# Урок 3



## Фильтрация (WHERE)

Фильтрация данных является одной из ключевых задач при работе с базами данных. Она необходима для выбора одной или более строк по заданному условию. Тем самым фильтрация позволяет сосредоточиться только на интересующих нас данных.

Общий вид запроса с фильтрацией:

```
SELECT *             -- выбор полей
  FROM table         -- выбор таблицы
 WHERE <условие>     -- условие
```

Выражение, записанное в качестве условия фильтрации, должно возвращать булево значение (True или False). Синтаксис похож на работу с булевыми переменными:

- **Равенство и неравенство** (`=`, `<>`)
- **Сравнения** (`>`, `>=`, `<`, `<=`)
- **Инструкция IN** — проверка вхождения значения в список значений

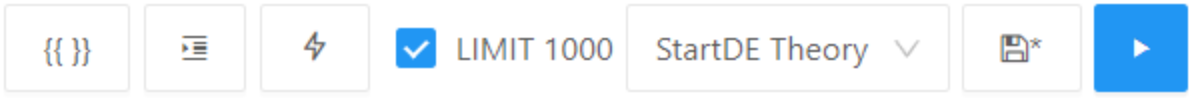
- **Инструкция NOT** — инверсия условия (`NOT IN`, `NOT BETWEEN`, `NOT LIKE`, `NOT EXISTS`, `IS NOT NULL`, а также комбинированные условия `NOT (department = 'HR' AND age >= 30)`)
- **Инструкция BETWEEN** — проверка вхождения значения в диапазон

Для фильтрации по нескольким условиям используются логические операторы `AND` и `OR`. При их использовании следует помнить, что **AND** имеет более высокий приоритет, чем **OR**.

**Пример использования фильтрации:**

```
SELECT * FROM characters
 WHERE species = 'Human' AND status = 'Alive'
 LIMIT 10;
```

```
1  SELECT * FROM characters
2   WHERE species = 'Human' AND status = 'Alive'
3   LIMIT 10;
```




Table

| id | name | status | species | type | gender |
|---|---|---|---|---|---|
| 1 | Rick Sanchez | Alive | Human | | Male |
| 2 | Morty Smith | Alive | Human | | Male |
| 3 | Summer Smith | Alive | Human | | Female |

Если в фильтрации используется несколько условий, то для явного указания приоритетности выполнения операций можно использовать скобки (без них СУБД будет выполнять операции согласно приоритетам, которые мы рассмотрели в предыдущем уроке).
Использование скобок важно для правильного выполнения запросов и получения ожидаемых результатов, также оно улучшает читаемость. Поэтому их лучше использовать всегда, даже если порядок выполнения с ними и без них будет одинаков.
Следующие два запроса выдадут одинаковый результат, однако запрос с явным указанием приоритетности с помощью скобок легче читается и интерпретируется:

```
SELECT * FROM employees
 WHERE department = 'Sales' OR department = 'Marketing' AND sala
```

```
SELECT * FROM employees
 WHERE department = 'Sales' OR (department = 'Marketing' AND sal
```

Запрос выберет всех сотрудников из отдела продаж и тех сотрудников из отдела маркетинга, у которых зарплата больше 50000.

А в следующем запросе скобки изменили порядок выполнения операций. Запрос выберет сотрудников, которые работают либо в отделе продаж, либо в отделе маркетинга, при этом зарплата которых зарплата больше 50000:

```
SELECT * FROM employees
 WHERE (department = 'Sales' OR department = 'Marketing') AND sa
```

💡 Для избегания ошибок рекомендуется задавать приоритет выполнения логических операторов с помощью круглых скобок

## Основные методы фильтрации текста

Особое внимание в SQL следует уделить фильтрации символьных типов данных, которая предоставляет множество возможностей для поиска и отбора строк, которые соответствуют определённым текстовым условиям. Использование операторов сравнения, шаблонов, встроенных функций и логических операторов позволяет создавать гибкие и мощные запросы для работы с текстовыми данными.

Наиболее часто используемые из них:

1. **Оператор равенства (`=`) и неравенства (`<>` или `!=`)**

   Оператор равенства используется для поиска строк, которые точно соответствуют указанному тексту.

```
SELECT * FROM customers
 WHERE city = 'New York';
```

Этот запрос выбирает всех клиентов, у которых город проживания указан как «New York».

Оператор неравенства используется для поиска строк, которые не соответствуют указанному тексту.

```
SELECT * FROM customers
 WHERE city != 'New York';
```

Этот запрос выбирает всех клиентов, у которых город проживания не «New York».

2. **Оператор `LIKE`**

   Оператор `LIKE` используется для поиска строк по шаблону. В нем используются два специальных символа:

   - `%`: заменяет любое количество символов (в том числе ноль символов).
   - `_`: заменяет один символ.

   Примеры:

```
SELECT * FROM customers
 WHERE name LIKE 'J%';
```

   Этот запрос выбирает всех клиентов, чьи имена начинаются с буквы «J».

```
SELECT * FROM customers
 WHERE email LIKE '%@gmail.com';
```

   Этот запрос выбирает всех клиентов, чьи email-адреса заканчиваются на «@gmail.com».

```
SELECT * FROM products
 WHERE code LIKE 'A_1';
```

Этот запрос выбирает все продукты, чей код начинается с буквы «A», имеет любой один символ на втором месте, и заканчивается на «1».

3. **Оператор `ILIKE` (в некоторых СУБД, например, PostgreSQL)**

   В некоторых системах управления базами данных (СУБД) оператор `ILIKE` используется для регистронезависимого поиска строк по шаблону. Например, в PostgreSQL:

   ```
   SELECT * FROM customers
    WHERE name ILIKE 'j%';
   ```

   Этот запрос выберет всех клиентов, чьи имена начинаются с буквы «j» или «J».

4. **Оператор `IN`**

   Оператор `IN` позволяет фильтровать строки, которые соответствуют одному из значений из списка.

   ```
   SELECT * FROM customers
    WHERE city IN ('New York', 'Los Angeles', 'Chicago');
   ```

   Этот запрос выбирает всех клиентов, которые живут в одном из трех указанных городов.

5. **Использование функций для работы с текстом**

   Многие СУБД предоставляют встроенные функции для работы с текстом, такие как `LOWER()`, `UPPER()`, `TRIM()`, `SUBSTRING()` и другие. Эти функции могут использоваться в условиях `WHERE` для выполнения более сложных фильтров.

   Примеры:

   - Преобразование текста к нижнему регистру для регистронезависимого поиска:

```
SELECT * FROM customers
 WHERE LOWER(name) = 'john doe';
```

- Поиск строк, содержащих определенную подстроку:

```
SELECT * FROM courses
 WHERE SUBSTRING(description, 1, 4) = 'Data';
```

- Удаление пробелов из начала и конца строки:

```
SELECT * FROM customers
 WHERE TRIM(name) = 'John Doe';
```

**Комбинирование условий**

Вы можете комбинировать различные условия фильтрации текста с помощью логических операторов `AND`, `OR`, и `NOT` для создания сложных запросов.

Пример:

```
SELECT * FROM customers
 WHERE (city = 'New York' OR city = 'Los Angeles')
   AND name LIKE 'J%'
   AND email LIKE '%@gmail.com';
```

Этот запрос выбирает всех клиентов, которые живут в Нью-Йорке или Лос-Анджелесе, их имена начинаются с буквы «J» и email-адреса заканчиваются на «@gmail.com».

Когда в SQL-запросе есть вычисляемые поля (поля, значения которых вычисляются во время выполнения запроса), для фильтрации по такому полю необходимо продублировать вычисляемое выражение. Использование алиаса в данном случае вызовет ошибку в СУБД.

Пример правильного использования:

```
SELECT
    product_id,
    price,
    discount,
    price * discount AS discounted_price
FROM
    products
WHERE
    price * discount > 3000;  -- Условие фильтрации дублирует
вычисляемое выражение
```

Пример неправильного использования:

```
SELECT
    product_id,
    price,
    discount,
    price * discount AS discounted_price
FROM
    products
WHERE
    discounted_price > 3000;  -- Использование алиаса вызовет
ошибку
```

💡 Не используйте алиасы для фильтрации по вычисляемым полям в SQL-запросах

# Сортировка (ORDER BY)

Сортировка данных позволяет упорядочивать строки в порядке возрастания или убывания. Сортировка происходит по одному или нескольким столбцам, указанным в SQL-запросе. Для этого используется инструкция `ORDER BY`:

- **ASC** (используется по умолчанию) — сортировка по возрастанию

- **DESC** — сортировка по убыванию


Общий вид запроса с сортировкой:

```
  SELECT *                          -- выбор полей
    FROM table                      -- выбор таблицы
   WHERE <условие>                  -- условие фильтрации
ORDER BY <поле_1> DESC, <поле_2> -- сортировка по убыванию поля
```

В качестве указания поля сортировки используется наименование поля, либо его алиас, либо порядковый номер поля (начиная с 1) среди всех полей к выводу, указанных в `SELECT`.

Сортировка чисел осуществляется в порядке возрастания (от меньшего к большему), а сортировка строковых типов данных осуществляется в лексикографическом порядке.

# Ограничение выборки (LIMIT, OFFSET)

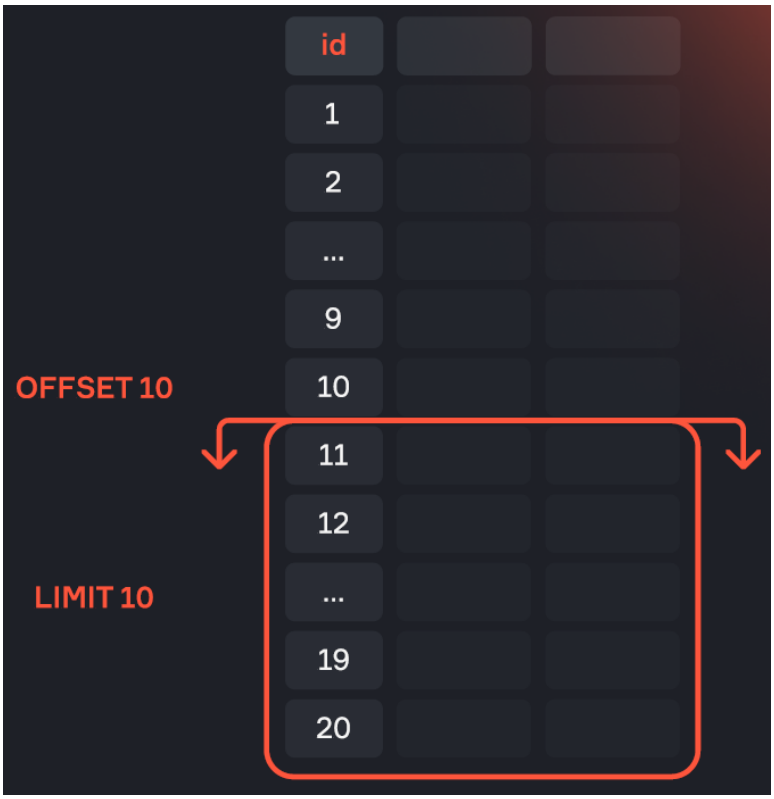
Для ограничения количества возвращаемых строк используется инструкция `LIMIT`. В сочетании с `OFFSET` она позволяет реализовать пагинацию

(разделение данных на страницы для удобства отображения и управления большим объемом информации).

Общий вид запроса с ограничением выборки:

```
  SELECT *                      -- выбор полей
    FROM table                  -- выбор таблицы
   WHERE <условие>              -- условие
ORDER BY <поле_1>, <поле_2>     -- сортировка
   LIMIT n1                     -- выводим n1 строк
  OFFSET n2                     -- пропускаем первые n2 строк
```

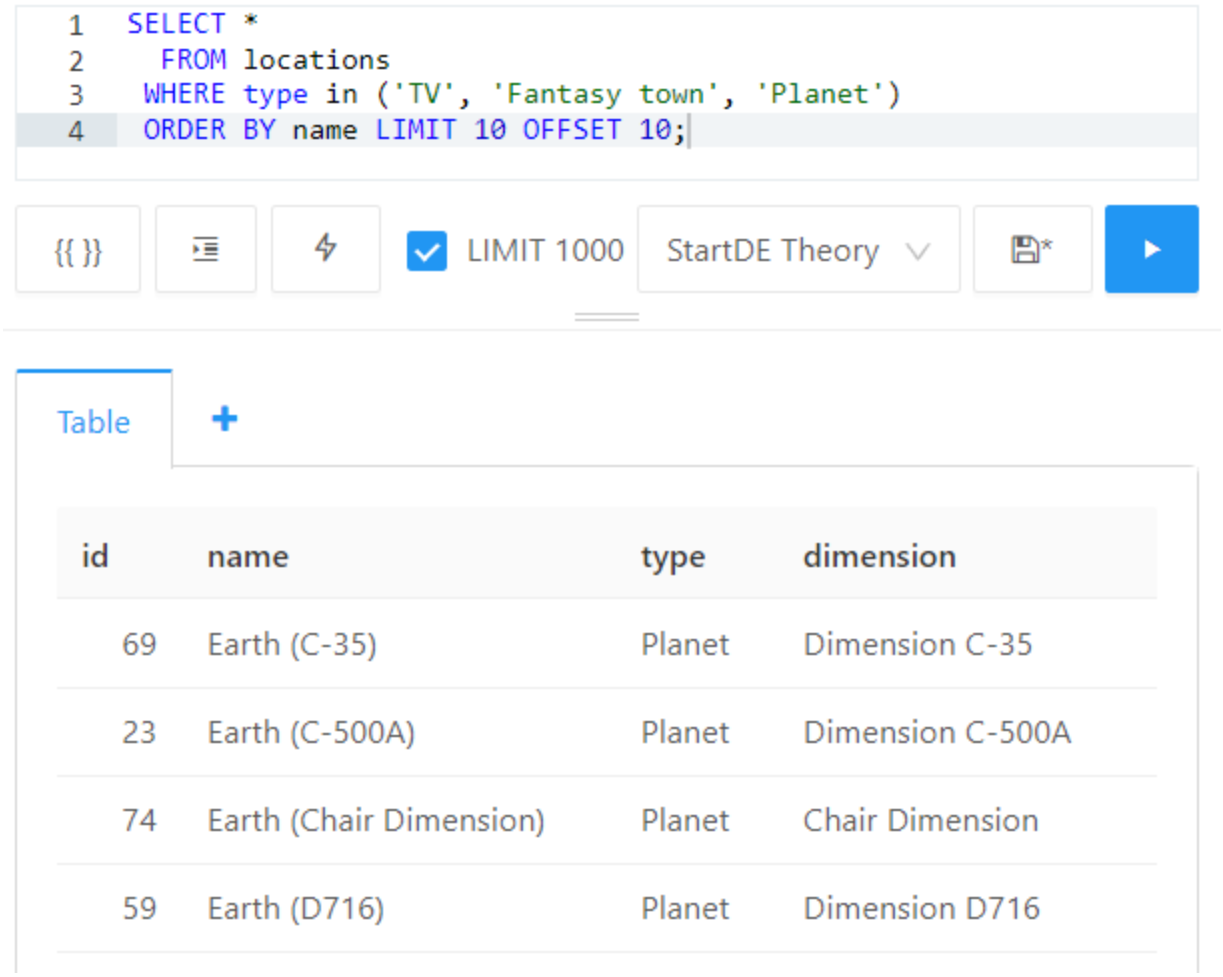
Например, `LIMIT 10 OFFSET 10` выберет 10 строк, начиная с 11-й.




**Пример использования сортировки и ограничения выборки:**

```
SELECT *
  FROM locations
```

```
WHERE type in ('TV', 'Fantasy town', 'Planet')
ORDER BY name LIMIT 10 OFFSET 10;
```




# Code Style

Хороший стиль написания SQL-запросов делает их более читаемыми и поддерживаемыми. Необходимо придерживаться следующих простых правил:

- **Инструкции** пишутся **заглавными буквами** (SELECT, FROM, WHERE, ORDER BY)

> 💡 Использование верхнего регистра для ключевых слов — это вопрос стиля и соглашений в команде. Это необязательное требование SQL, но широко принятое соглашение, которое помогает улучшить читаемость и поддерживаемость кода.

- **Каждая инструкция** начинается **с новой строки**
- **Использование пробелов** для выравнивания улучшает читаемость (правило «коридора», когда ключевые элементы выровнены и легко различимы)
- В сложных запросах можно использовать **комментарии** для лучшей читаемости:
  - `--` используется для комментария на одной строке
  - `/* */` используется для комментария на нескольких строках

  **Пример написания SQL-запроса с использованием правила «коридора»:**

```sql
SELECT *
  FROM locations
 WHERE type in ('TV', 'Fantasy town', 'Planet')
 ORDER BY name LIMIT 10 OFFSET 10;
```

# Работа с отсутствующими данными (NULL)

**NULL** — это специальное значение, которое обозначает отсутствие данных в ячейке таблицы. Оно отличается от пустой строки или нулевого значения и требует особого подхода при работе:

- **NULL не участвует в обычных сравнениях**. Вместо операторов сравнения необходимо использовать конструкции `IS NULL` и `IS NOT NULL`, которые возвращают True или False.

```
-- Выбор всех сотрудников, у которых не указана электронная п
SELECT *
  FROM employees
 WHERE email IS NULL;

-- Выбор всех сотрудников, у которых указана электронная почт
SELECT *
  FROM employees
 WHERE email IS NOT NULL;
```

- Если **при выполнении арифметических операций** используются поля со значениями `NULL` в строках, то результатом вычисления для данных строк будет также `NULL`.

```
-- Расчет итоговой цены с учетом скидки
SELECT
    sale_id,
    price,
    discount,
    price - discount AS final_price
FROM
    sales;
-- Для строк со значениями NULL в discount результат в final_
```

- **Функция** `COALESCE` возвращает первое не NULL значение из переданных ей аргументов. Она применится к каждому значению в колонке, и если это значение окажется NULL — заменяет его на значение, указанное вторым аргументом. В противном случае — функция просто вернёт значение колонки.

```
-- Выбор данных сотрудника с использованием COALESCE для обра
SELECT
    employee_id,
    first_name,
    COALESCE(last_name, '') AS last_name
```

```
FROM
    employees;
-- Если last_name NULL, то возвращается пустая строка
```

- **Сортировка.** В разных СУБД NULL может сортироваться как первое или последнее значение. Для явного указания необходимо использовать `NULLS FIRST` или `NULLS LAST`.

```
-- Сортировка продуктов по цене с NULL в начале
SELECT
    product_id,
    product_name,
    price
FROM
    products
ORDER BY
    price NULLS FIRST;
```

**Пример обработки `NULL` с помощью функции `COALESCE`:**

```
SELECT id, name, type,
       COALESCE(dimension, 'unknown')
  FROM locations
 WHERE type in ('TV', 'Fantasy town', 'Planet');
```

```sql
SELECT id, name, type,
       COALESCE(dimension, 'unknown')
  FROM locations
 WHERE type in ('TV', 'Fantasy town', 'Planet');
```

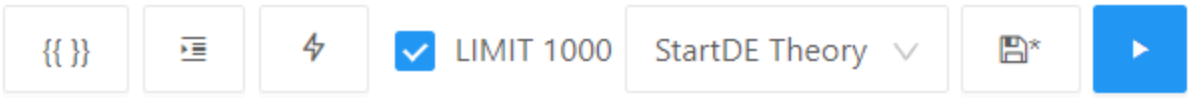



Table +

| id | name | type | coalesce |
|---|---|---|---|
| 1 | Earth (C-137) | Planet | Dimension C-137 |
| 4 | Worldender's lair | Planet | unknown |
| 6 | Interdimensional Cable | TV | unknown |
| 8 | Post-Apocalyptic Earth | Planet | Post-Apocalyptic Dimension |
| 9 | Purge Planet | Planet | Replacement Dimension |
| 10 | Venzenulon 7 | Planet | unknown |
| 11 | Bepis 9 | Planet | unknown |
| 12 | Cronenberg Earth | Planet | Cronenberg Dimension |

# Вопросы на собеседовании

1. **Что вернет** `NULL = NULL`**?**

**Ответ:** `NULL = NULL` вернет `NULL`, потому что сравнение с `NULL` всегда неопределенно.

2. **Что вернет `NULL OR TRUE`?**

   **Ответ:** `NULL OR TRUE` вернет `TRUE`, так как оператор `OR` возвращает true, если хотя бы одно из условий истинно.

3. **Что вернет `NULL AND TRUE`?**

   **Ответ:** `NULL AND TRUE` вернет `NULL`, так как результат зависит от неизвестного значения `NULL`.

4. **Как обрабатывается NULL при сортировке и как это изменить?**

   **Ответ:** В разных СУБД `NULL` может сортироваться по-разному. Для изменения порядка используйте `NULLS FIRST` или `NULLS LAST`.

# Резюме

- Для выбора строк по заданному условию необходимо использовать фильтрацию данных в запросе с помощью инструкции `WHERE`. Для объединения нескольких условий используются `AND` и `OR`. Приоритет выполнения `AND` выше, чем у `OR`. Для задания приоритета в явном виде логические выражения помещают в круглые скобочки.
- Для сортировки данных в запросе используется инструкция `ORDER BY`, после которой указываются поля сортировки и порядок (`ASC` — по возрастанию, `DESC` — по убыванию).
- Для ограничения количества возвращаемых строк используются инструкции `LIMIT` и `OFFSET`, которые позволяют реализовать пагинацию выдачи.
- Хорошей практикой при написании кода считается руководство общепринятыми негласными правилами:
  - написание инструкций заглавными буквами
  - написание инструкций с новой строки
  - использование выравнивания для читаемости

- NULL обозначает отсутствие данных (в отличие от цифры 0 или пустой строки ‘’). Особенности работы с NULL:
  - При сравнениях используются операторы `IS NULL` и `IS NOT NULL`
  - При участии `NULL` в качестве одного из слагаемых при выполнении арифметических операций результатом будет также `NULL`
  - Для замены значений `NULL` на константные значения используется функция `COALESCE`
  - При сортировке полей значения `NULL` могут сортироваться как первое или последнее значение в зависимости от СУБД

**К концу текущего урока вы можете написать SQL-запрос с использованием инструкций:**

| | |
|---|---|
| SELECT | Основной оператор для запроса данных из одной или нескольких таблиц. Позволяет выбрать конкретные столбцы и строки на основе заданных условий. |
| FROM | Указывает на таблицу в базе данных, к которой обращается запрос |
| WHERE | Указывает условия, по которым строки будут выбраны |
| ORDER BY | Упорядочивает строки в результате запроса по одному или нескольким столбцам. Может быть использован для сортировки по возрастанию (ASC) или убыванию (DESC) |
| LIMIT и OFFSET | LIMIT ограничивает количество возвращаемых строк. OFFSET пропускает указанное количество строк перед возвратом оставшихся строк |