[onUpgrade. Обновляем БД в SQLite](http://startandroid.ru/ru/uroki/vse-uroki-spiskom/79-urok-39-onupgrade-obnovljaem-bd-v-sqlite.html" \o "Урок 39. onUpgrade. Обновляем БД в SQLite)

С **развитием**приложения может возникнуть необходимость **изменения структуры БД**, которую оно использует. На одном из прошлых уроков я упоминал, что для этого используется метод **onUpgrade**класса **SQLiteOpenHelper**. Этот метод вызывается, если **существующая**версия БД **отличается**от той, к которой мы пытаемся **подключиться**. **Версию**мы обычно **указывали** при вызове **конструктора** супер-класса **SQLiteOpenHelper**в конструкторе DBHelper.

Попробуем воспользоваться методом onUpgrade и посмотреть, как происходит переход на новую версию БД. Для этого напишем небольшое приложение, аналогичное одному из приложений с прошлых уроков – про сотрудников и должности.

**Первая**версия БД будет содержать только таблицу **people**с именем сотрудника и его должностью. Но такая таблица будет не совсем корректна. Если вдруг у нас изменится название должности, придется обновлять все соответствующие записи в people. Поэтому мы решаем изменить БД и организовать данные немного по-другому.

Во **второй**версии добавим таблицу **position**с названием должности и зарплатой. И в таблице **people**вместо названия должности пропишем соответствующий **ID**из **position**.

Создадим проект:

**Project name**: P0391\_SQLiteOnUpgradeDB  
**Build Target**: Android  
**Application name**: SQLiteOnUpgradeDB  
**Package name**: ru.startandroid.develop.p0391sqliteonupgradedb  
**Create Activity**: MainActivity

Экран снова не используем, будем выводить все в лог.

Открываем **MainActivity.java** и кодим:

package ru.startandroid.develop.p0391sqliteonupgradedb;

import android.app.Activity;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.os.Bundle;

import android.util.Log;

public class MainActivity extends Activity {

  final String LOG\_TAG = "myLogs";

  final String DB\_NAME = "staff"; // имя БД

  final int DB\_VERSION = 1; // версия БД

  /\*\* Called when the activity is first created. \*/

  @Override

  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    DBHelper dbh = new DBHelper(this);

    SQLiteDatabase db = dbh.getWritableDatabase();

    Log.d(LOG\_TAG, " --- Staff db v." + db.getVersion() + " --- ");

    writeStaff(db);

    dbh.close();

  }

  // запрос данных и вывод в лог

  private void writeStaff(SQLiteDatabase db) {

    Cursor c = db.rawQuery("select \* from people", null);

    logCursor(c, "Table people");

    c.close();

  }

  // вывод в лог данных из курсора

  void logCursor(Cursor c, String title) {

    if (c != null) {

      if (c.moveToFirst()) {

        Log.d(LOG\_TAG, title + ". " + c.getCount() + " rows");

        StringBuilder sb = new StringBuilder();

        do {

          sb.setLength(0);

          for (String cn : c.getColumnNames()) {

            sb.append(cn + " = "

                + c.getString(c.getColumnIndex(cn)) + "; ");

          }

          Log.d(LOG\_TAG, sb.toString());

        } while (c.moveToNext());

      }

    } else

      Log.d(LOG\_TAG, title + ". Cursor is null");

  }

  // класс для работы с БД

  class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {

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

    }

    public void onCreate(SQLiteDatabase db) {

      Log.d(LOG\_TAG, " --- onCreate database --- ");

      String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша",

          "Борис", "Костя", "Игорь" };

      String[] people\_positions = { "Программер", "Бухгалтер",

          "Программер", "Программер", "Бухгалтер", "Директор",

          "Программер", "Охранник" };

      ContentValues cv = new ContentValues();

      // создаем таблицу людей

      db.execSQL("create table people ("

          + "id integer primary key autoincrement,"

          + "name text, position text);");

      // заполняем ее

      for (int i = 0; i < people\_name.length; i++) {

        cv.clear();

        cv.put("name", people\_name[i]);

        cv.put("position", people\_positions[i]);

        db.insert("people", null, cv);

      }

    }

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

    }

  }

}

Код несложен. Я сгруппировал операции по**выводу в лог данных** из Cursor – метод **logCursor**. Метод **writeStaff**– выбирает данные из таблицы people и вызывает метод для вывода данных в лог. В методе Activity **onCreate**мы создаем объект **DBHelper**,**подключаемся**к БД, выводим в лог **версию**БД, вызываем **writeStaff**и **отключаемся**.

В **DBHelper**все как обычно. В конструкторе вызываем конструктор супер-класса. Обратите внимание, **DB\_VERSION**= 1 – мы будем подключаться к базе версии 1. В методе **onCreate**создаем таблицу и заполняем ее.

Все сохраним и запустим приложение. Смотрим лог:

*--- onCreate database ---   
--- Staff db v.1 ---   
Table people. 8 rows  
id = 1; name = Иван; position = Программер;   
id = 2; name = Марья; position = Бухгалтер;   
id = 3; name = Петр; position = Программер;   
id = 4; name = Антон; position = Программер;   
id = 5; name = Даша; position = Бухгалтер;   
id = 6; name = Борис; position = Директор;   
id = 7; name = Костя; position = Программер;   
id = 8; name = Игорь; position = Охранник;*

БД **создалась**, версия = **1**и данные из таблицы вывелись в лог. Приложение работает, все ок. Но тут мы (внезапно!) понимаем, что при проектировании структуры БД была допущена ошибка. Записывать название должности в поле таблицы people – неправильно. К тому же у нас еще добавляются данные по зарплатам. Надо создать таблицу должностей - position, и использовать из нее id в таблице people. Тем самым структура нашей БД меняется и мы присваиваем ей версию – **2**.

Но наше приложение уже установлено у пользователей. Оно уже создало БД версии 1, и в этой БД уже есть данные. Мы не можем просто удалить существующие таблицы и создать новые, т.к. возможно пользователь уже хранит там свои данные. Нам надо будет написать скрипты для обновления без потери данных.

План обновления такой:

- создаем и заполняем данными таблицу **position**  
- добавляем в таблицу **people**столбец – **posid**для хранения **id**из **position**  
- заполняем**people.posid** данными из **position**в зависимости от значения **people.position**  
- удаляем столбец **people.position**

Давайте менять **MainActivity.java**. Наше приложение теперь будет ориентировано на БД версии 2. Укажем это, изменив значение константы **DB\_VERSION**на 2:

|  |  |
| --- | --- |
| 1 | final int DB\_VERSION = 2; // версия БД |

Метод writeStaff перепишем таким образом:

private void writeStaff(SQLiteDatabase db) {

    Cursor c = db.rawQuery("select \* from people", null);

    logCursor(c, "Table people");

    c.close();

    c = db.rawQuery("select \* from position", null);

    logCursor(c, "Table position");

    c.close();

    String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "

      + "from people as PL "

      + "inner join position as PS "

      + "on PL.posid = PS.id ";

    c = db.rawQuery(sqlQuery, null);

    logCursor(c, "inner join");

    c.close();

  }

Будем выводить в лог данные из таблиц people, position и их объединения.

Реализуем метод обновления - **onUpgrade**в **DBHelper**:

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

  Log.d(LOG\_TAG, " --- onUpgrade database from " + oldVersion

      + " to " + newVersion + " version --- ");

  if (oldVersion == 1 && newVersion == 2) {

    ContentValues cv = new ContentValues();

    // данные для таблицы должностей

    int[] position\_id = { 1, 2, 3, 4 };

    String[] position\_name = { "Директор", "Программер",

        "Бухгалтер", "Охранник" };

    int[] position\_salary = { 15000, 13000, 10000, 8000 };

    db.beginTransaction();

    try {

      // создаем таблицу должностей

      db.execSQL("create table position ("

          + "id integer primary key,"

          + "name text, salary integer);");

      // заполняем ее

      for (int i = 0; i < position\_id.length; i++) {

        cv.clear();

        cv.put("id", position\_id[i]);

        cv.put("name", position\_name[i]);

        cv.put("salary", position\_salary[i]);

        db.insert("position", null, cv);

      }

      db.execSQL("alter table people add column posid integer;");

      for (int i = 0; i < position\_id.length; i++) {

        cv.clear();

        cv.put("posid", position\_id[i]);

        db.update("people", cv, "position = ?",

            new String[] { position\_name[i] });

      }

      db.execSQL("create temporary table people\_tmp ("

          + "id integer, name text, position text, posid integer);");

      db.execSQL("insert into people\_tmp select id, name, position, posid from people;");

      db.execSQL("drop table people;");

      db.execSQL("create table people ("

          + "id integer primary key autoincrement,"

          + "name text, posid integer);");

      db.execSQL("insert into people select id, name, posid from people\_tmp;");

      db.execSQL("drop table people\_tmp;");

      db.setTransactionSuccessful();

    } finally {

      db.endTransaction();

    }

  }

}

Все в соответствии с планом обновления, который я приводил выше. Есть пара нюансов.

Во-первых, используем**БД-транзакцию**. Т.е. нам надо чтобы на БД накатились **все**наши обновления. А в случае **ошибки**в процессе обновления - **все изменения должны быть отменены** и БД должна остаться прежней. Тут транзакции очень выручают.

Во-вторых, в SQLite **нельзя**просто так **удалить**столбец, приходится создавать временную таблицу, перекидывать туда данные, удалять оригинал, создавать его снова с нужной структурой, скидывать в него данные из временной таблицы и удалять временную таблицу. Подробнее об этом можно почитать тут - [How do I add or delete columns from an existing table in SQLite](http://www.sqlite.org/faq.html#q11).

Наше приложение **обновилось**. И теперь, при запуске, оно попытается подключиться к БД версии **2**, но увидит, что существующая версия = **1**и вызовет метод **onUpgrade**, дав нам возможность внести необходимые изменения в структуру БД. Но это произойдет в случае обновления приложения. А что будет если пользователь поставит наше новое приложение на свежий смартфон первый раз?

В этом случае приложение также попытается подключиться к БД версии **2**. Но т.к. приложение только что установлено, то БД еще не существует. Приложение **создаст**БД и присвоит ей версию номер **2**, т.к. оно умеет работать именно с такой версией. При создании будет вызван метод **onCreate**в **DBHelper**. Значит, в нем мы должны прописать код, который будет создавать нам БД версии **2**– т.е. обновленную таблицу **people**и новую таблицу **position**.

Пишем **onCreate**в DBHelper:

public void onCreate(SQLiteDatabase db) {

  Log.d(LOG\_TAG, " --- onCreate database --- ");

  String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша",

      "Борис", "Костя", "Игорь" };

  int[] people\_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };

  // данные для таблицы должностей

  int[] position\_id = { 1, 2, 3, 4 };

  String[] position\_name = { "Директор", "Программер", "Бухгалтер",

      "Охранник" };

  int[] position\_salary = { 15000, 13000, 10000, 8000 };

  ContentValues cv = new ContentValues();

  // создаем таблицу должностей

  db.execSQL("create table position (" + "id integer primary key,"

      + "name text, salary integer" + ");");

  // заполняем ее

  for (int i = 0; i < position\_id.length; i++) {

    cv.clear();

    cv.put("id", position\_id[i]);

    cv.put("name", position\_name[i]);

    cv.put("salary", position\_salary[i]);

    db.insert("position", null, cv);

  }

  // создаем таблицу людей

  db.execSQL("create table people ("

      + "id integer primary key autoincrement,"

      + "name text, posid integer);");

  // заполняем ее

  for (int i = 0; i < people\_name.length; i++) {

    cv.clear();

    cv.put("name", people\_name[i]);

    cv.put("posid", people\_posid[i]);

    db.insert("people", null, cv);

  }

}

Создание и заполнение данными двух таблиц. Все понятно.

Теперь можно все сохранить и запустить приложение.

Смотрим лог:

*--- onUpgrade database from 1 to 2 version ---   
--- Staff db v.2 ---   
Table people. 8 rows  
id = 1; name = Иван; posid = 2;   
id = 2; name = Марья; posid = 3;   
id = 3; name = Петр; posid = 2;   
id = 4; name = Антон; posid = 2;   
id = 5; name = Даша; posid = 3;   
id = 6; name = Борис; posid = 1;   
id = 7; name = Костя; posid = 2;   
id = 8; name = Игорь; posid = 4;   
Table position. 4 rows  
id = 1; name = Директор; salary = 15000;   
id = 2; name = Программер; salary = 13000;   
id = 3; name = Бухгалтер; salary = 10000;   
id = 4; name = Охранник; salary = 8000;   
inner join. 8 rows  
Name = Иван; Position = Программер; Salary = 13000;   
Name = Марья; Position = Бухгалтер; Salary = 10000;   
Name = Петр; Position = Программер; Salary = 13000;   
Name = Антон; Position = Программер; Salary = 13000;   
Name = Даша; Position = Бухгалтер; Salary = 10000;   
Name = Борис; Position = Директор; Salary = 15000;   
Name = Костя; Position = Программер; Salary = 13000;   
Name = Игорь; Position = Охранник; Salary = 8000;*

Видим, что вызывался **onUpgrade**и **обновил**нам БД с версии**1** на **2**. Далее выводим все данные, чтобы убедиться, что обновление прошло корректно.

Можно также убедиться, что новый onCreate в DBHelper корректно отработает. Для этого надо удалить файл БД и запустить приложение. Приложение не найдет БД и создаст ее сразу в новом формате и с версией 2.

Сценарий выдуманный, есть к чему придраться и о чем поспорить, но смысл не в этом. Смысл в том, что мы увидели, как происходит **обновление**БД, если приложение запросило новую версию. Поначалу, возможно, покажется запутанным этот механизм создания и обновления. Но сложного реально ничего нет. С опытом придет полное понимание.

Еще хочу отметить, что у объекта Cursor есть метод close(), который освобождает занимаемые им ресурсы. Не забывайте про него.

Думаю, теперь можно смело сказать, что работу с **SQLite** в Android мы изучили достаточно основательно. И в дальнейших уроках сможем свободно использовать эти знания.

Полный код **MainActivity.java**:

package ru.startandroid.develop.p0391sqliteonupgradedb;

import android.app.Activity;

import android.content.ContentValues;

import android.content.Context;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.os.Bundle;

import android.util.Log;

public class MainActivity extends Activity {

  final String LOG\_TAG = "myLogs";

  final String DB\_NAME = "staff"; // имя БД

  final int DB\_VERSION = 2; // версия БД

  /\*\* Called when the activity is first created. \*/

  @Override

  public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    DBHelper dbh = new DBHelper(this);

    SQLiteDatabase db = dbh.getWritableDatabase();

    Log.d(LOG\_TAG, " --- Staff db v." + db.getVersion() + " --- ");

    writeStaff(db);

    dbh.close();

  }

  // запрос данных и вывод в лог

  private void writeStaff(SQLiteDatabase db) {

    Cursor c = db.rawQuery("select \* from people", null);

    logCursor(c, "Table people");

    c.close();

    c = db.rawQuery("select \* from position", null);

    logCursor(c, "Table position");

    c.close();

    String sqlQuery = "select PL.name as Name, PS.name as Position, salary as Salary "

        + "from people as PL "

        + "inner join position as PS "

        + "on PL.posid = PS.id ";

    c = db.rawQuery(sqlQuery, null);

    logCursor(c, "inner join");

    c.close();

  }

  // вывод в лог данных из курсора

  void logCursor(Cursor c, String title) {

    if (c != null) {

      if (c.moveToFirst()) {

        Log.d(LOG\_TAG, title + ". " + c.getCount() + " rows");

        StringBuilder sb = new StringBuilder();

        do {

          sb.setLength(0);

          for (String cn : c.getColumnNames()) {

            sb.append(cn + " = "

                + c.getString(c.getColumnIndex(cn)) + "; ");

          }

          Log.d(LOG\_TAG, sb.toString());

        } while (c.moveToNext());

      }

    } else

      Log.d(LOG\_TAG, title + ". Cursor is null");

  }

  // класс для работы с БД

  class DBHelper extends SQLiteOpenHelper {

    public DBHelper(Context context) {

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

    }

    public void onCreate(SQLiteDatabase db) {

      Log.d(LOG\_TAG, " --- onCreate database --- ");

      String[] people\_name = { "Иван", "Марья", "Петр", "Антон", "Даша",

          "Борис", "Костя", "Игорь" };

      int[] people\_posid = { 2, 3, 2, 2, 3, 1, 2, 4 };

      // данные для таблицы должностей

      int[] position\_id = { 1, 2, 3, 4 };

      String[] position\_name = { "Директор", "Программер", "Бухгалтер",

          "Охранник" };

      int[] position\_salary = { 15000, 13000, 10000, 8000 };

      ContentValues cv = new ContentValues();

      // создаем таблицу должностей

      db.execSQL("create table position (" + "id integer primary key,"

          + "name text, salary integer" + ");");

      // заполняем ее

      for (int i = 0; i < position\_id.length; i++) {

        cv.clear();

        cv.put("id", position\_id[i]);

        cv.put("name", position\_name[i]);

        cv.put("salary", position\_salary[i]);

        db.insert("position", null, cv);

      }

      // создаем таблицу людей

      db.execSQL("create table people ("

          + "id integer primary key autoincrement,"

          + "name text, posid integer);");

      // заполняем ее

      for (int i = 0; i < people\_name.length; i++) {

        cv.clear();

        cv.put("name", people\_name[i]);

        cv.put("posid", people\_posid[i]);

        db.insert("people", null, cv);

      }

    }

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

      Log.d(LOG\_TAG, " --- onUpgrade database from " + oldVersion

          + " to " + newVersion + " version --- ");

      if (oldVersion == 1 && newVersion == 2) {

        ContentValues cv = new ContentValues();

        // данные для таблицы должностей

        int[] position\_id = { 1, 2, 3, 4 };

        String[] position\_name = { "Директор", "Программер",

            "Бухгалтер", "Охранник" };

        int[] position\_salary = { 15000, 13000, 10000, 8000 };

        db.beginTransaction();

        try {

          // создаем таблицу должностей

          db.execSQL("create table position ("

              + "id integer primary key,"

              + "name text, salary integer);");

          // заполняем ее

          for (int i = 0; i < position\_id.length; i++) {

            cv.clear();

            cv.put("id", position\_id[i]);

            cv.put("name", position\_name[i]);

            cv.put("salary", position\_salary[i]);

            db.insert("position", null, cv);

          }

          db.execSQL("alter table people add column posid integer;");

          for (int i = 0; i < position\_id.length; i++) {

            cv.clear();

            cv.put("posid", position\_id[i]);

            db.update("people", cv, "position = ?",

                new String[] { position\_name[i] });

          }

          db.execSQL("create temporary table people\_tmp ("

              + "id integer, name text, position text, posid integer);");

db.execSQL("insert into people\_tmp select id, name, position, posid from people;");

          db.execSQL("drop table people;");

          db.execSQL("create table people ("

              + "id integer primary key autoincrement,"

              + "name text, posid integer);");

db.execSQL("insert into people select id, name, posid from people\_tmp;");

          db.execSQL("drop table people\_tmp;");

          db.setTransactionSuccessful();

        } finally {

          db.endTransaction();

        }

      }

    }

  }

}