Лекция №12
«База данных SQLite»

Москва 2019

AlertDialog

Используется класс AlertDialog.Builder. Мы указываем заголовок, текст сообщения, иконку и кнопки. Диалог может содержать максимум три кнопки ответа: положительная, отрицательная и нейтральная. Для каждой указываем текст и обработчик. Метод create создает диалог и мы его возвращаем (return).

Обработчик кнопок myClickListener реализует интерфейс DialogInterface.OnClickListener и в нашем случае является общим для всех кнопок. В нем мы проверяем, какая кнопка была нажата:
если положительная (BUTTON_POSITIVE), то сохраняем данные и закрываем приложение
если отрицательная (BUTTON_NEGATIVE), то закрываем приложение без сохранения
если нейтральная (BUTTON_NEUTRAL), то не делаем ничего

Файлы

Разработать программу, которая будет оценивать 20 учеников по 4 предметам . После выставления оценок, программа будет выводить их на экран, а затем выполнять операцию записи оценок в файл.
В программе будут две активности, первая основная отображает список учеников и их оценки, во второй – есть возможность редактирования и сохранения.
Создадим:
а) внутренний файл для записи и чтения
б) внутренний файл только для чтения

База данных SQLite

Минимальные затраты ресурсов.
Для работы большинства систем управления базами данных необходим специальный процесс сервера базы данных. SQLite обходится без сервера; база данных SQLite представляет собой обычный файл. Когда база данных не используется, она не расходует процессорное время. Это особенно важно на мобильных устройствах, чтобы избежать разрядки аккумулятора.

Оптимизация для одного пользователя.
С базой данных взаимодействует только наше приложение, поэтому можно обойтись без идентификации с именем пользователя и паролем.

Надежность и быстрота.
Базы данных SQLite невероятно надежны. Они поддерживают транзакции баз данных (другими словами, если при обновлении нескольких блоков данных что-то пойдет не так, SQLite сможет вернуться к исходному состоянию). Кроме того, операции чтения и записи данных реализуются на оптимизированном коде C. Этот код не только быстро работает, но и сокращает объем необходимых вычислительных ресурсов.

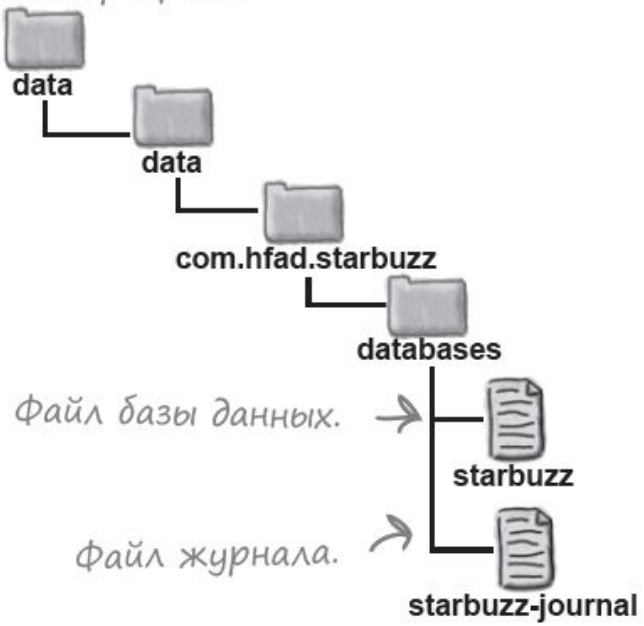
База данных SQLite

Где хранится база данных?

Android автоматически создает для каждого приложения папку, в которой хранятся базы данных этого приложения. Когда мы создаем базу данных для приложения Starbuzz, она будет храниться в следующей папке:

/data/data/com.hfad.starbuzz/databases

com.hfad.starbuzz — уникальный идентификатор приложения.



В этой папке приложение может хранить несколько баз данных. Каждая база данных состоит из двух файлов.

Имя первого — **файла базы данных** — соответствует имени базы данных: например, “starbuzz”. Это основной файл баз данных SQLite; в нем хранятся все данные.

Второй файл — **файл журнала**. Его имя состоит из имени базы данных и суффикса “-journal” — например, “starbuzz-journal”. В файле журнала хранится информация обо всех изменениях, внесенных в базу данных. Если в работе с данными возникнет проблема, Android использует данные журнала для отмены (или отката) последних изменений.

Типы данных базы данных

Тип	Описание
NULL	пустое значение
INTEGER	целочисленное значение
REAL	значение с плавающей точкой
TEXT	строки или символы в кодировке UTF-8, UTF-16BE или UTF-16LE
NUMERIC	здесь можно хранить булевы значения, а также время и дату
BLOB	бинарные данные

База данных SQLite

Помощник SQLite

Помощник SQLite создается расширением класса `SQLiteOpenHelper`. Он предоставляет средства для создания и управления базами данных.

Класс базы данных SQLite

Класс `SQLiteDatabase` предоставляет доступ к базе данных. Его можно сравнить с классом `SQLConnection` в JDBC.

Курсор

Класс `Curs`
чен для чте
в базу данн
но сравнит
`ResultSet` в JDBC.

Пример работы с базой данных

4 активности приложения:
В 1-й активности пользователь выбирает хочет ли пройти регистрацию или идентификацию
Во 2-й активности он получает возможность зарегистрироваться
В 3-й активности он получает возможность идентифицироваться
В 4-ю он переходит из второй или третьей после успешного завершения действия в ней.

Класс Cursor

Класс **Cursor** содержит немало возможностей для навигации (но не ограничивается только ими):

- moveToFirst() — перемещает курсор на первую строку в результате запроса;
- moveToNext() — перемещает курсор на следующую строку;
- moveToLast() - перемещает курсор на последнюю строку;
- moveToPrevious() — перемещает курсор на предыдущую строку;
- getCount() — возвращает количество строк в результирующем наборе данных;
- getColumnIndexOrThrow() — возвращает индекс для столбца с указанным именем (выбрасывает исключение, если столбец с таким именем не существует);
- getColumnName() — возвращает имя столбца с указанным индексом;
- getColumnNames() — возвращает массив строк, содержащий имена всех столбцов в объекте Cursor;
- moveToPosition() — перемещает курсор на указанную строку;
- getPosition() — возвращает текущую позицию курсора

Также Android предоставляет следующие методы:

- isBeforeFirst()
- isAfterLast() - полезный метод, сигнализирующий о достижении конца запроса. Используется в циклах
- isClosed()

Базы данных

Класс ContentValues

Класс **ContentValues** используется для добавления новых строк в таблицу. Каждый объект этого класса представляет собой одну строку таблицы и выглядит как ассоциативный массив с именами столбцов и значениями, которые им соответствуют.

Курсоры

В Android запросы к базе данных возвращают объекты класса Cursor. Вместо того чтобы извлекать данные и возвращать копию значений, курсоры ссылаются на результирующий набор исходных данных. Курсоры позволяют управлять текущей позицией (строкой) в результирующем наборе данных, возвращаемом при запросе.

Базы данных

Класс SQLiteOpenHelper: Создание базы данных

Библиотека Android содержит абстрактный класс **SQLiteOpenHelper**, с помощью которого можно создавать, открывать и обновлять базы данных. Это основной класс, с которым вам придётся работать в своих проектах. При реализации этого вспомогательного класса от вас скрывается логика, на основе которой принимается решение о создании или обновлении базы данных перед ее открытием. Класс **SQLiteOpenHelper** содержит два обязательных абстрактных метода:

- **onCreate()** — вызывается при первом создании базы данных
- **onUpgrade()** — вызывается при модификации базы данных

Также используются другие методы класса:

- **onDowngrade(SQLiteDatabase, int, int)**
- **onOpen(SQLiteDatabase)**
- **getReadableDatabase()**
- **getWritableDatabase()**

Базы данных

В приложении необходимо создать собственный класс, наследуемый от **SQLiteOpenHelper**. В этом классе необходимо реализовать указанные обязательные методы, описав в них логику создания и модификации вашей базы.

В этом же классе принято объявлять открытые строковые константы для названия таблиц и полей создаваемой базы данных, которые клиенты могут использовать для определения столбцов при выполнении запросов к базе данных. Например, так можно объявить константы для таблицы **CONTACT**:

```
public static final String TABLE_NAME = "CONTACT";
public static final String NAME = "FIRST_NAME";
public static final String PHONE = "PHONE";
```

Лучше, если вы будете давать сразу понятные имена, указывающие на работу с таблицей. Если имя переменной **TABLE_NAME** ещё можно оставить, то для других лучше использовать более говорящие имена, например, **KEY_NAME** и **KEY_PHONE** или **COLUMN_NAME**, **COLUMN_PHONE**.

Базы данных

Ваш класс, расширяющий **SQLiteOpenHelper**, также неявно наследует интерфейс **BaseColumns**, в котором определена строковая константа **_ID**, представляющая имя поля для идентификаторов записей. В создаваемых таблицах базы данных поле **_ID** должно иметь тип **INTEGER PRIMARY KEY AUTOINCREMENT**. Описатель **AUTOINCREMENT** является необязательным. Часто в других примерах идентификатор создаётся вручную, смотрите как вам удобнее. Только всегда называйте его именно **_id**. Такое название используется в Android для работы с курсорами и поэтому придерживайтесь данного правила.

В методе **onCreate()** необходимо реализовать логику создания таблиц и при необходимости заполнить их начальными данными при помощи SQL-команды, например:

```
@Override
public void onCreate(SQLiteDatabase db)
{
    db.execSQL("CREATE TABLE + TABLE_NAME
        (_id INTEGER PRIMARY KEY AUTOINCREMENT,
        COLUMN_NAME TEXT,
        COLUMN_PHONE TEXT);");
}
```

Базы данных

В методе **onUpgrade()** можно, например, реализовать запрос в базу данных на уничтожение таблицы (DROP TABLE), после чего вновь вызвать метод **onCreate()** для создания версии таблицы с обновленной структурой, например, так:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
{
    db.execSQL("DROP TABLE IF EXISTS + TABLE_NAME);
    onCreate(db);
}
```

В параметрах метода используются номера версий базы данных - старая и новая. О номере версии говорилось выше.

Обновляется структура базы данных следующим образом. Сначала меняем порядковый номер.

```
private static final int DATABASE_VERSION = 2;
```

Базы данных

Обновляется структура базы данных следующим образом. Сначала меняем порядковый номер.

```
private static final int DATABASE_VERSION = 2;
```

При первой установке приложения базы данных ещё не существует. Тут проверять пока нечего. При установке новой версии приложения система проверит номер версии базы данных. Если он больше, чем было, то вызовется метод **onUpgrade()**. Если наоборот, то вызовется метод **onDowngrade()** (обычно так происходит редко).

Как использовать? Просто ставим условие на проверку.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    if (oldVersion == 1) {
        // Код для работы с базой данных с версией 1
    }
    if (oldVersion < 3) {
        // Код для работы с базой данных с версией 1 или 2
    }
```

Базы данных

SQLiteDatabase

Для управления базой данных SQLite существует класс **SQLiteDatabase**. В классе **SQLiteDatabase** определены методы **query()**, **insert()**, **delete()** и **update()** для чтения, добавления, удаления, изменения данных. Кроме того, метод **execSQL()** позволяет выполнять любой допустимый код на языке SQL применимо к таблицам базы данных, если вы хотите провести эти (или любые другие) операции вручную.

Каждый раз, когда вы редактируете очередное значение из базы данных, нужно вызывать метод **refreshQuery()** для всех курсоров, которые имеют отношение к редактируемой таблице.

Для составления запроса используются два метода: **rawQuery()** и **query()**, а также класс **SQLiteQueryBuilder**.

Базы данных

Для составления запроса используются два метода: **rawQuery()** и **query()**, а также класс **SQLiteQueryBuilder**.

Метод query()

Для чтения данных используют вызов метода **query()**:

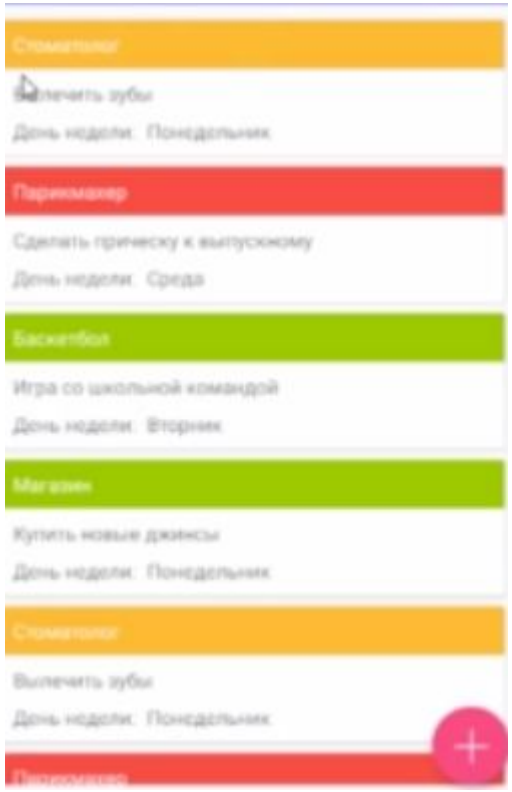
```
Cursor query (String table,
        String[] columns,
        String selection,
        String[] selectionArgs,
        String groupBy,
        String having,
        String sortOrder)
```

В метод **query()** передают семь параметров. Если какой-то параметр для запроса вас не интересует, то оставляете *null*:

- **table** — имя таблицы, к которой передается запрос;
- **String[] columnNames** — список имен возвращаемых полей (массив). При передаче *null* возвращаются все

RecyclerView, CardView, ViewHolder
Приложение «Заметки»

Заголовок заметки
Текст, что нужно сделать и день недели, когда сделать.
Если приоритет высокий, то красный цвет.
Если приоритет низкий, то цвет зеленый.
Также есть возможность добавить заметки.



Внутренние файлы

-Доступ возможен из той аппликации для которой и в которой они созданы

-Они хранятся в папке связанной с пакетом самой аппликации

Внутренние файлы только для чтения

Они находятся в папке res/raw. Они закрыты для записи.

Являются частью аппликации.

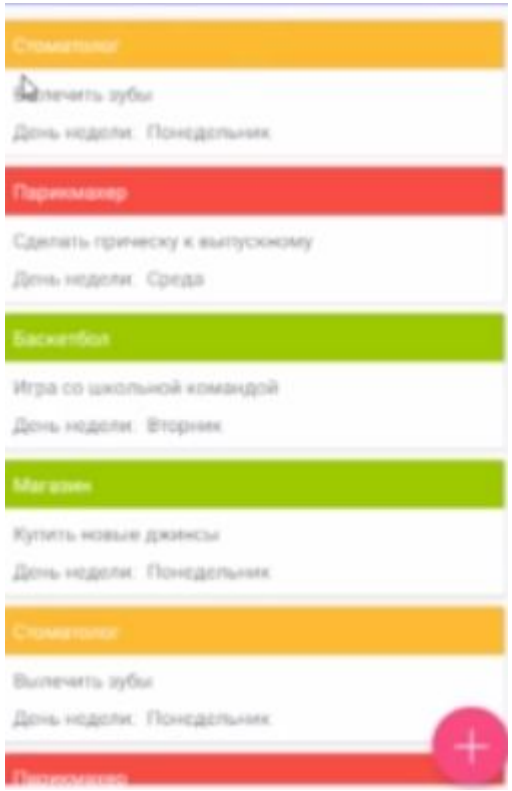
Внешние открытые файлы на карте памяти

Они доступны всем и всегда. Открыты для чтения и записи.

Внешние файлы вне аппарата

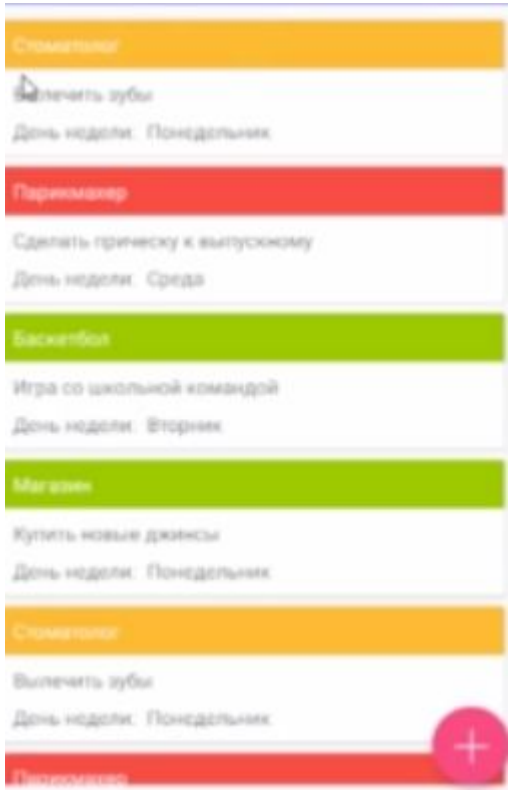
RecyclerView, CardView, ViewHolder
Приложение «Заметки»

Создаем макет, который будет содержать одну заметку
Добавляем RecycleView и размещаем его по центру
В RecyclreView выводим список всех заметок. Чтобы соединить эти два элеиента надо создать адаптер.
В памяти будут храниться только те элементы, которые видны на экране и пара сверху и снизу и будут подгружаться при проктрутке. Иначе придеться вызывать findViewById при добавлении нового элемента, что медленно



RecyclerView, CardView, ViewHolder
Приложение «Заметки»

При прокрутке получаем уже созданный элемент. В этом методе макет который создавали и передать в качестве аргумента



Постоянное хранилище Android

SharedPreferences — постоянное хранилище на платформе Android, используемое приложениями для хранения своих настроек, например. Это хранилище является относительно постоянным, пользователь может зайти в настройки приложения и очистить данные приложения, тем самым очистив все данные в хранилище.

Для работы с данными постоянного хранилища нам понадобится экземпляр класса SharedPreferences, который можно получить у любого объекта, унаследованного от класса android.content.Context (например, Activity или Service).