**Справочник по android API**

**База данных по SQLite**

Андроид поддерживает локальные базы данных [SQLite](http://www.sqlite.org/" \t "_blank) для хранения пользовательской информации. SQL cинтаксис данной базы стандартный.

**класс SQLiteOpenHelper**

Для создания, обновления и удаления базы используются вспомогательный класс наследник от *SQLiteOpenHelper*. При запуске новой версии программы через Eclipse файл базы данных сохраняется. Если необходимо пересоздать базу, тогда проще изменить версию базы в коде, чем удалять приложение из телефона.

Перед работой с базой ее необходимо открыть, после завершения работы - обязательно закрыть.

// пример базы с одной таблицей Settings

public class LBD {

// имя файла базы данных

private static final String DATABASE\_NAME = "appstore.db";

// версия базы данных

private static final int DATABASE\_VERSION = 3;

private Context context;

private SQLiteDatabase db;

private Settings settings;

public static LBD get(Context context) {

LBD lbd = new LBD(context);

lbd.open();

return lbd;

}

public LBD(Context c) {

context = c;

}

// открываем базу

final public void open() {

db = (new OpenHelper(this.context)).getWritableDatabase();

settings = null;

}

// закрыть базу (без закрытия базы будет эксцепшн)

final public void close() {

db.close();

}

// получить управление таблицей

final public Settings tblSettings() {

if (settings == null) {

settings = new Settings(db);

}

return settings;

}

// =============================================================================

//

public static class OpenHelper extends SQLiteOpenHelper {

OpenHelper(Context context) {

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

@Override

public void onCreate(SQLiteDatabase db) {

Settings.create(db); // см. пример ниже

}

@Override

public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

Settings.create(db); // просто пересоздаю

}

}

}

**интерфейс Cursor**

Объект с интерфейсом *Cursor* хранит в себе записи выборки sql запроса. Курсор имеет внутренний указатель на текущую запись. В самом начале этот указатель равен -1, поэтому перед началом обработки курсора вызываются методы типа moveToNext(). Существует лимит на число одновременно открытых курсоров. Все открытые курсоры должны быть закрыты. Ниже приведены некоторые методы:

* ***close***() - закрыть курсор;
* ***copyStringToBuffer***(int columnIndex, CharArrayBuffer buffer) сохранить текст указанного столбца в буфер;
* byte[] ***getBlob***(int columnIndex) - получить значение столбца в виде байтового массива;
* int ***getColumnCount***() - число столбцов в записи;
* int ***getColumnIndex***(String columnName) - индекс столбца по его имени, индексация от 0;
* String ***getColumnName***(int columnIndex) - имя столбца по его имени;
* String[] ***getColumnNames***() - массив имен столбцов;
* int ***getCount***() - число записей в выборке;
* double ***getDouble***(int columnIndex) - получить значение столбца в виде double, существуют аналогичные методы для других типов, включая String;
* Bundle ***getExtras***() - дополнительный пакет данных;
* int ***getPosition***() - текущая позиция внутреннего указателя курсора;
* int ***getType***(int columnIndex) - тип данных столбца, может возвращать следующие значения:
  + **FIELD\_TYPE\_BLOB** - бинарный тип;
  + **FIELD\_TYPE\_FLOAT** - тип с плавающей точкой;
  + **FIELD\_TYPE\_INTEGER** - целый тип;
  + **FIELD\_TYPE\_NULL** - null значение;
  + **FIELD\_TYPE\_STRING** - строка;
* boolean ***isAfterLast***() - истина, если курсор указывает на элемент после последней записи;
* boolean ***isBeforeFirst***() - истина, если курсор указывает на элемент перед первой записью;
* boolean ***isClosed***() - истина, если курсор закрыт;
* boolean ***isFirst***() - истина, если курсор указывает на первую запись;
* boolean ***isLast***() - истина, если курсор указывает на последнюю запись;
* boolean ***isNull***(int columnIndex) - истина, если значение указанного столбца null;
* boolean ***move***(int offset) - сдвинуть курсор относительного текущей позиции на указанное число;
* boolean ***moveToFirst***() - установить курсор на первую запись;
* boolean ***moveToNext***() - сдвинуть курсор на следующую запись;
* boolean ***moveToPosition***(int position) - установить курсор в указанную позицию;
* boolean ***moveToPrevious***() - сдвинуть курсор на предыдущую запись;

**курсор и активность**

Активность имеет несколько методов для работы с базой данных. Наиболее полезный - *startManagingCursor*, передает управление курсором системе, которая сама закроет курсор, когда нужно. Это удобно, если курсор используется [прямо в коде активности](http://www.darkraha.com/rus/mobile/android/gui/010_adapter.php#startManagingCursor).

**класс SQLiteDatabase**

Ниже приведены основные методы класса.

* SQLiteStatement ***compileStatement***(String sql) - скомпилировать sql выражение, многократное использование скомпилированных выражений убыстряет обращение к БД;
* void ***execSQL***(String sql) - выполнение SQL запроса не возвращающего данные (не SELECT);
* void ***execSQL***(String sql, Object[] bindArgs) - предыдущее, но также исключаются INSERT/UPDATE/DELETE запросы;
* int ***delete***(String table, String whereClause, String[] whereArgs) - удаление записей, возвращается число удаленных записей;
* long ***insert***(String table, String nullColumnHack, ContentValues values) - вставка записи, возвращается id записи;
* int ***update***(String table, ContentValues values, String whereClause, String[] whereArgs) - обновить записи, возвращается число обновленных записей;
* Cursor ***query***(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy) - выборка записей из таблицы;
* int ***getVersion***() - версия базы данных;
* String ***getPath***() - путь к файлу базы данных;
* void ***setLocale***(Locale locale) - установить локаль базы данных;
* void ***beginTransaction***() - начать транзакцию в экслюзивном режиме;
* void ***endTransaction***() - завершить транзакцию;

Аргументы whereClause и selection представляют блок WHERE sql запроса, но без самого WHERE, аналогично другие блоки. В блоке WHERE позволяется использовать символ ? для задания изменяемых параметров запроса. В этом случае должен быть определен массив whereArgs, число элементов соответствует числу ? в запросе.

Вспомогательный класс *ContentValues* задает значения столбцов для вставляемой/обновляемой записи.

Работу с каждой таблицей удобно вынести в отдельный класс.

public class Settings {

private static final String sqlCREATE = "create table appsettings"

+ " (id INTEGER PRIMARY KEY, "//

+ "name TEXT,"// имя настройки/свойства

+ "value TEXT"// значение

+ ")";

private static final String sqlInsert[] = new String[] {

"insert into appsettings (name, value) values('login','');"//

,

"insert into appsettings (name, value) values('password','');"//

,...

};

private static HashMap<String , String> props = new HashMap<String

, String>();

private static String name = "appsettings";

private static boolean logged = false;

private SQLiteDatabase db;

public Settings(SQLiteDatabase ddb) {

db = ddb;

}

// загрузка свойств из таблицы

public void loadProps() {

Cursor cur = null;

props.clear();

try {// делаем выборку из таблицы

cur = db.query(name, null, null, null, null, null, null);

while (cur.moveToNext()) {

props.put(cur.getString(1), cur.getString(2));

}

} catch (Exception e) {

e.printStackTrace();

} finally {

if (cur != null) {

cur.close();

}

}

}

// создание таблицы при создании базы (см. предыдущий пример)

public static void create(SQLiteDatabase db) {

db.execSQL("DROP TABLE IF EXISTS appsettings");

// db.execSQL("DROP INDEX IF EXISTS appsettings\_nameind");

db.execSQL(sqlCREATE);

for (String ins : sqlInsert) {

db.execSQL(ins);

}

}

}