<TeachMeSkills/>

курс

Android разработчик

Базы данных

Агенда занятия:

Реляционные и нереляционные базы данных

Базовый SQL-синтаксис

Связи между таблицами

Миграции

- **Реляционные и нереляционные базы данных**
  - Базовый SQL-синтаксис
  - Связи между таблицами
  - Миграции

База данных

База данных — это организованная структура для хранения, управления и поиска данных.
Она позволяет быстро и надёжно сохранять большие объемы информации, работать с ней (добавлять, удалять, изменять, искать) и обеспечивать целостность и безопасность данных.

База данных

Зачем нужны базы данных?

- Хранение больших объемов информации;
- Быстрый поиск и выборка данных;
- Гарантия целостности данных (например, нельзя потратить несуществующие деньги);
- Возможность одновременной работы для многих пользователей;
- Безопасность и резервное копирование.

Реляционные и нереляционные СУБД

Реляционные БД (SQL) хранят данные в таблицах – в строках и столбцах. Каждая таблица имеет строго заданную схему, а связи между данными реализуются через внешние ключи. Это обеспечивает целостность данных (ACID), но снижает гибкость изменений структуры.

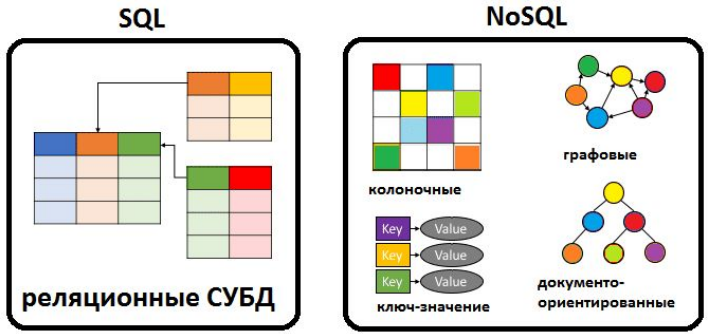
Примеры: PostgreSQL, MySQL, SQLite.

Реляционные и нереляционные СУБД

Нереляционные БД (NoSQL) используют другие модели хранения: документы (JSON-подобные), ключ-значение, графы, столбцовые хранилища и т.д. Схема может быть свободной – разные записи могут иметь разную структуру. Это дает гибкость и масштабируемость (горизонтальный рост), но ослабляет строгую согласованность. Популярные примеры: MongoDB (документная), Redis (ключ-значение, кеш), Firebase Realtime DB (документно-ориентированная JSON БД).

Реляционные и нереляционные СУБД

Пример: в SQL вы бы создали таблицу «Users» с колонками (id, name, email), а в NoSQL сохранили JSON-документ { _id: 1, name: "Иван", preferences: {theme: "dark"} }. - Безопасность и резервное копирование



Реляционные и нереляционные базы данных

▶ **Базовый SQL-синтаксис**

Связи между таблицами

Миграции

Базовый SQL-синтаксис

- Создание базы и таблиц/коллекций;
- Добавление данных (INSERT/CREATE);
- Изменение данных (UPDATE);
- Удаление данных (DELETE);
- Поиск/запрос данных (SELECT/READ).

Создание базы и таблиц/коллекций

SQLite — это встраиваемая СУБД, где вся база хранится в одном файле.

Файл базы можно создать разными способами:

- Через командную строку: **sqlite3 example.db**. Если такого файла нет, SQLite автоматически его создаст
- В интерактивной оболочке sqlite3: команда **open example.db** — также откроет или создаст файл базы

Создание таблиц

Для создания таблицы используется команда CREATE TABLE.

Пример создания таблицы:

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

Создание таблиц

id INTEGER PRIMARY KEY AUTOINCREMENT — целочисленный первичный ключ.
Если при вставке не указывать id, SQLite присвоит этому полю следующее число.

Ключевое слово AUTOINCREMENT гарантирует, что идентификаторы не будут повторно использоваться после удаления строк
name TEXT NOT NULL — текстовое поле (строка), не допускает NULL.
age INTEGER — целочисленное поле.

Основные типы хранения данных в SQLite

INTEGER, REAL, TEXT, BLOB и NULL.

SQLite использует динамическую типизацию: фактический тип значения определяется само́ значением, а не строго типом столбца.

Например, в любой столбец (кроме INTEGER PRIMARY KEY) можно записать значение любого типа, и SQLite корректно его сохранит.

SQLite поддерживает ограничения NOT NULL, UNIQUE, CHECK, PRIMARY KEY, FOREIGN KEY.

Например, UNIQUE гарантирует уникальность значений в столбце, а FOREIGN KEY(user_id) REFERENCES Users(id) задаёт внешний ключ на таблицу Users.

Добавление данных (INSERT/CREATE)

INSERT – добавляет новую строку в таблицу.

Синтаксис: INSERT INTO таблица (col1, col2) VALUES (val1, val2);.

Пример:

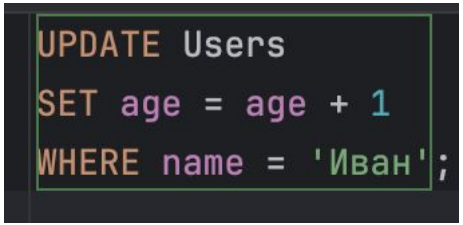
```
INSERT INTO Users (name, age)
VALUES ( name 'Иван', age 20);
```

Изменение данных (UPDATE)

UPDATE – изменяет существующие записи.

Синтаксис: UPDATE таблица SET col1 = val1, col2 = val2 WHERE условие;

Пример:



Это повысит возраст «Ивану» на 1. Без WHERE обновятся все строки!

Удаление данных (DELETE)

DELETE – удаляет записи из таблицы.

Синтаксис: DELETE FROM таблица WHERE условие;

Пример:

```
DELETE FROM Users
WHERE age < 18;
```

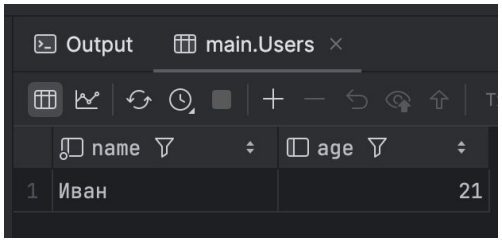
Поиск/запрос данных (SELECT/READ)

SELECT – выбирает данные из таблицы.

Синтаксис: SELECT столбец1, столбец2 FROM таблица WHERE условие;

Пример:

```
SELECT name, age
FROM Users
WHERE age >= 18;
```

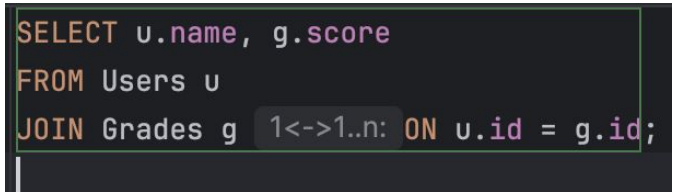


Поиск/запрос данных (SELECT/READ)

JOIN – объединяет данные из нескольких таблиц по общему ключу. Обычно пишут FROM A JOIN B ON A.key = B.key.

Например, есть таблицы Students(id,name) и Grades(student_id,score).

Запрос выведет оценки каждого студента. INNER JOIN выбирает только совпадающие записи из обеих таблиц



Поиск/запрос данных (SELECT/READ)

Смотрим БД

Реляционные и нереляционные базы данных

Базовый SQL-синтаксис

▶ **Связи между таблицами**

Миграции

Один к одному (1:1)

Один к одному (1:1): каждая запись в таблице связана ровно с одной записью в другой и наоборот.
Например, «Страна» ↔ «Столица». Каждая страна имеет одну столицу, и каждая столица принадлежит только одной стране. В БД это часто делается дополнительной таблицей с уникальным внешним ключом или просто разбиением данных.

Пример: таблица Person(id, name) и таблица Passport(id, person_id unique, serial). Уникальный ключ person_id гарантирует, что одному человеку соответствует только один паспорт

Один ко многим (1:N)

Один ко многим (1:N): одна запись «родительской» таблицы соответствует многим записям «дочерней».

Например, «Мать» – «Дети»: у одной матери может быть много детей, но у каждого ребёнка только одна мать. Это самый распространённый тип. В реляционной БД «многие» таблица содержит внешний ключ на «одну» таблицу.

«Пользователь может иметь много телефонов (Phone), а телефон принадлежит только одному пользователю (Person)». В примере таблица Phone имеет поле PersonId (FK), так что несколько телефонов ссылаются на одного человека.

Пример: таблицы Author(id,name) и Book(id,title,author_id), где author_id – внешний ключ. Один автор может написать много книг.

Многие ко многим (M:N)

Многие ко многим (M:N): каждая запись первой таблицы может быть связана со многими записями второй таблицы, и наоборот.

Пример: «Ученики» ↔ «Курсы»: один студент может посещать многие курсы, и один курс посещает много студентов. Непосредственно такое отношение не создаётся; вместо этого вводится связующая таблица (junction table) с двумя внешними ключами.

В примере «Сотрудники» и «Должности»: одному сотруднику может соответствовать несколько должностей, каждой должности – несколько сотрудников. Создается дополнительная таблица Employee_Position(employee_id, position_id) – каждая строка означает, что определенному сотруднику назначена определенная должность.

Реляционные и нереляционные базы данных

Базовый SQL-синтаксис

Связи между таблицами

- **Миграции**

Миграция

Миграция – это версия схемы БД: способ фиксировать изменения структуры (таблиц, столбцов) в файлах и автоматически применять их на разных окружениях.

По сути, миграции – это «система контроля версий» для схемы.

Миграция

Зачем нужны миграции: по мере разработки проекта появляются новые таблицы и колонки – миграции помогают отслеживать эти изменения и синхронизировать схему между средами разработки, тестирования и продакшн.

В Agile-разработке схемы эволюционируют от спринта к спринту, и миграции позволяют «рефакторить» БД так же, как код. Также миграции хранятся в репозитории – их изменения проходят код-ревью перед применением

Практика

Задача 1.

Создайте базу и 3 таблицы:
1. Первая таблица – Пользователи: имя, почта, возраст;
2. Вторая таблица – Товары: наименование, цена, количество на складе;
3. Третья таблица – Корзина: товар, пользователь, количество.

Задача 2.

Наполните таблицы данными.
Добавьте по 3 пользователя, товара и записи в корзину.

Задача 3.

Удалите всех пользователей младше 18.

Задача 4.

Измените количество оставшихся товаров на 0 для 1 товара.

Задача 5.

Измените количество оставшихся товаров на 0 для 1 товара.

Домашнее задание

Задача 1.

Спроектируйте базу данных для задания FakeStore из урока Network часть 2.

Создайте все необходимые таблицы и запросы в базу для добавления в корзину, удаления из корзины и обновления товара.

Функции:

- Каталог товаров (GET)
- Экран детали товара (GET /{id})
- Добавление в корзину (POST)
- Удаление из корзины (DELETE)
- Редактирование товара (PUT)

Q&A

Ваши вопросы

Спасибо

<TeachMeSkills/>