



Передовые
инженерные
школы



МИНОБРНАУКИ
РОССИИ



УНИВЕРСИТЕТ
ИННОПОЛИС

Лекция 1.3

SQL команды SELECT и основы фильтрации данных

- SQL команды SELECT и FROM.
- Операторы сравнения.
- Логические операции.
- Фильтрация данных с SQL командой WHERE.
- Обработка NULL значений.

Виды запросов в SQL

Существуют 4 вида запросов:

- **DDL (data definition language)** – для определения данных. Позволяют настраивать БД – создавать с нуля и прописывать ее структуру.

Примеры DDL-запросов: CREATE, DROP, RENAME, ALTER.

- **DML (data manipulation language)** – для управления данными в таблицах. Помогают добавлять, обновлять, удалять и выбирать данные.

Примеры DML-запросов: SELECT, UPDATE, DELETE, INSERT.

- **DCL (data control language)** – для выдачи и отзыва права доступа для пользователей.

Примеры DCL-запросов: GRANT, REVOKE, DENY.

- **TCL (transaction control language)** – для управления транзакциями. Относятся запросы, связанные с подтверждением или откатом изменений в БД.

Примеры TCL-запросов: COMMIT, ROLLBACK, BEGIN.

Основные операции с таблицами

<https://sqliteonline.com/>

Для создания таблиц в языке SQL служит команда CREATE TABLE. Ее полный синтаксис представлен в документации на PostgreSQL, а упрощенный синтаксис таков:

```
CREATE TABLE имя-таблицы  
(  
    имя-поля тип-данных [ограничения-целостности],  
    имя-поля тип-данных [ограничения-целостности],  
    ...  
    имя-поля тип-данных [ограничения-целостности],  
    [ограничение-целостности],  
    [первичный-ключ],  
    [внешний-ключ]  
);
```

В квадратных скобках показаны необязательные элементы команды. После команды нужно поставить символ «;».

Любая команда должна начинаться с ключевого слова — или действия, которое должно произойти. Например, выбрать строку, вставить новую, изменить старую или удалить таблицу целиком.

Одно из таких ключевых слов — **SELECT**.

SQL команда «**SELECT**» - это основное средство для извлечения данных из реляционных баз данных. Она позволяет выбирать один или несколько столбцов из одной или нескольких таблиц.

Пример классического SQL-запроса, состоящего из шести операторов (два из них обязательные, а другие четыре — используются по обстоятельствам):

- **SELECT** — выбирает отдельные столбцы или всю таблицу целиком (обязательный);
- **FROM** — из какой таблицы получить данные (обязательный);
- **WHERE** — условие, по которому SQL выбирает данные;
- **GROUP BY** — столбец, по которому мы будем группироваться данные;
- **HAVING** — условие, по которому сгруппированные данные будут отфильтрованы;
- **ORDER BY** — столбец, по которому данные будут отсортированы;

Полный синтаксис команды `SELECT` включает в себя следующие элементы:

SELECT [DISTINCT] column1, column2, ...

FROM table_name

[WHERE condition]

[ORDER BY column1, column2, ... [ASC | DESC]]

[GROUP BY column1, column2, ...]

DISTINCT (необязательное ключевое слово) используется для выбора уникальных значений столбцов. Например, `SELECT DISTINCT city FROM customers;` вернет уникальные значения городов из таблицы `customers`.

Операторы сравнения SQL

Оператор	Описание	Пример
=	Оператор сравнения на равенство. Если равенство верное, то результат – TRUE, иначе – FALSE.	(4 = 2+2) Результат TRUE
!=	Оператор сравнения на неравенство. True – если значения не равны.	(8!=12) Результат True
<>	Аналогичный предыдущему SQL оператор. TRUE мы получим в том случае, если значения будут не равны.	(7<>14) Результат True
>	Оператор “больше”. Если левая часть больше правой, то True.	(7>2) Результат True
<	Оператор “меньше”. Значение True, если правый операнд больше левого.	(3<4) Результат True
>=	Оператор “больше либо равно”. Выдает значение TRUE, если правая часть больше либо равна левой.	(7 >= 7) Результат TRUE
<=	Оператор “меньше либо равно”. Значение TRUE появляется тогда, когда правая часть больше либо равна левой.	(5<=9) Результат TRUE

Логические операторы

Оператор	Описание
AND	SQL оператор AND представляет собой получение результата при соблюдении двух поставленных условий.
ANY	SQL оператор ANY (любой). Осуществляет сравнение текущего задания с дополнительным запросом.
BETWEEN	SQL оператор BETWEEN. Для этого оператора SQL условия можно установить в определённом диапазоне. Для корректной работы нужно задать минимальное и максимальное значение.
EXISTS	SQL оператор EXISTS. Применяется тогда, когда нужно обозначить, интересует ли пользователя результат подзапроса.
IN	Простой SQL оператор IN указывает, с какими значениями нужно вывести строки.
LIKE	Популярный SQL оператор LIKE. Данный оператор позволяет осуществлять поиск подстроки в тексте и, если подстрока найдена, то она выводится.

Логические операторы

Оператор	Описание
NOT	SQL оператор отрицания NOT. Аннулирует любые условия.
OR	SQL оператор «ИЛИ». Дает результат в том случае, когда значение TRUE есть хотя бы в одном из операндов.
IS NULL	SQL оператор IS NULL позволяет проверить значение поля на NULL.

Фильтрация данных с SQL командой **WHERE**



ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Команда **WHERE** позволяет фильтровать строки, выбираемые из таблицы, на основе заданных условий. Условие может содержать операторы сравнения, логические операторы и функции.

Синтаксис:

WHERE <search_condition>

<search_condition> определяет условия, которые должны быть выполнены для всех возвращаемых строк.

Примеры использования WHERE

Для начала создадим таблицу 'orders' и заполним ее данными:

```
CREATE TABLE orders (
```

```
    order_id SERIAL,
```

```
    customer_id INT,
```

```
    order_date DATE,
```

```
    order_amount DECIMAL(10, 2),
```

```
    order_status VARCHAR(20)
```

```
);
```

```
INSERT INTO orders (customer_id, order_date, order_amount, order_status)
```

```
VALUES (1, '2023-01-15', 100.50, 'Delivered'),
```

```
    (2, '2023-02-10', 75.20, 'Shipped'),
```

```
    (3, '2023-03-05', 200.00, 'Delivered'),
```

```
    (1, '2023-04-20', 50.75, 'Pending'),
```

```
    (4, '2023-05-10', 300.00, 'Delivered');
```

Теперь таблица имеет вид:

```
postgres=# SELECT * FROM orders;
 order_id | customer_id | order_date | order_amount | order_status
-----+-----+-----+-----+-----
         1 |           1 | 2023-01-15 |       100.50 | Delivered
         2 |           2 | 2023-02-10 |        75.20 | Shipped
         3 |           3 | 2023-03-05 |       200.00 | Delivered
         4 |           1 | 2023-04-20 |        50.75 | Pending
         5 |           4 | 2023-05-10 |       300.00 | Delivered
(5 строк)
```

- Нахождение строки с помощью простого равенства:

Выбор всех заказов с определенным статусом:

```
postgres=# SELECT * FROM orders
postgres=# WHERE order_status = 'Delivered';
 order_id | customer_id | order_date | order_amount | order_status
-----+-----+-----+-----+-----
         1 |           1 | 2023-01-15 |         100.50 | Delivered
         3 |           3 | 2023-03-05 |         200.00 | Delivered
         5 |           4 | 2023-05-10 |         300.00 | Delivered
(3 строки)
```

- Нахождение строк с использованием оператора сравнения:

Выбор заказов, созданных после определенной даты:

```
postgres=# SELECT * FROM orders
postgres=# WHERE order_date >= '2023-01-01';
 order_id | customer_id | order_date | order_amount | order_status
-----+-----+-----+-----+-----
         1 |           1 | 2023-01-15 |         100.50 | Delivered
         2 |           2 | 2023-02-10 |          75.20 | Shipped
         3 |           3 | 2023-03-05 |         200.00 | Delivered
         4 |           1 | 2023-04-20 |          50.75 | Pending
         5 |           4 | 2023-05-10 |         300.00 | Delivered
(5 строк)
```


Выбор заказов, сумма которых больше 100:

```
postgres=# SELECT * FROM orders
postgres=# WHERE order_amount > 100;
 order_id | customer_id | order_date | order_amount | order_status
-----+-----+-----+-----+-----
         1 |           1 | 2023-01-15 |         100.50 | Delivered
         3 |           3 | 2023-03-05 |         200.00 | Delivered
         5 |           4 | 2023-05-10 |         300.00 | Delivered
(3 строки)
```

- Нахождение строк, удовлетворяющих любому из условий:

Выбор заказов, созданных до 15 апреля 2023 года или со статусом "Pending":

```
postgres=# SELECT * FROM orders
postgres=# WHERE order_date < '2023-04-15' OR order_status = 'Pending';
 order_id | customer_id | order_date | order_amount | order_status
-----+-----+-----+-----+-----
         1 |           1 | 2023-01-15 |         100.50 | Delivered
         2 |           2 | 2023-02-10 |          75.20 | Shipped
         3 |           3 | 2023-03-05 |         200.00 | Delivered
         4 |           1 | 2023-04-20 |          50.75 | Pending
(4 строки)
```

- Нахождение строк, которые должны удовлетворять нескольким условиям:

Выбор заказов, созданных после 1 марта 2023 года и со статусом "Delivered":

```
postgres=# SELECT * FROM orders
postgres=# WHERE order_date > '2023-03-01' AND order_status = 'Delivered';
 order_id | customer_id | order_date | order_amount | order_status
-----+-----+-----+-----+-----
          3 |          3 | 2023-03-05 |         200.00 | Delivered
          5 |          4 | 2023-05-10 |         300.00 | Delivered
(2 строки)
```

Обработка NULL значений

NULL – это специальное значение, которое указывает на отсутствие данных или неопределенное значение в столбце базы данных.

Обработка NULL значений является важной частью SQL-запросов, так как они могут оказать влияние на результаты запроса.

Почему знание NULL важно для SQL - разработчиков

- Необходимо убедиться, что данные в таблице корректны и не содержат NULL, если это не предусмотрено требованиями для соответствующего столбца таблицы.
- Знание правильной работы с NULL в SQL позволяет избежать неожиданного поведения запросов и операторов, которые могут привести к ошибкам или неверным результатам.
- Понимание того, как обрабатывать значения NULL, может улучшить эффективность запросов, так как правильное использование функций для работы с NULL может сократить количество кода и убрать дублирование.

IS NULL & IS NOT NULL

IS NULL – оператор используется для проверки, является ли значение NULL. Возвращает истину, если операнда является NULLом. Соответственно, если операнд не является NULLом, то значение будет ложным.

IS NOT NULL – оператор используется для проверки, не является ли значение NULL. Значение будет истинным, если операнд не является NULLом, и ложным, если он таковым является.

Пример обработки NULL значений

Создадим таблицу «Employees» и заполним ее данными:

```
CREATE TABLE Employees (  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    phone_number VARCHAR(15)  
);
```

```
INSERT INTO Employees (first_name, last_name, phone_number)  
VALUES  
    ('Иван', 'Иванов', '123-456-7890'),  
    ('Петр', 'Петров', NULL),  
    ('Мария', 'Сидорова', '987-654-3210');
```

Теперь таблица имеет вид:

```
postgres=# SELECT * FROM employees;  
first_name | last_name | phone_number  
-----+-----+-----  
Иван      | Иванов   | 123-456-7890  
Петр      | Петров   |  
Мария     | Сидорова | 987-654-3210  
(3 строки)
```


Пример использования **IS NULL**:

```
postgres=# SELECT first_name, last_name
postgres=# FROM employees
postgres=# WHERE phone_number IS NULL;
 first_name | last_name
-----+-----
    Петр   | Петров
(1 строка)
```

Пример использования **IS NOT NULL**:

```
postgres=# SELECT first_name, last_name
postgres=# FROM employees
postgres=# WHERE phone_number IS NOT NULL;
 first_name | last_name
-----+-----
    Иван   | Иванов
    Мария  | Сидорова
(2 строки)
```

COALESCE – это специальное выражение, которое вычисляет по порядку каждый из своих аргументов и на выходе возвращает значение первого аргумента, который был не NULL.

```
SELECT COALESCE(description, short_description, '(none)') ...
```

Этот запрос вернёт значение **description**, если оно не равно NULL, либо **short_description**, если оно не NULL, и строку (**none**), если оба эти значения равны NULL

Примеры COALESCE



```
postgres=# SELECT COALESCE(NULL, NULL, 1, 2, NULL, 3);
 coalesce 
-----
          1
(1 строка)
```

```
postgres=# SELECT COALESCE(phone_number, '000-000-0000') FROM Employees;
 coalesce 
-----
123-456-7890
000-000-0000
987-654-3210
(3 строки)
```

Задание 1

Создадим таблицу 'students' и заполним ее данными:

```
CREATE TABLE students (  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    age INT  
);  
  
INSERT INTO students (first_name, last_name, age)  
VALUES ('Иван', 'Иванов', 20),  
      ('Петр', 'Петров', NULL),  
      ('Карина', 'Тимофеева', 21),  
      ('Алексей', 'Бирюк', NULL),  
      ('Мария', 'Корецкая', 20);
```

Теперь таблица имеет вид:

```
postgres=# SELECT * FROM students;  
first_name | last_name | age  
-----+-----+-----  
Иван       | Иванов   | 20  
Петр       | Петров   |  
Карина     | Тимофеева | 21  
Алексей    | Бирюк    | 18  
Мария      | Корецкая | 20  
(5 строк)
```

Задания

1. Выбор всех данных из таблицы.
2. Выбор имени и фамилии студентов.
3. Выбор студентов, возраст которых больше 18.
4. Выбор студентов с именем "Иван".
5. Выбор студентов, у которых не указан возраст.
6. Выбор студентов, у которых указан возраст.
7. Выбор студентов с именем “Иван” или “Алексей” и возрастом более 19 лет.
8. Вывести имя и возраст студента, если возраст отсутствует, то вывести 99.

Решение:

-- 1

```
SELECT * FROM students;
```

-- 2

```
SELECT first_name, last_name  
FROM students;
```

-- 3

```
SELECT * FROM students  
WHERE age > 18;
```

-- 4

```
SELECT first_name, age FROM  
students  
WHERE first_name = 'Иван';
```

-- 5

```
SELECT * FROM students  
WHERE age IS NULL;
```

-- 6

```
SELECT * FROM students  
WHERE age IS NOT NULL;
```

-- 7. Выбор студентов с именем “Иван” или “Алексей” и возрастом более 19 лет.

```
SELECT * FROM students  
WHERE (first_name = 'Иван' OR first_name = 'Алексей') AND  
age > 19;
```

-- 8. Вывести имя и возраст студента, если возраст отсутствует, то вывести 99.

```
SELECT first_name, COALESCE(age, 99)  
FROM students;
```


Домашнее задание

Создайте таблицу и заполните ее значениями, включая NULL-значения.

Выполните любые 5 запросов с фильтрацией командой WHERE и 3 запроса с обработкой параметра NULL (IS NULL, IS NOT NULL, COALESCE).

Формат сдачи: word/pdf, где будет запрос и результат запроса.

*подробности формата сдачи уточнять у своего наставника.

Для самостоятельного изучения:

- <https://sql-ex.ru/>
- “SQL для простых смертных” - Мартин Грабер



Передовые
инженерные
школы



МИНОБРНАУКИ
РОССИИ



УНИВЕРСИТЕТ
ИННОПОЛИС

Спасибо за внимание

