

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ  
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**Факультет безопасности информационных технологий**

**Направление подготовки: 10.03.01 Информационная безопасность**  
**Образовательная программа: "Информационная безопасность / Information security"**

**Дисциплина:**  
**«Информационная безопасность баз данных»**

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2**  
**«Манипулирование данными в БД на языке SQL»**

**Выполнил студент(ы):**  
группа/поток 1.3  
Бардышев Артём Антонович/\_\_\_\_\_  
*Подпись*

**Проверил:**  
Карманова Наталья Андреевна/\_\_\_\_\_  
*Подпись*

*Отметка о выполнении (один из вариантов:  
отлично, хорошо, удовлетворительно, зачленено)*

*Дата*

Санкт-Петербург  
2025г.

1. **Цель работы:** Получение навыков манипулирования данными в БД при помощи операторов SQL.

## 2. Теоретическая информация:

Особенности хранения данных в БД накладывает ограничения на манипулирование этими данными. Для эффективного решения задач, связанных с получением нужной информации из БД существует специальных язык структурированных запросов в БД – SQL. Все запросы оформляются в виде операторов SQL, обладающих собственным синтаксисом. В зависимости от реализации СУБД реализация SQL запросов может несколько отличаться, но базовые примитивы остаются неизменными. Наряду со стандартными операциями работы с данными – вставка, удаление, извлечение, обновление, SQL обладает расширенными возможностями по работе с данными, позволяющими получать данные в соответствии с разнообразными условиями. Владение навыками работы с БД при помощи SQL позволяет эффективно решать задачи по систематизации, поиску, хранению информации в СУБД.

В качестве СУБД, используемой в лабораторной работе, предполагается PostgreSQL. Основные операторы представлены ниже.

## Оператор SELECT

В общем виде оператор SELECT конструируется в виде:

[WITH запросы\_with] SELECT список\_выборки FROM табличное\_выражение [WHERE условие\_ограничения][определение\_сортировки]

Где WITH подразумевает возможность использования результата другого оператора SQL, например, SELECT, с которым уже производится работа, как с таблицей в текущем запросе. Список выборки - перечень атрибутов, которые должны быть отображены в полученной таблице, указываются атрибуты, присутствующие в табличном выражении, порядок указания значения не имеет. В качестве табличного выражения выступает таблица, которая может вычисляться на этапе выполнения оператора. Обычная таблица представляет собой физическую таблицу в БД, но при необходимости можно преобразовывать и комбинировать на основе физических таблиц и получать виртуальную таблицу. Условие ограничения - это любое выражение, выдающие результат в виде boolean, которое применяется к каждой строке полученной таблицы из табличного выражения. Если результат вычисление true, то такая строка остается в противном случае отбрасывается. Определение сортировки позволяет упорядочить строки по значению одного из атрибутов.

Пример:

```
SELECT sensor_name FROM sensor_translation WHERE mach_id=65540 AND lang_id=1  
AND sensor_no=0 AND group_name='E1'
```

## Оператор INSERT

[WITH запросы\_with] INSERT INTO таблица (список\_атрибутов) VALUES  
(список\_значений)

Добавление новых записей в таблицу производится оператором INSERT. Необходимо в явном виде указать таблицу, в которую добавляются строки, а также список атрибутов добавляемой записи и ее значения.

Пример:

```
INSERT INTO gps (gps_id,utc_datetime,mach_id,long_value,lat_value,speed) VALUES  
(132,'2016-01-04 14:00',60031,4545.9090,7878.9090,0);
```

## Оператор UPDATE

[WITH запросы\_with] UPDATE таблица SET атрибут=новое значение [WHERE условие\_обновления]

Для обновления записи в таблице используется оператор UPDATE. В качестве параметров необходимо указывать атрибуты, которые необходимо обновить и новые значения этих атрибутов после ключевого слова SET. Для обновления только определенных атрибутов существует возможность применять условие обновления после ключевого слова WHERE с тем же синтаксисом что в операторе SELECT.

Пример:

```
UPDATE machines SET mach_type=7 WHERE mach_id=65537;
```

## Оператор DELETE

[WITH запросы\_with] DELETE FROM таблица [WHERE условие\_удаления]

Оператор DELETE позволяет удалять записи из таблицы. При необходимости можно использовать условие удаления с ключевым словом WHERE.

Пример:

```
DELETE FROM report WHERE type=7;
```

Все перечисленные операторы имеют возможности по использованию дополнительных параметров, с которыми необходимо ознакомиться в документации.

### 3. Задание

1. Создайте по крайней мере 3 связанные таблицы. Должны быть определены первичные и внешние ключи для таблиц, т.е. по крайней мере для одной пары таблиц должна быть определена связь 1:М
2. Наполнить таблицы базы данных при помощи операторов INSERT. Каждая таблица должна иметь не менее 5 разных записей.
3. Обновить записи в одной таблице на основании записи из другой (между таблицами должна быть связь).

4. Удалить несколько записей из одной таблицы на основании информации из другой таблицы.
5. Вывести часть столбцов из таблицы.
6. Вывести несколько записей из таблицы, используя условие ограничения.
7. Сделать декартово произведение двух таблиц.
8. Вывести записи из таблицы на основании условия ограничения, содержащегося в другой таблице.
9. Применить функции агрегирования к выводимым записям (sum, avg, min, max)
10. Вывести записи из таблицы, используя сортировку от большего к меньшему.
11. Вывести записи из таблицы, используя сортировку от меньшего к большему с ограничением количества выводимых строк.
12. Произвести агрегирование выводимых записей по одному из полей ( group by).
13. Выполнить запрос, когда табличное выражение представляет собой другой запрос.

#### ХОД РАБОТЫ:

Мы создадим **5 таблиц**:

1. **customers** — Клиенты магазина.
2. **orders** — Заказы, сделанные клиентами.
3. **products** — Товары (iPhone, MacBook, AirPods и т. д.).
4. **order\_items** — Какие товары включены в заказ.
5. **categories** — Категории товаров (iPhone, MacBook, iPad и др.).

#### 1) Создание таблиц

-- 1. Таблица клиентов

CREATE TABLE customers (

```
customer_id SERIAL PRIMARY KEY,  
name VARCHAR(100) NOT NULL,  
email VARCHAR(100) UNIQUE NOT NULL,  
phone VARCHAR(15) UNIQUE NOT NULL  
);
```

-- 2. Таблица заказов (связана с customers)

```
CREATE TABLE orders (
    order_id SERIAL PRIMARY KEY,
    customer_id INT REFERENCES customers(customer_id) ON DELETE CASCADE,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_price DECIMAL(10,2) DEFAULT 0 CHECK (total_price >= 0)
);
```

-- 3. Таблица категорий товаров (iPhone, MacBook и т. д.)

```
CREATE TABLE categories (
    category_id SERIAL PRIMARY KEY,
    category_name VARCHAR(50) UNIQUE NOT NULL
);
```

-- 4. Таблица товаров (связана с categories)

```
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    category_id INT REFERENCES categories(category_id) ON DELETE CASCADE,
    product_name VARCHAR(100) NOT NULL,
    price DECIMAL(10,2) NOT NULL CHECK (price > 0),
    stock_quantity INT NOT NULL CHECK (stock_quantity >= 0)
);
```

-- 5. Таблица товаров в заказе (связана с orders и products)

```
CREATE TABLE order_items (
    item_id SERIAL PRIMARY KEY,
    order_id INT REFERENCES orders(order_id) ON DELETE CASCADE,
    product_id INT REFERENCES products(product_id) ON DELETE CASCADE,
    quantity INT NOT NULL CHECK (quantity > 0),
    price DECIMAL(10,2) NOT NULL CHECK (price > 0)
);
```

```

postgres=# -- 1. Таблица клиентов
postgres=# CREATE TABLE customers (
postgres(#     customer_id SERIAL PRIMARY KEY,
postgres(#     name VARCHAR(100) NOT NULL,
postgres(#     email VARCHAR(100) UNIQUE NOT NULL,
postgres(#     phone VARCHAR(15) UNIQUE NOT NULL
postgres(# );
CREATE TABLE
postgres=#
postgres=# -- 2. Таблица заказов (связана с customers)
postgres=# CREATE TABLE orders (
postgres(#     order_id SERIAL PRIMARY KEY,
postgres(#     customer_id INT REFERENCES customers(customer_id) ON DELETE CASCADE,
postgres(#     order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
postgres(#     total_price DECIMAL(10,2) DEFAULT 0 CHECK (total_price >= 0)
postgres(# );
CREATE TABLE
postgres=#
postgres=# -- 3. Таблица категорий товаров (iPhone, MacBook и т. д.)
postgres=# CREATE TABLE categories (
postgres(#     category_id SERIAL PRIMARY KEY,
postgres(#     category_name VARCHAR(50) UNIQUE NOT NULL
postgres(# );
CREATE TABLE
postgres=#
postgres=# -- 4. Таблица товаров (связана с categories)
postgres=# CREATE TABLE products (
postgres(#     product_id SERIAL PRIMARY KEY,
postgres(#     category_id INT REFERENCES categories(category_id) ON DELETE CASCADE,
postgres(#     product_name VARCHAR(100) NOT NULL,
postgres(#     price DECIMAL(10,2) NOT NULL CHECK (price > 0),
postgres(#     stock_quantity INT NOT NULL CHECK (stock_quantity >= 0)
postgres(# );
CREATE TABLE
postgres=#
postgres=# -- 5. Таблица товаров в заказе (связана с orders и products)
postgres=# CREATE TABLE order_items (
postgres(#     item_id SERIAL PRIMARY KEY,
postgres(#     order_id INT REFERENCES orders(order_id) ON DELETE CASCADE,
postgres(#     product_id INT REFERENCES products(product_id) ON DELETE CASCADE,
postgres(#     quantity INT NOT NULL CHECK (quantity > 0),
postgres(#     price DECIMAL(10,2) NOT NULL CHECK (price > 0)
postgres(# );
CREATE TABLE

```

## 2) Наполнение таблиц данными

```

INSERT INTO customers (name, email, phone) VALUES
('Иван Иванов', 'ivan@example.com', '+79160000001'),
('Анна Смирнова', 'anna@example.com', '+79160000002'),
('Пётр Петров', 'petr@example.com', '+79160000003'),
('Мария Соколова', 'maria@example.com', '+79160000004'),
('Алексей Фёдоров', 'alex@example.com', '+79160000005');

```

```
postgres=# INSERT INTO customers (name, email, phone) VALUES
postgres-# ('Иван Иванов', 'ivan@example.com', '+79160000001'),
postgres-# ('Анна Смирнова', 'anna@example.com', '+79160000002'),
postgres-# ('Пётр Петров', 'petr@example.com', '+79160000003'),
postgres-# ('Мария Соколова', 'maria@example.com', '+79160000004'),
postgres-# ('Алексей Фёдоров', 'alex@example.com', '+79160000005');
INSERT 0 5
```

Добавляем категории товаров

```
INSERT INTO categories (category_name) VALUES
('iPhone'),
('MacBook'),
('iPad'),
('AirPods'),
('Accessories');
```

```
postgres=# INSERT INTO categories (category_name) VALUES
postgres-# ('iPhone'),
postgres-# ('MacBook'),
postgres-# ('iPad'),
postgres-# ('AirPods'),
postgres-# ('Accessories');
INSERT 0 5
```

Добавляем товары

```
INSERT INTO products (category_id, product_name, price, stock_quantity) VALUES
-- iPhone
(1, 'iPhone 15 Pro Max', 139990.00, 10),
(1, 'iPhone 15 Pro', 129990.00, 15),
(1, 'iPhone 15', 99990.00, 20),
(1, 'iPhone 14', 89990.00, 30),
-- MacBook
(2, 'MacBook Pro 16 M3', 299990.00, 5),
(2, 'MacBook Air 13 M2', 149990.00, 10),
-- iPad
(3, 'iPad Pro 12.9 M2', 129990.00, 8),
(3, 'iPad Air 10.9', 79990.00, 12),
-- AirPods
(4, 'AirPods Pro 2', 29990.00, 25),
(4, 'AirPods 3', 19990.00, 30),
-- Accessories
```

```
(5, 'Apple Magic Keyboard', 34990.00, 10),
```

```
(5, 'Apple Watch Ultra 2', 89990.00, 7);
```

```
postgres=# INSERT INTO products (category_id, product_name, price, stock_quantity) VALUES
postgres-# -- iPhone
postgres-# (1, 'iPhone 15 Pro Max', 139990.00, 10),
postgres-# (1, 'iPhone 15 Pro', 129990.00, 15),
postgres-# (1, 'iPhone 15', 99990.00, 20),
postgres-# (1, 'iPhone 14', 89990.00, 30),
postgres-# -- MacBook
postgres-# (2, 'MacBook Pro 16 M3', 299990.00, 5),
postgres-# (2, 'MacBook Air 13 M2', 149990.00, 10),
postgres-# -- iPad
postgres-# (3, 'iPad Pro 12.9 M2', 129990.00, 8),
postgres-# (3, 'iPad Air 10.9', 79990.00, 12),
postgres-# -- AirPods
postgres-# (4, 'AirPods Pro 2', 29990.00, 25),
postgres-# (4, 'AirPods 3', 19990.00, 30),
postgres-# -- Accessories
postgres-# (5, 'Apple Magic Keyboard', 34990.00, 10),
postgres-# (5, 'Apple Watch Ultra 2', 89990.00, 7);
INSERT 0 12
```

Добавляем заказы

```
INSERT INTO orders (customer_id, order_date) VALUES
```

```
(1, '2024-03-10 10:00:00'),
```

```
(2, '2024-03-11 12:30:00'),
```

```
(3, '2024-03-12 15:45:00'),
```

```
(4, '2024-03-13 18:00:00'),
```

```
(5, '2024-03-14 20:15:00');
```

```
postgres=# INSERT INTO orders (customer_id, order_date) VALUES
postgres-# (1, '2024-03-10 10:00:00'),
postgres-# (2, '2024-03-11 12:30:00'),
postgres-# (3, '2024-03-12 15:45:00'),
postgres-# (4, '2024-03-13 18:00:00'),
postgres-# (5, '2024-03-14 20:15:00');
INSERT 0 5
```

Добавляем товары в заказы

```
INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
```

```
(1, 1, 1, 139990.00), -- iPhone 15 Pro Max
```

```
(1, 10, 1, 19990.00), -- AirPods 3
```

```
(2, 6, 1, 149990.00), -- MacBook Air 13
```

```
(3, 3, 1, 99990.00), -- iPhone 15
```

```
(3, 9, 1, 29990.00), -- AirPods Pro 2
```

```
(4, 7, 1, 129990.00), -- iPad Pro 12.9
```

```
(5, 12, 1, 89990.00); -- Apple Watch Ultra 2
```

```

postgres=# INSERT INTO order_items (order_id, product_id, quantity, price) VALUES
postgres-# (1, 1, 1, 139990.00), -- iPhone 15 Pro Max
postgres-# (1, 10, 1, 19990.00), -- AirPods 3
postgres-# (2, 6, 1, 149990.00), -- MacBook Air 13
postgres-# (3, 3, 1, 99990.00), -- iPhone 15
postgres-# (3, 9, 1, 29990.00), -- AirPods Pro 2
postgres-# (4, 7, 1, 129990.00), -- iPad Pro 12.9
postgres-# (5, 12, 1, 89990.00); -- Apple Watch Ultra 2
INSERT 0 7

```

3) Вывести часть столбцов

SELECT product\_name, price FROM products;

```

postgres=# SELECT product_name, price FROM products;
   product_name    |   price
-----+-----
iPhone 15 Pro Max | 139990.00
iPhone 15 Pro     | 129990.00
iPhone 15          | 99990.00
iPhone 14          | 89990.00
MacBook Pro 16 M3  | 299990.00
MacBook Air 13 M2 | 149990.00
iPad Pro 12.9 M2  | 129990.00
iPad Air 10.9     | 79990.00
AirPods Pro 2     | 29990.00
AirPods 3          | 19990.00
Apple Magic Keyboard | 34990.00
Apple Watch Ultra 2 | 89990.00
(12 rows)

```

4) Вывести несколько записей с условием

SELECT \* FROM products WHERE price > 100000;

SELECT \* FROM products WHERE price > 100000;

```

postgres=# SELECT * FROM products WHERE price > 100000;
   product_id | category_id |   product_name    |   price | stock_quantity
-----+-----+-----+-----+-----+
        1 |         1 | iPhone 15 Pro Max | 139990.00 |          10
        2 |         1 | iPhone 15 Pro     | 129990.00 |          15
        5 |         2 | MacBook Pro 16 M3 | 299990.00 |           5
        6 |         2 | MacBook Air 13 M2 | 149990.00 |          10
        7 |         3 | iPad Pro 12.9 M2  | 129990.00 |           8
(5 rows)

```

5) Вывод части столбцов

SELECT name, email FROM customers;

6) Декартово произведение

```
SELECT customers.name, products.product_name
```

```
FROM customers, products;
```

name	product_name
???? ?????	iPhone 15 Pro Max
???? ???????	iPhone 15 Pro Max
???? ?????	iPhone 15 Pro Max
????? ????????	iPhone 15 Pro Max
?????? ????????	iPhone 15 Pro Max
????? ????????	iPhone 15 Pro
????? ????????	iPhone 15
????? ????????	iPhone 14
????? ????????	MacBook Pro 16 M3
????? ????????	MacBook Pro 16 M3
????? ????????	MacBook Pro 16 M3
????? ????????	MacBook Pro 16 M3
????? ????????	MacBook Pro 16 M3
????? ????????	MacBook Air 13 M2
????? ????????	MacBook Air 13 M2
????? ????????	MacBook Air 13 M2
????? ????????	MacBook Air 13 M2
????? ????????	MacBook Air 13 M2
????? ????????	iPad Pro 12.9 M2
????? ????????	iPad Pro 12.9 M2
????? ????????	iPad Pro 12.9 M2
????? ????????	iPad Pro 12.9 M2
????? ????????	iPad Pro 12.9 M2
????? ????????	iPad Air 10.9
????? ????????	AirPods Pro 2
????? ????????	AirPods 3
????? ????????	Apple Magic Keyboard

— More — |

7) Вывести записи из одной таблицы, используя данные из другой

```
SELECT customers.name, orders.order_id, orders.order_date
```

```
FROM customers
```

```
JOIN orders ON customers.customer_id = orders.customer_id;
```

order_id	order_date
1	2024-03-10 10:00:00
2	2024-03-11 12:30:00
3	2024-03-12 15:45:00
4	2024-03-13 18:00:00
5	2024-03-14 20:15:00

8) Агрегатные функции (SUM, AVG, MIN, MAX)

```
SELECT SUM(price) AS total_revenue, AVG(price) AS average_price,
```

```
MIN(price) AS min_price, MAX(price) AS max_price
```

```
FROM order_items;
```

total_revenue	average_price	min_price	max_price
659930.00	94275.714285714286	19990.00	149990.00
(1 row)			

9) Сортировка от большего к меньшему

```
SELECT * FROM products ORDER BY price DESC;
```

product_id	category_id	product_name	price	stock_quantity
5	2	MacBook Pro 16 M3	299990.00	5
6	2	MacBook Air 13 M2	149990.00	10
1	1	iPhone 15 Pro Max	139990.00	10
7	3	iPad Pro 12.9 M2	129990.00	8
2	1	iPhone 15 Pro	129990.00	15
3	1	iPhone 15	99990.00	20
4	1	iPhone 14	89990.00	30
12	5	Apple Watch Ultra 2	89990.00	7
8	3	iPad Air 10.9	79990.00	12
11	5	Apple Magic Keyboard	34990.00	10
9	4	AirPods Pro 2	29990.00	25
10	4	AirPods 3	19990.00	30
(12 rows)				

10) Сортировка от меньшего к большему с ограничением

```
SELECT * FROM products ORDER BY price ASC LIMIT 3;
```

product_id	category_id	product_name	price	stock_quantity
10	4	AirPods 3	19990.00	30
9	4	AirPods Pro 2	29990.00	25
11	5	Apple Magic Keyboard	34990.00	10
(3 rows)				

## 11) Группировка по категориям

```
SELECT categories.category_name, COUNT(products.product_id) AS product_count
FROM categories
JOIN products ON categories.category_id = products.category_id
GROUP BY categories.category_name;
```

category_name	product_count
iPad	2
iPhone	4
Accessories	2
AirPods	2
MacBook	2
(5 rows)	

## 12) Использование WITH

```
WITH order_summary AS (
    SELECT order_id, SUM(price * quantity) AS total_order_price
    FROM order_items
    GROUP BY order_id
)
SELECT customers.name, order_summary.total_order_price
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id
JOIN order_summary ON orders.order_id = order_summary.order_id;
```

```
postgres=# WITH order_summary AS (
postgres(#     SELECT order_id, SUM(price * quantity) AS total_order_price
postgres(#     FROM order_items
postgres(#     GROUP BY order_id
postgres(# )
postgres# SELECT customers.name, order_summary.total_order_price
postgres# FROM customers
postgres# JOIN orders ON customers.customer_id = orders.customer_id
postgres# JOIN order_summary ON orders.order_id = order_summary.order_id;
      name       | total_order_price
-----+-----
    ???  ?????? |      159980.00
    ???  ??????? |      149990.00
    ???  ?????? |      129980.00
    ???  ??????? |      129990.00
    ???  ??????? |      89990.00
(5 rows)
```

Вывод:

Я проделал полный цикл работы с базой данных **Apple Store** в PostgreSQL, начиная с проектирования структуры, наполнения данными, выполнения сложных SQL-запросов и заканчивая удалением всех данных.