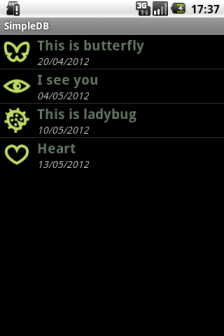
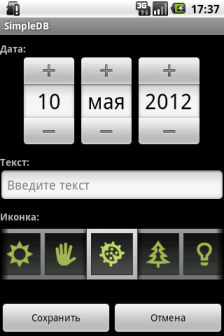
**[Работа с базой данных](http://ru-code-android.livejournal.com/3923.html" \t "_self)**

В этой статье посмотрим как хранить данные в базе данных. Система android предоставляет полную поддержку базы данных [SQLite](http://ru.wikipedia.org/wiki/SQLite" \t "_self). Создадим простое приложение, в котором можно будет создавать, редактировать и удалять записи. Каждая такая запись содержит дату, текст и иконку.   
  
  
  
Для работы с базой нам понадобятся два класса:  
**SQLiteOpenHelper** — класс для создания и обновления базы данных. В приложении необходимо будет создать класс-наследник от SQLiteOpenHelper и описать в нем два метода:  
**onCreate(SQLiteDatabase db)** — Вызывается один раз при создании БД.  
**onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)** — вызывается когда необходимо обновить БД (в данном случае под обновлением имеется в виду не обновление записей, а обновление структуры базы данных)  
Для приложения можно создать несколько БД, все они будут доступны из любого класса программы, но не доступны для других программ.  
  
**SQLiteDatabase** — класс, который предоставляет методы для добавления, обновления, удаления и выборки данных из БД. Основными методами являются:  
  
**long insert(String table, String nullColumnHack, ContentValues values)** — добавление строки в таблицу  
  
*table* — название таблицы  
  
*nullColumnHack* — SQL не позволяет вставлять полностью пустую строку без названия хотя бы одного столбца. То есть выражение:

INSERT INTO my\_table;

завершится ошибкой, необходимо написать хотя бы один столбец, например

INSERT INTO my\_table (my\_column) VALUES (NULL);

Поэтому если values пустой, то в таблицу строка не добавится, если же это необходимо, то в nullColumnHack передается название столбца, которое будет использоваться при построении такого запроса. Очевидно, что столбец при этом может быть любым. Если же при пустом values добавлять строку не надо (что чаще всего и бывает), то передается null.  
  
*values* — структура (ассоциативный массив) содержащая данные которые необходимо вставить. Ключом массива является название столбца, а значением — значение столбца.  
  
Возвращает метод id добавленной строки или -1 если произошла ошибка (например, а случае пустого values строка добавлена не будет и метод вернет -1).  
  
**int update(String table, ContentValues values, String whereClause, String[] whereArgs)** — обновление одной или нескольких строк  
  
*table* и *values* — аналогично как и в insert, название таблицы и значения  
  
*whereClause* и *whereArgs* — используются для построения выражения WHERE в sql. Причем в строке whereClause пишется само выражение, например «column\_one = ? AND column\_two < ?», а в whereArgs передается массив строк, которые будут подставляться вместо знаков «?» в выражение whereClause.  
  
Возвращает метод количество обновленных строк.  
  
**int delete(String table, String whereClause, String[] whereArgs)** — удаление строк  
Параметры аналогичны методу update(), возвращается количество удаленных строк.  
  
**Cursor query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)** — выборка строк  
  
*distinct* — Определяет уникальность выбранных строчек. См [DISTINCT](http://www.w3schools.com/sql/sql_distinct.asp) в sql   
  
*table* — название таблицы  
  
*columns* — массив строк с названиями столбцов, которые необходимо выбрать. Для выборки всех столбцов передается null.  
  
*selection* и *selectionArgs* — условия выборки и аргументы для выборки. Аналогичны whereClause и whereArgs в методе update ()  
  
*groupsBy, having, orderBy, limit* — определяет группировку, сортировку и количество выбираемых строк. Строки имеют тот же формат как они писались бы в sql-выражении (без слов «GROUP BY», «HAVING», «ORDER BY», «LIMIT» соответственно). См [GROUP BY](http://www.w3schools.com/sql/sql_groupby.asp) , [HAVING](http://www.w3schools.com/sql/sql_having.asp) , [ORDER BY](http://www.w3schools.com/sql/sql_orderby.asp) в sql.  
  
**Cursor rawQuery (String sql, String[] selectionArgs)** — метод, который делает то же, что и предыдущий, только здесь запрос пишется как обычный sql-запрос. С помощью этого метода можно производить выборку из нескольких таблиц, использовать JOIN и писать другие витиеватые запросы :)  
  
Два последних метода возвращают экземпляр класса Cursor. Класс **Cursor** содержит все строки, которые выбираются из БД, а также имеет понятие итератора (указателя), который указывает на одну из строк. То есть в каждый момент времени мы можем прочитать значения только одной строки, затем передвигаем итератор на другую строку и читаем ее значения. Метод **int getCount()** - возвращает общее количество строк.   
  
Также в классе есть методы для передвижения итератора:  
**boolean moveToFirst()  
boolean moveToLast()  
boolean moveToNext()  
boolean moveToPosition(int position)  
boolean moveToPrevious()**  
  
Проверки положения итератора:  
**boolean isAfterLast()  
boolean isBeforeFirst()  
boolean isFirst()  
boolean isLast()  
int getPosition()**  
  
И методы возвращающее значение столбца в текущей строке  
**abstract double getDouble(int columnIndex)  
abstract float getFloat(int columnIndex)  
abstract int getInt(int columnIndex)  
abstract long getLong(int columnIndex)  
abstract short getShort(int columnIndex)  
abstract String getString(int columnIndex)  
byte[] getBlob(int columnIndex)**  
  
На этом теория заканчивается, теперь попробуем это на практике.  
  
2. В нашем приложении будет 4 класса. Два из них — наследники класса Activity. Первый (у меня он называется SimpleDBActivity) это список записей (см. скришот выше). Второй (AddActivity) — окно для добавления и обновления записи.   
  
В целях экономии места их код и разметку приводить не буду, можно посмотреть в исходниках. Как сделать список можно почитать [тут](http://ru-code-android.livejournal.com/671.html), а про галерею картинок [тут](http://ru-code-android.livejournal.com/3804.html).  
Третий класс (DBConnector) — класс для работы с БД. В нем будет создаваться база данных, а также будут методы для добавления/обновлени/удаления записей в таблице. Четвертый (MyData) вспомогательный класс описывающий каждую запись списка (и строки в БД). В принципе без него можно обойтись, но с ним код выглядит более понятным и удобочитаемым.   
  
**Создание базы данных**  
3. Начнем как раз с этого четвертого класса. Это просто класс java, который ничего явно не наследует, но реализует интерфейс Serializable, для того чтобы экземпляры класса можно было сериализовать

1. public class MyData implements Serializable{
3. private long id;
4. private long date;
5. private String title;
6. private int icon;
8. public MyData (long id, long date, String title, int icon) {
9. this.id = id;
10. this.date = date;
11. this.title = title;
12. this.icon = icon;
13. }
15. public long getID () {return id;}
16. public long getDate () {return date;}
17. public String getTitle () {return title;}
18. public int getIcon () {return icon;}
19. }

Класс очень простой и не требует никаких пояснений.  
  
4. Перейдем к DBConnector, он также является просто классом java и ничего не наследует. В каждой нашей записи будет 4 поля: дата, название, иконка, а также уникальный идентификатор. В системе android есть соглашение, что поле с идентификатором должно называться «\_id».

1. public class DBConnector {
3. // Данные базы данных и таблиц
4. private static final String DATABASE\_NAME = "simple.db";
5. private static final int DATABASE\_VERSION = 1;
6. private static final String TABLE\_NAME = "MyData";
8. // Название столбцов
9. private static final String COLUMN\_ID = "\_id";
10. private static final String COLUMN\_DATE = "Date";
11. private static final String COLUMN\_TITLE = "Title";
12. private static final String COLUMN\_ICON = "Icon";
14. // Номера столбцов
15. private static final int NUM\_COLUMN\_ID = 0;
16. private static final int NUM\_COLUMN\_DATE = 1;
17. private static final int NUM\_COLUMN\_TITLE = 2;
18. private static final int NUM\_COLUMN\_ICON = 3;
20. private SQLiteDatabase mDataBase;
22. public DBConnector(Context context) {
23. // открываем (или создаем и открываем) БД для записи и чтения
24. OpenHelper mOpenHelper = new OpenHelper(context);
25. mDataBase = mOpenHelper.getWritableDatabase();
26. }
28. // Класс для создания БД
29. private class OpenHelper extends SQLiteOpenHelper {
31. OpenHelper(Context context) {
32. super(context, DATABASE\_NAME, null, DATABASE\_VERSION); }
33. @Override
34. public void onCreate(SQLiteDatabase db) {
35. String query = "CREATE TABLE " + TABLE\_NAME + " (" +
36. COLUMN\_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
37. COLUMN\_DATE + " LONG, " +
38. COLUMN\_TITLE + " TEXT, " +
39. COLUMN\_ICON + " INTEGER); ";
40. db.execSQL(query);
41. }
43. @Override
44. public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
45. db.execSQL("DROP TABLE IF EXISTS " + TABLE\_NAME);
46. onCreate(db);
47. }
48. }
49. }

В классе мы описываем константы для имени и версии БД, названия таблиц и их столбцов. Создаем класс-наследник от SQLiteOpenHelper. В методе onCreate() класса OpenHelper создаем таблицу MyData. В методе onUograde() удаляем таблицу и создаем ее занова. В реальных приложениях обновления структуры БД должно происходить без потери данных, но нам сейчас это не так важно.   
  
В SQLite нет типа данных «Дата», поэтому дату можно хранить в текстовом виде (например в формате «YYYY-MM-DD HH:MM:SS.SSS») или числовом (unixtime — количество секунд с 01.01.1970). Я буду хранить даты в типе long, это не совсем unixtime, это количество миллисекунд с 01.01.1970, но именно это значение возвращает метод Date.getTime().  
  
В конструкторе класса DBConnector создается экземпляр класса SQLiteDatabase, с помощью которого мы сможем добавлять и удалять записи в БД.  
  
Теперь в основном классе SimpleDBActivity (Activity со списком записей) в методе onCreate() можно создать экземпляр класса DBConnector.

1. DBConnector mDBConnector = new DBConnector (this);

Запустим нашу программу, она пока ничего не умеет, но базу данных simple.db с единственной таблицей MyData уже создает.  
  
**Просмотр баз данных на устройстве**  
5. Подключимся к устройству. Для этого используется программа **adb** входящая в состав android SDK (находится в директории platform-tools/). Подключим устройство к компьютеру (или запустим эмулятор) и посмотрим доступные устройства командой:

**$ ./adb devices**  
List of devices attached   
emulator-5554     device   
HT07GPY03496      device

У меня два устройства — эмулятор и телефон, подключимся к эмулятору:

**$ ./adb -s emulator-5554 shell**  
#

Где emulator-5554 — название устройства, а shell — команда, которую необходимо выполнить.   
  
После подключения, можно зайти в базу данных. Делается это командой sqlite3, после которой можно указать название необходимой базы данных. Базы данных приложений хранятся в директориях /data/data//databases/. Подключимся к нашей базе данных

**# sqlite3 /data/data/mj.android.simpledb/databases/simpledb.db**  
SQLite version 3.5.9   
Enter ".help" for instructions   
sqlite>

Советую набрать «.help» и посмотреть список весьма полезных команд, которые есть в sqlite. Например команда «.tables» выведет список таблиц в базе данных.  
  
**Работа с базой данных**  
6. Для работы с базой данных надо добавить в класс DBConnector методы insert(), update(), delete() и т.д.

1. public class DBConnector {
3. ...
5. public DBConnector(Context context) {
6. ...
7. }
9. // Метод добавления строки в БД
10. public long insert(MyData md) {
11. ContentValues cv=new ContentValues();
12. cv.put(COLUMN\_DATE, md.getDate());
13. cv.put(COLUMN\_TITLE, md.getTitle());
14. cv.put(COLUMN\_ICON, md.getIcon());
15. return mDataBase.insert(TABLE\_NAME, null, cv);
16. }
18. // Метод редактирования строки в БД
19. public int update(MyData md) {
20. ContentValues cv=new ContentValues();
21. cv.put(COLUMN\_DATE, md.getDate());
22. cv.put(COLUMN\_TITLE, md.getTitle());
23. cv.put(COLUMN\_ICON, md.getIcon());
24. return mDataBase.update(TABLE\_NAME, cv, COLUMN\_ID + " = ?", new String[] {String.valueOf(md.getID()) });
25. }
27. // Метод удаления всех записей из БД
28. public int deleteAll() {
29. return mDataBase.delete(TABLE\_NAME, null, null);
30. }
32. // Метод удаления записи
33. public void delete(long id) {
34. mDataBase.delete(TABLE\_NAME, COLUMN\_ID + " = ?", new String[] { String.valueOf(id) });
35. }
37. // Метод выборки одной записи
38. public MyData select(long id) {
39. Cursor mCursor = mDataBase.query(TABLE\_NAME, null, COLUMN\_ID + " = ?", new String[] {String.valueOf(id)}, null, null, COLUMN\_DATE);
41. mCursor.moveToFirst();
42. long date = mCursor.getLong(NUM\_COLUMN\_DATE);
43. String title = mCursor.getString(NUM\_COLUMN\_TITLE);
44. int icon = mCursor.getInt(NUM\_COLUMN\_ICON);
45. mCursor.close();
46. return new MyData(id, date, title, icon);
47. }
49. // Метод выборки всех записей
50. public ArrayList<MyData> selectAll() {
51. Cursor mCursor = mDataBase.query(TABLE\_NAME, null, null, null, null, null, COLUMN\_DATE);
53. ArrayList<MyData> arr = new ArrayList<MyData>();
54. mCursor.moveToFirst();
55. if (!mCursor.isAfterLast()) {
56. do {
57. long id = mCursor.getLong(NUM\_COLUMN\_ID);
58. long date = mCursor.getLong(NUM\_COLUMN\_DATE);
59. String title = mCursor.getString(NUM\_COLUMN\_TITLE);
60. int icon = mCursor.getInt(NUM\_COLUMN\_ICON);
61. arr.add(new MyData(id, date, title, icon));
62. } while (mCursor.moveToNext());
63. }
64. mCursor.close();
65. return arr;
66. }
68. // Класс для создания БД
69. private class OpenHelper extends SQLiteOpenHelper {
70. ...
71. }
72. }

В методы insert() и update() мы передаем экземпляр нашего класса MyData. Причем в методе insert() у объекта MyData поле id равно null, так как при добавлении sqlite сама сгенерирует его значение.  
  
Класс для работы с БД готов, теперь можно его использовать. В классе SimpleDBActivity в методе onCreate() напишем:

1. public void onCreate(Bundle savedInstanceState) {
3. ...
5. // Класс для работы с БД
6. mDBConnector = new DBConnector (this);
7. mListView = (ListView)findViewById(R.id.list);
8. // Адаптер списка
9. mAdapter = new myListAdapter(mContext);
10. // Передаем в адаптер массив объектов MyData, которые необходимо вывести
11. mAdapter.setArrayMyData(mDBConnector.selectAll());
12. mListView.setAdapter(mAdapter);
13. ...
14. }

А сам адаптер списка выглядит как-то так:

1. class myListAdapter extends BaseAdapter {
2. private LayoutInflater mLayoutInflater;
3. private ArrayList<MyData> arrayMyData;
5. public myListAdapter (Context ctx) {
6. mLayoutInflater = LayoutInflater.from(ctx);
7. }
9. public void setArrayMyData(ArrayList<MyData> arrayMyData) {
10. this.arrayMyData = arrayMyData;
11. }
13. ...
15. public View getView(int position, View convertView, ViewGroup parent) {
16. if (convertView == null)
17. convertView = mLayoutInflater.inflate(R.layout.item, null);
19. ImageView vIcon = (ImageView)convertView.findViewById(R.id.Icon);
20. TextView vTitle = (TextView)convertView.findViewById(R.id.Title);
21. TextView vDate = (TextView)convertView.findViewById(R.id.Date);
23. SimpleDateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");
25. MyData md = arrayMyData.get(position);
26. vDate.setText(dateFormat.format(md.getDate()));
27. vTitle.setText(md.getTitle());
28. vIcon.setImageResource(md.getIcon());
30. return convertView;
31. }
32. }

Мы передаем в адаптер массив объектов MyData и затем выводим их. Более подробно про адапетеры списков читать [тут](http://ru-code-android.livejournal.com/671.html).  
  
В статье разобраны основные методы по работе с БД, более подробное их использование можно посмотреть в исходниках.  
  
Исходники:  
[SimpleDB.zip](http://narod.ru/disk/49014492001.93178efaf371213e3fc85ac1c84325c9/SimpleDB.zip.html)