderSummary;

CREATE VIEW EmployeeOrderSummary AS
    SELECT e.employee_name, SUM(o.total_amount) AS total_sales
    FROM orders AS o
    RIGHT JOIN Employees AS e ON o.employee_id = e.employee_id
    GROUP BY e.employee_name;

SELECT * FROM EmployeeOrderSummary;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_name	total_sales
Alice Johnson	1050.00
Bob Brown	1300.00
Charlie Davis	920.00
David Wilson	1230.00
Emily Clark	960.00

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.

```
119  /* Задание 3:
120     Ранжируйте заказы каждого сотрудника на основе суммы заказа (total_amount)
121     */
122
123     SELECT e.employee_name, o.order_id, o.total_amount,
124     RANK() OVER (PARTITION BY e.employee_id ORDER BY o.total_amount DESC) AS order_rank
125     FROM Employees e
126     JOIN Orders o ON e.employee_id = o.employee_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_name	order_id	total_amount	order_rank
Alice Johnson	1	550.00	1
Alice Johnson	6	500.00	2
Bob Brown	7	700.00	1
Bob Brown	2	600.00	2
Charlie Davis	8	620.00	1
Charlie Davis	3	300.00	2
David Wilson	4	750.00	1
David Wilson	9	480.00	2
Emily Clark	10	510.00	1
Emily Clark	5	450.00	2

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.

```
119  /* Задание 4:
120     Используйте оконные функции для получения предыдущего и следующего заказа для каждого
121     заказа на основе даты заказа
122     */
123
124     SELECT order_id, order_date,
125     LAG(order_date) OVER (ORDER BY order_date) AS prev_order_date,
126     LEAD(order_date) OVER (ORDER BY order_date) AS next_order_date
127     FROM Orders;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

order_id	order_date	prev_order_date	next_order_date
1	2023-01-15	NULL	2023-02-20
2	2023-02-20	2023-01-15	2023-03-10
3	2023-03-10	2023-02-20	2023-04-25
4	2023-04-25	2023-03-10	2023-05-18
5	2023-05-18	2023-04-25	2023-06-12
6	2023-06-12	2023-05-18	2023-07-21
7	2023-07-21	2023-06-12	2023-08-30
8	2023-08-30	2023-07-21	2023-09-14
9	2023-09-14	2023-08-30	2023-10-03
10	2023-10-03	2023-09-14	NULL

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*
- *min_amount: минимальная сумма заказа для каждого сотрудника*
- *max_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.

```sql
/* Задание 3:
СоздИспользуйте оконные функции для получения среднего, минимального и максимального значения
суммы заказов для каждого сотрудника
*/

SELECT e.employee_name, o.order_id, o.total_amount,
    AVG(o.total_amount) OVER (PARTITION BY e.employee_id) AS avg_amount,
    MIN(o.total_amount) OVER (PARTITION BY e.employee_id) AS min_amount,
    MAX(o.total_amount) OVER (PARTITION BY e.employee_id) AS max_amount
FROM Employees e
JOIN Orders o ON e.employee_id = o.employee_id;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

employee_name	order_id	total_amount	avg_amount	min_amount	max_amount
Alice Johnson	1	550.00	525.000000	500.00	550.00
Alice Johnson	6	500.00	525.000000	500.00	550.00
Bob Brown	2	600.00	650.000000	600.00	700.00
Bob Brown	7	700.00	650.000000	600.00	700.00
Charlie Davis	3	300.00	460.000000	300.00	620.00
Charlie Davis	8	620.00	460.000000	300.00	620.00
David Wilson	4	750.00	615.000000	480.00	750.00
David Wilson	9	480.00	615.000000	480.00	750.00
Emily Clark	5	450.00	480.000000	450.00	510.00
Emily Clark	10	510.00	480.000000	450.00	510.00

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*
- *min_amount: минимальная сумма заказа для каждого сотрудника*
- *max_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.

```sql
/* Задание 6:
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
*/

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,
    AVG(o.total_amount) OVER (PARTITION BY e.employee_id) AS avg_amount,
    MIN(o.total_amount) OVER (PARTITION BY e.employee_id) AS min_amount,
    MAX(o.total_amount) OVER (PARTITION BY e.employee_id) AS max_amount,
    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
FROM Employees e
JOIN Orders o ON e.employee_id = o.employee_id
JOIN Customers c ON o.customer_id = c.customer_id;

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