## Язык запросов SQL: основы (на примере SQLite)

SQLite – это лёгкая реляционная система управления базами данных (СУБД). В отличие от «тяжёлых» СУБД (MySQL, PostgreSQL, Oracle), SQLite хранит все данные в одном файле и не требует отдельного сервера. Поэтому она часто используется:

- в приложениях для хранения локальных данных;
- в мобильной разработке (Android, iOS);
- для обучения SQL, так как запускается очень просто.

Для работы с SQLite используется стандартный язык структурированных запросов – **SQL (Structured Query Language)**. Этот язык универсален: его основные конструкции одинаковы во всех СУБД, различаются лишь расширения.

### Основные типы данных в SQLite

SQLite поддерживает гибкую систему типов. Основные:

**INTEGER** – целое число.
**REAL** – число с плавающей точкой.
**TEXT** – строка (текст).
**BLOB** – двоичные данные (например, картинка).
**NULL** – отсутствие значения.

Важно: SQLite не всегда строго проверяет типы, например, в поле `INTEGER` можно записать строку. Но для ясности и дисциплины всегда указывайте правильный тип.

### Создание таблицы

Для начала работы необходимо создать таблицу. Таблица – это структура для хранения данных в виде строк и столбцов.

Пример создадим таблицу для хранения данных студентов:

```
CREATE TABLE students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    age INTEGER,
    email TEXT UNIQUE);
```

Разберём пример:

`id` – уникальный идентификатор, автоинкрементируемый.

`name` – имя студента, обязательное (`NOT NULL`).

`age` – возраст (может быть `NULL`).

`email` – уникальный адрес (двух студентов с одинаковым email быть не должно).

**Вставка данных**

Добавить данные в таблицу можно командой INSERT:

```
INSERT INTO students (name, age, email)
VALUES ('Иван Иванов', 20, 'ivan@example.com');
```

Можно вставить сразу несколько строк:

```
INSERT INTO students (name, age, email) VALUES
('Мария Петрова', 22, 'maria@example.com'),
('Сергей Смирнов', 19, 'sergey@example.com');
```

**Выборка данных (SELECT)**

Команда SELECT используется для извлечения данных.

Пример:

```
SELECT * FROM students;
```

Здесь * означает «выбрать все столбцы».

Выберем только имя и возраст:

```
SELECT name, age FROM students;
```

**Фильтрация данных (WHERE)**

Фильтрация осуществляется условием WHERE.

Пример: выбрать студентов старше 20 лет:

```
SELECT name, age FROM students
WHERE age > 20;
```

Выбрать студентов по email:

```
SELECT * FROM students
WHERE email = 'ivan@example.com';
```

**Сортировка (ORDER BY)**

Чтобы упорядочить данные, используют ORDER BY.

Пример:

```
SELECT name, age FROM students
ORDER BY age ASC;  -- по возрастанию

SELECT name, age FROM students
ORDER BY age DESC; -- по убыванию
```

**Ограничение количества (LIMIT)**

Иногда нужно взять только несколько строк:

```
SELECT * FROM students
LIMIT 2;
```

**Обновление данных (UPDATE)**

Команда UPDATE изменяет данные.

Пример: изменим возраст студента Иванова:

```
UPDATE students
SET age = 21
WHERE name = 'Иван Иванов';
```

⚠ Важно: без `WHERE` изменятся все строки!

**Удаление данных (DELETE)**

Удалим студента по email:

```
DELETE FROM students
WHERE email = 'sergey@example.com';
```

Удалить все строки (остается пустая таблица):

```
DELETE FROM students;
```

**Объединение условий**

Условия можно комбинировать:

```sql
SELECT * FROM students
WHERE age > 18 AND email LIKE '%example.com';
```

AND – логическое «и».

OR – логическое «или».

LIKE – поиск по шаблону (% — любое количество символов).

**Агрегатные функции**

SQL позволяет выполнять подсчёты:

```sql
SELECT COUNT(*) FROM students;  -- количество студентов
SELECT AVG(age) FROM students;  -- средний возраст
SELECT MAX(age) FROM students;  -- максимальный возраст
```

**Группировка (GROUP BY)**

Группировка используется для статистики. Например, посчитать, сколько студентов каждого возраста:

```sql
SELECT age, COUNT(*) FROM students
GROUP BY age;
```