SQLite

http://www.sqlite.org.

Содержание

- SQLite3
- Компонент SQL database engine
- Чего конкретно в поддержке SQL нет
- Что необычного?
- Тип столбца
- А как с архитектурой? Сервера-нет?
- SQLite и Android
- Подписываем контракт
  (интеграция базы данных в приложение)
- BaseColumn
- SQLiteOpenHelper
- Работаем с записями базы данных

SQLite3

- Платформа Android имеет встроенный инструментарий для управления базой данных sqlite3. SQLite - это замечательный продукт, который успел завоевать признание во всем мире и получить множество наград
- iOS (iPhone, iPod, iPad) также используют базы данных SQLite.
- Microsoft присоединилась к данному решению и телефоны Windows Phone 8 также работают с SQLite.
- SQLite - это проект с открытыми исходными кодами, поддерживающий стандартные возможности обычной SQL: синтаксис, транзакции и др. Занимает очень мало места - около 250 кб

Компонент SQL database engine

. SQLite является наиболее часто используемым ядром базы данных в мире.

Формат файла SQLite является стабильным, кросс-платформенным и обратно совместимым, и разработчики обещают сохранить его таким образом, по крайней мере, до 2050 года.

Исходный код SQLite находится в свободном доступе и может быть использован любым пользователем для любых целей

Чего конкретно в поддержке SQL нет

Нельзя удалить или изменить столбец в таблице (ALTER TABLE DROP COLUMN…, ALTER TABLE ALTER COLUMN…).

Есть триггеры, но не настолько мощные как у крупных RDBMS.

Есть поддержка foreign key, но по умолчанию — она ОТКЛЮЧЕНА.

Нет встроенной поддержки UNICODE (но ее нетрудно добиться).

Нет хранимых процедур.

Что необычного?

a) каждая запись содержит виртуальный столбец rowid, который равен 64-битному номеру (уникальному для таблицы).

Можно объявить свой столбец INTEGER PRIMARY KEY и тогда этот столбец станет rowid (со своим именем, имя rowid все равно работает).

При вставке записи можно указать rowid, а можно — не указывать (и система тогда вставит уникальный).

b) можно без труда организовать БД

c) легко переносить: по умолчанию, БД — это один файл (в кроссплатформенном формате);

d) тип столбца не определяет тип хранимого значения в этом поле записи, то есть в любой столбец можно занести любое значение;

e) много встроенных функций (которые можно использовать в SQL.

Тип столбца

Тип столбца определяет как сравнивать значения (нужно их привести к единому типу при сравнении, например, внутри индекса).

Но это не обязывает заносить значения именно такого типа в столбец (weak typing).

Например, объявили столбец как «INTEGER». SQlite позволяет занести в этот столбец значения любого типа (999, «abc», «123», 678.525).

Если вставляемое значение — не целое, то SQlite пытается привести его к целому.

Т.е. строка «123» превратится в целое 123, а остальные значения запишутся «как есть».

Так можно вообще не задавать тип столбца?

Очень часто так и делается: *CREATE TABLE foo (a,b,c,d)*.

Динамические типы данных

SQLite поддерживает динамическое типизирование данных.

Возможные типы значений: INTEGER, REAL, TEXT и BLOB. Так же поддерживается специальное значение NULL.

Размеры значений типа TEXT и BLOB не ограничены ничем, кроме константы SQLITE_MAX_LENGTH в исходниках sqlite, равной 10^9.

Каждое значение в любом поле любой записи может быть любого из этих типов, независимо от типа, указанного при объявлении полей таблицы.

Указанный при объявлении поля тип хранится для справки в его исходном написании, и используется в качестве основы для выбора предпочтений (так называемое «type affinity»: это подход, редко встречающийся в других СУБД) при выполнении неявных преобразований типов на основании похожести этого названия типа на что-либо, знакомое SQLite.

Если безопасного преобразования записываемого значения в предпочитаемый тип не получается, SQLite записывает значение в его исходном виде.

Для получения значений из базы есть ряд функций для каждого из типов, и если тип хранимого значения не соответствует запрашиваемому, оно тоже, по возможности, преобразуется.

А как с архитектурой? Сервера-нет?

Сервера нет, само приложение является сервером.

Доступ к БД происходит через «подключения» к БД (нечто вроде хэндла файла ОС), которые открываем через вызов функции DLL.

При открытии указывается имя файла БД.

Если такого нету — он автоматически создается.

Допустимо открывать множество подключений к одной и тоже БД (через имя файла) в одном или разных приложениях.

Система использует механизмы блокировки доступа к файлу на уровне ОС, чтобы это все работало (эти механизмы обычно плохо работают на сетевых дисках, так что не рекомендуется использовать SQlite с файлом на сети).

Изначально SQlite работал по принципу «многие читают — один пишет». То есть только одно соединение пишет в БД в данный момент времени.

Если другие соединения попробуют тоже записать, то будет ошибка SQLITE_BUSY.
Можно, однако, ввести таймаут операций.

Тогда подключение, столкнувшись с занятостью БД, будет ждать N секунд прежде, чем отвалиться с ошибкой SQLITE_BUSY.

Как быть с ошибкой SQLITE_BUSY?

Либо одно подключение и все запросы через него, либо исходить из возможного таймаута и предусмотреть повтор выполнения SQL.

Есть и еще одна возможность: не так давно появился новый вид лога SQlite:

Write Ahead Log, WAL.

Если включить для БД именно этот режим лога, то несколько подключений смогут одновременно модифицировать БД.

Но в этом режиме БД уже занимает несколько файлов.

НЕТ ГЛОБАЛЬНОГО КЭША

Действительно, все современные RDBMS немыслимы без глобального разделяемого кэша, который может хранить что-то вроде скомпилированных параметризованных запросов.

Этим занят сервер, которого тут нет.

Однако, в рамках одного приложения SQlite может разделять кэш между несколькими подключениями и немного сэкономить память.

SQLite — тормозит?

Две причины.

Первая — настройки по умолчанию.

Они работают на надежность, а не на производительность.

Вторая — непонимание механизма фиксации транзакций.

По умолчанию после любой команды SQlite будет фиксировать транзакцию (то есть ожидать пока БД окажется в целостном состоянии для отключения питания).

В зависимости от режима SQLite потратит на это от 50 до 300 мс (ожидая окончания записи данных на диск).

Нужно вставить 100 тыс записей и быстро!

- Удалить индексы, включить режим синхронизации OFF (или NORMAL), вставлять порциями по N тысяч (N — подобрать, для начала взять 5000).
- Перед вставкой порции сделать BEGIN TRANSACTION, после — COMMIT.

SQLite и Android

С помощью SQLite вы можете создавать для своего приложения независимые реляционные базы данных.

Android хранит базы данных в каталоге **/data/data/<имя_вашего_пакета>/databases** на эмуляторе, на устройстве путь может отличаться.

По умолчанию все базы данных закрытые, доступ к ним могут получить только те приложения, которые их создали.

Каждая база данных состоит из двух файлов.

Имя первого файла базы данных соответствует имени базы данных.

Это основной файл баз данных SQLite, в нём хранятся все данные. Вы будете создавать его программно.

Второй файл — файл журнала. Его имя состоит из имени базы данных и суффикса "-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. Вы с ним не будете взаимодействовать, но если вы будете просматривать внутренности своего устройства, то будете знать, зачем этот файл там присутствует.

Подписываем контракт (интеграция базы данных в приложение)

При работе с базой данных принято создавать новый пакет **data** внутри основного пакета. Щёлкаем правой кнопкой мыши по имени пакета, выбираем **New | Package** и вводим новое имя.

В последних рекомендациях Гугла **рекомендуется создавать класс-контракт.**

Будем придерживаться этого правила. Мы как бы подписываем контракт на работу с базой данных и предоставляем все нужные данные.

Внутри созданного пакета создаём новый класс **HotelContract**. Класс-контракт является контейнером для базы данных и может содержать несколько внутренних классов, которые представляют отдельные таблицы.

Внутри класса создаём внутренний класс. В нашем случае будет один класс для таблицы **guests**.

Зададим схему таблицы и константы для столбцов для удобства.

Класс будет выглядеть так

```
import android.provider.BaseColumns;
public final class HotelContract { private HotelContract() { };
public static final class GuestEntry implements BaseColumns {
public final static String TABLE_NAME = "guests";
 public final static String _ID = BaseColumns._ID;
public final static String COLUMN_NAME = "name";
 public final static String COLUMN_CITY = "city";
public final static String COLUMN_GENDER = "gender";
 public final static String COLUMN_AGE = "age";
public static final int GENDER_FEMALE = 0;
public static final int GENDER_MALE = 1;
public static final int GENDER_UNKNOWN = 2; } }
```

BaseColumn

В классе используется реализация интерфейса **BaseColumn**:

public static final class GuestEntry implements **implements BaseColumns** {

Что это даёт? В большинстве случаев работа с базой данных происходит через специальные объекты **Cursor**, которые требуют наличия в таблице колонки с именем **_id**.

Вы можете создать столбец вручную в коде, а можно положиться на **BaseColumn**, который создаст столбец с нужным именем автоматически.

Если вы не будете работать с курсорами, то можете использовать и стандартное наименование **id** или вообще не использовать данный столбец, но не советуют так поступать.

Использование класса HotelContract

После создания класса мы можем изменить код в **EditorActivity** в том месте, где происходит выбор пола гостя через выпадающий список.

```
if (selection.equals(getString(R.string.gender_female))) {
    mGender =
    HotelContract.GuestEntry.GENDER_FEMALE; // Кошка
    } else if
    (selection.equals(getString(R.string.gender_male))) {
    mGender = HotelContract.GuestEntry.GENDER_MALE;
    // Кот } else { mGender =
    HotelContract.GuestEntry.GENDER_UNKNOWN; // Не
    определен }
```

SQLiteOpenHelper

Следующий шаг - создание класса в пакете **data**, который наследуется от специального класса **SQLiteOpenHelper** и непосредственно работает с базой данных.

В классе создаются константы для удобной работы. Также реализуются методы **onCreate()** и **onUpgrade()**.

Созданный класс будет работать с базой данных - добавлять, выбирать, удалять записи и прочие операции.

Схема нашей таблицы.

CREATE TABLE guests(_id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, city TEXT NOT NULL, gender INTEGER NOT NULL DEFAULT 3, age INTEGER NOT NULL DEFAULT 0);

Наследуемся от **SQLiteOpenHelper**.

Щёлкаем правой кнопкой мыши на имени пакета в левой части студии и выбираем в меню **New** | **Java Class** и в диалоговом окне выбираем имя для нового класса, например, **HotelDbHelper**.

Слово **Helper** обычно используют, чтобы показать, что класс является обёрткой (вспомогательным классом) какого-то абстрактного класса.

Студия предложит создать два обязательных метода **onCreate()** и **onUpgrade().**

Класс **HotelDbHelper**.

```
import android.content.Context;
 import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
 public class HotelDbHelper extends SQLiteOpenHelper {
public static final String LOG_TAG =
    HotelDbHelper.class.getSimpleName();
 /** * Имя файла базы данных */
 private static final String DATABASE_NAME = "hotel.db";
 /** * Версия базы данных. При изменении схемы увеличить
    на единицу */
private static final int DATABASE_VERSION = 1;
 /** * Конструктор {@link HotelDbHelper}. * * @param context
    Контекст приложения */
public HotelDbHelper(Context context) {
 super(context, DATABASE_NAME, null, DATABASE_VERSION); }
```

Создание базы данных

```
/** * Вызывается при создании базы данных */
@Override
 public void onCreate(SQLiteDatabase db) {
 // Строка для создания таблицы
 String SQL_CREATE_GUESTS_TABLE = "CREATE TABLE " + GuestEntry.TABLE_NAME + " (“
 + HotelContract.GuestEntry._ID
 + " INTEGER PRIMARY KEY AUTOINCREMENT, "
+ GuestEntry.COLUMN_NAME
+ " TEXT NOT NULL, «
 + GuestEntry.COLUMN_CITY
+ " TEXT NOT NULL, "
+ GuestEntry.COLUMN_GENDER
 + " INTEGER NOT NULL DEFAULT 3, «
 + GuestEntry.COLUMN_AGE
 + " INTEGER NOT NULL DEFAULT 0);";
 // Запускаем создание таблицы
 db.execSQL(SQL_CREATE_GUESTS_TABLE); }
 /** * Вызывается при обновлении схемы базы данных */
 @Override
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) { } }
```

DATABASE_VERSION

Третий параметр **null** в суперклассе используется для работы с курсорами. Сейчас их не используем, поэтому оставим в покое.

Константа **DATABASE_NAME** отвечает за имя файла, в котором будет храниться база данных приложения.

Вторая константа в конструкторе - **DATABASE_VERSION** требует дополнительных объяснений.

Она отвечает за номер версии базы.

Принцип её работы схож с номером версий самого приложения.

Когда мы видим, что вышла новая версия то понимаем, что пора обновляться.

Аналогично поступает и само приложение, когда замечает, что номер версии базы стал другим.

Как только программа заметила обновление номера базы, она запускает метод **onUpgrade()**, который у нас сформировался автоматически.

В этом методе необходимо разместить код, который должен

Метод **onCreate()**

Метод **onCreate()** - здесь создаётся сама база данных с необходимыми данными для работы.

Метод вызывается, если в устройстве нет базы данных и наш класс должен создать его.

У метода есть параметр **db**, который относится к классу **SQLiteDatabase**.

У класса есть специальный метод **execSQL()**, которому нужно передать запрос (SQL-скрипт) для создания таблицы.

Для создания таблицы в SQL используется команда **CREATE TABLE ...**.

Для удобства вынесем команду в отдельную строку.

Аналогично поступим с командой **DROP TABLE**.

Так как строка очень длинная и состоит из множества строковых переменных, которые нужно соединить в одну цепочку, то поступают следующим образом.

Создаём ещё одну строковую константу для формирования скрипта и передадим её в метод.

Ситуация с использованием **onUpgrade()**

Новые пользователи, которые установят программу первый раз, радуются жизни - у них есть все необходимые данные для работы.

Но что делать тем, кто уже работает со старой программой?

Обновившись, они увидят дополнительное текстовое поле для ввода даты рождения, но в старой базе нет колонки для хранения новых данных.

И ваша программа завершится с ошибкой.

Полностью удалять и устанавливать новую версию программы тоже не выход - тогда пропадут старые данные, что тоже не желательно.

Для таких случаев вы пишете код в методе **onUpgrade()**, чтобы при обновлении поменялась структура базы данных у старых пользователей.

Вариант **onUpgrade()**

метод **onUpgrade()** вызывается при несовпадении версий.

Часто в этом методе просто удаляют существующую таблицу и заменяют её на новую.

Это самое простое и практичное решение.

```
@Override public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
// Запишем в журнал
Log.w("SQLite", "Обновляемся с версии " + oldVersion + " на версию " + newVersion);
// Удаляем старую таблицу и создаём новую
db.execSQL("DROP TABLE IF IT EXISTS " + DATABASE_TABLE); // Создаём новую таблицу
onCreate(db); }
```

Работаем с записями базы данных

Чтобы проверить работоспособность базы данных, в главной активности поместим вспомогательный метод **displayDatabaseInfo()** для отображения информации.

```
private void displayDatabaseInfo() {
 // Создадим и откроем для чтения базу данных
 SQLiteDatabase db = mDbHelper.getReadableDatabase();
// Зададим условие для выборки - список столбцов
String[] projection = { GuestEntry._ID, GuestEntry.COLUMN_NAME,
     GuestEntry.COLUMN_CITY, GuestEntry.COLUMN_GENDER, GuestEntry.COLUMN_AGE };
// Делаем запрос
Cursor cursor = db.query( GuestEntry.TABLE_NAME, // таблица
 projection, // столбцы
 null, // столбцы для условия WHERE
 null, // значения для условия WHERE
 null, // Don't group the rows
 null, // Don't filter by row groups
 null); // порядок сортировки
```

// **Массив projection - это список столбцов, которые нас интересуют. В SQL-запросе мы их указываем в операторе SELECT:**

Работаем с записями базы данных

```
TextView displayTextView = (TextView) findViewById(R.id.text_view_info);
 try { displayTextView.setText("Таблица содержит " + cursor.getCount() + "
     гостей.\n\n");
 displayTextView.append(GuestEntry._ID + " - " + GuestEntry.COLUMN_NAME +
     " - " + GuestEntry.COLUMN_CITY + " - " + GuestEntry.COLUMN_GENDER +
     " - " + GuestEntry.COLUMN_AGE + "\n");
// Узнаем индекс каждого столбца
 int idColumnIndex = cursor.getColumnIndex(GuestEntry._ID);
int nameColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_NAME);
 int cityColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_CITY);
int genderColumnIndex =
     cursor.getColumnIndex(GuestEntry.COLUMN_GENDER);
 int ageColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_AGE);
```

Работаем с записями базы данных

```
// Проходим через все ряды
while (cursor.moveToNext()) {
// Используем индекс для получения строки или числа
int currentID = cursor.getInt(idColumnIndex);
String currentName = cursor.getString(nameColumnIndex);
 String currentCity = cursor.getString(cityColumnIndex);
 int currentGender = cursor.getInt(genderColumnIndex);
 int currentAge = cursor.getInt(ageColumnIndex);
 // Выводим значения каждого столбца
    displayTextView.append(("\n" + currentID + " - " + currentName + "
    - " + currentCity + " - " + currentGender + " - " + currentAge)); } }
    finally {
// Всегда закрываем курсор после чтения
 cursor.close(); } } }
```

Запрос и код в классе

SELECT * FROM guests WHERE _id = 1;

В коде такое выражение выглядело бы так.

String selection = GuestEntry._ID + "=?";

String[] selectionArgs = {"1"};

Как видим, в знак вопроса подставляется нужное значение.

Запрос и код в классе

SELECT name FROM guests WHERE _id > 1 BY age DESC; //
Зададим условие для выборки - список столбцов

```
 String[] projection = { GuestEntry.COLUMN_NAME };
String selection = GuestEntry._ID + ">?";
 String[] selectionArgs = {"1"};
Cursor cursor = db.query( GuestEntry.TABLE_NAME, // таблица
 projection, // столбцы
 selection, // столбцы для условия WHERE
selectionArgs, // значения для условия WHERE
 null, // Don't group the rows
null, // Don't filter by row groups
 GuestEntry.COLUMN_AGE + " DESC"); // порядок сортировки
```

Литература

- http://developer.alexanderklimov.ru/android/sqlite/cathouse2.php