



*Белорусский государственный университет
информатики и радиоэлектроники*

Эргономика мобильных приложений

Преподаватель: Меженная Марина Михайловна

К.Т.Н., доцент,

доцент кафедры инженерной психологии и эргономики
а 609-2

mezhennaya@bsuir.by



Кафедра инженерной психологии и эргономики

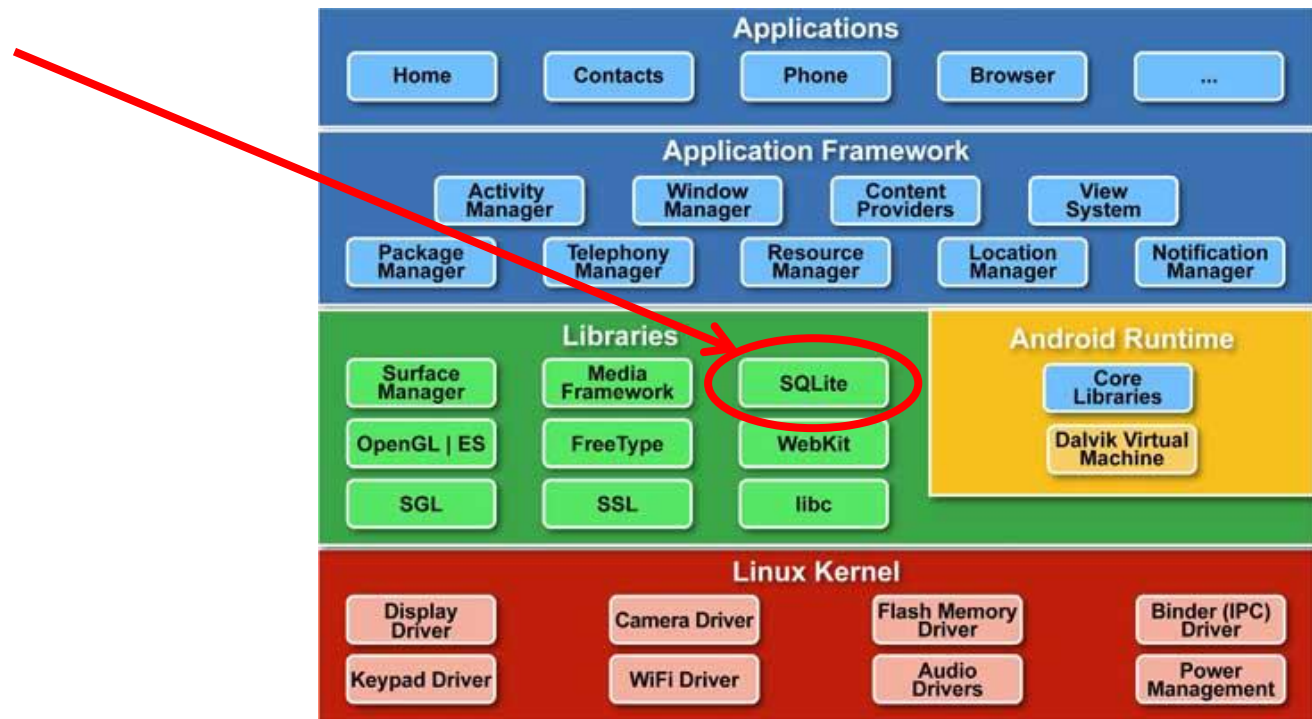
Лекция 8: База данных SQLite

План лекции:

1. Особенности работы с SQLite в Android-платформе
2. Стратегия простейшей работы с SQLite в Android-платформе
3. Пример простейшей работы с SQLite
4. Пример работы с SQLite через список

Особенности работы с SQLite в Android-платформе

SQLite доступен на любом Android-устройстве, его не нужно устанавливать отдельно.



Особенности работы с SQLite в Android-платформе

SQLite поддерживает типы:

- TEXT (аналог String в Java),
- INTEGER (аналог long в Java)
- REAL (аналог double в Java).

Остальные типы следует конвертировать, прежде чем сохранять в базе данных.

Для хранения изображений в базе данных указывают путь к изображениям, а сами изображения хранят в файловой системе.

Особенности работы с SQLite в Android-платформе

Работа с SQLite в Android-приложениях в реальности всегда выполняется в отдельном (не в UI) потоке, чтобы не возникало временных задержек в работе пользователя с приложением.

Стратегия простейшей работы с SQLite в Android-платформе

Работа с базой данных сводится к следующим задачам:

1.Создание базы данных, создание таблицы.

Класс DBHelper

2.Открытие базы данных

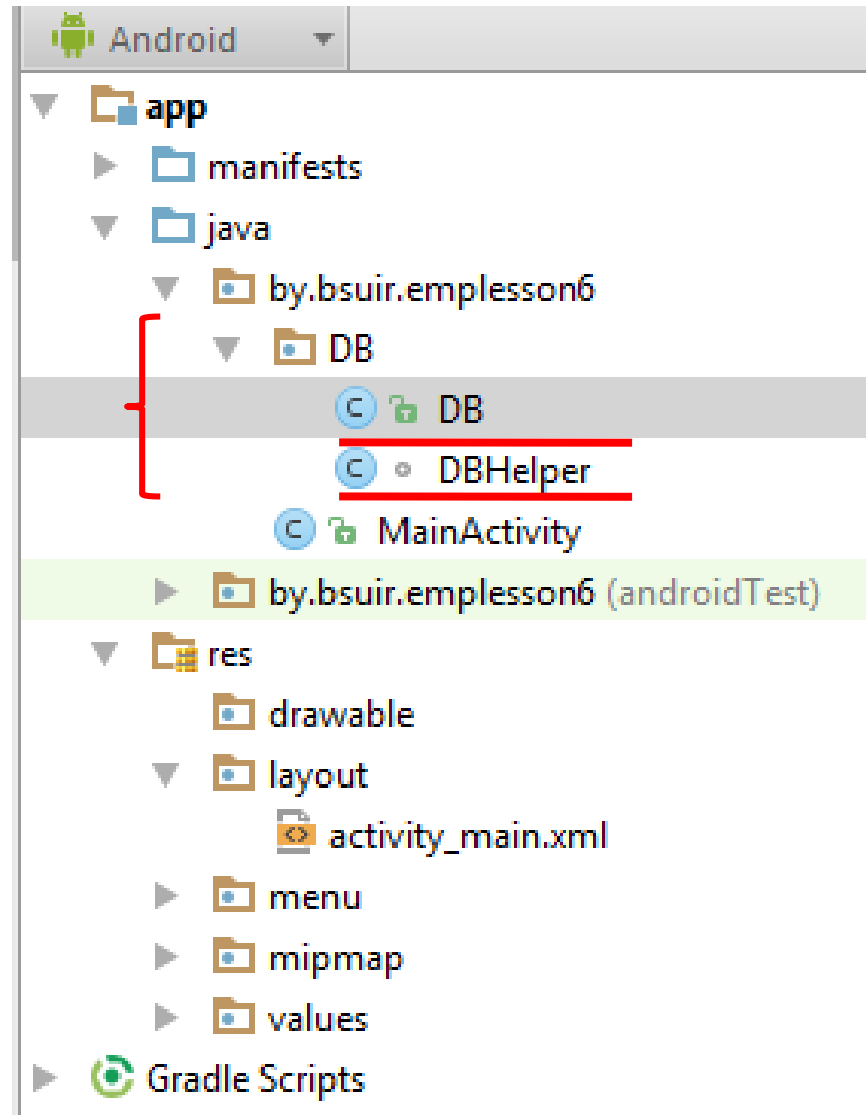
3.Создание интерфейса для вставки данных.

4.Создание интерфейса для выполнения запросов (выборки данных).

5.Заккрытие базы данных.

Класс DB

Простейшая структура проекта с SQLite



Стратегия простейшей работы с SQLite в Android-платформе

Для работы с SQLite в Android-приложениях используются вспомогательные классы:

SQLiteOpenHelper

DBHelper extends SQLiteOpenHelper

ContentValues

Cursor

Класс DB

Стратегия простейшей работы с SQLite в Android-платформе

DBHelper extends SQLiteOpenHelper

1.Создание базы данных, создание таблицы.

Реализуется с помощью класса SQLiteOpenHelper.

Стратегия простейшей работы с SQLite в Android-платформе

Класс DB

2. Открытие базы данных.

Открытие базы данных, как правило, выполняется путем вызова соответствующего метода класса DB в методе onCreate() java-класса Activity.

4. Заккрытие базы данных.

Заккрытие базы данных, как правило, выполняется путем вызова соответствующего метода класса DB в методе onDestroy() java-класса Activity.

Стратегия простейшей работы с SQLite в Android-платформе

Класс DB

3.Создание интерфейса для вставки данных.

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.

Стратегия простейшей работы с SQLite в Android-платформе

Класс DB

4.Создание интерфейса для выполнения запросов (выборки данных).

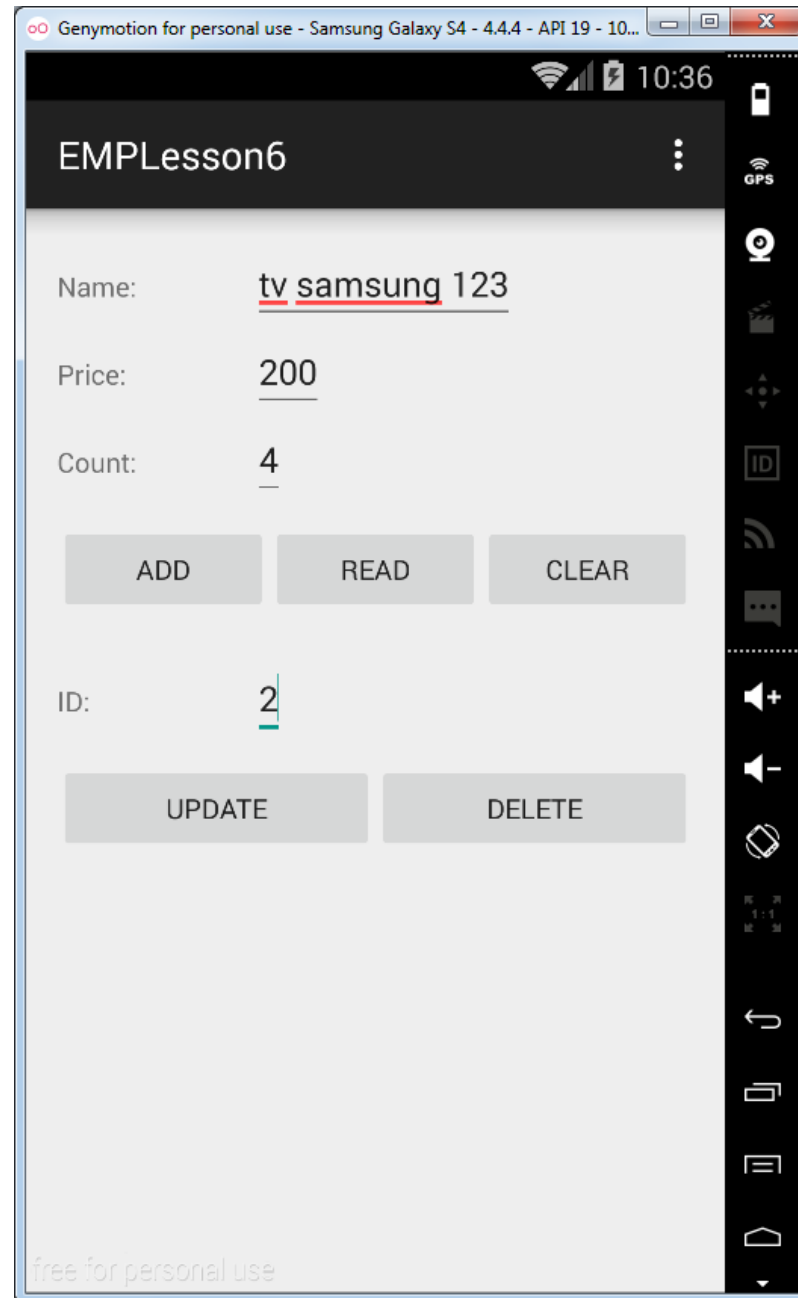
Запросы к базе данных возвращают объекты класса **Cursor**. Вместо того чтобы извлекать данные и возвращать копию значений, курсоры ссылаются на результирующий набор исходных данных. Курсоры позволяют управлять текущей позицией (строкой) в результирующем наборе данных, возвращаемом при запросе.

Пример

Создадим таблицу товаров с id, name, price, count.

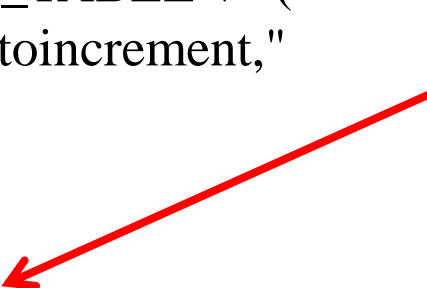
Для работы с таблицей товаров реализуем функции:

- добавления (кнопка Add),
- чтения всех данных таблицы и вывода информации в лог (кнопка Read),
- очистки всей таблицы (кнопка Clear),
- ввода name, price, count в качестве новых данных уже существующего товара с номером id (кнопка Update),
- удаления товара по id (кнопка Delete).



DBHelper extends SQLiteOpenHelper

```
class DBHelper extends SQLiteOpenHelper {  
  
    private static final String DB_TABLE = "goods";  
  
    public DBHelper(Context context, String name,  
        SQLiteDatabase.CursorFactory factory, int version) {  
        super(context, name, factory, version);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL("create table " + DB_TABLE + "("  
            + "id integer primary key autoincrement,"  
            + "name text,"  
            + "price integer,"  
            + "count integer" + ");");  
    }  
}
```



Таблиц
может быть и
больше...

DBHelper extends SQLiteOpenHelper

@Override

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int  
newVersion) {  
    // Удаляем старую таблицу и создаём новую  
    db.execSQL("DROP TABLE IF EXISTS " + DB_TABLE);  
    onCreate(db);  
}  
}  
}
```

Нельзя после создания БД и таблицы (таблиц) вручную вносить изменения в структуру БД (добавлять поля, изменять имена/типы столбцов и т.д.) – проект не запустится! Это необходимо делать программно (либо через повышение текущей версии БД, либо с помощью спец. методов).

Класс DB

```
public class DB {  
  
    private static final String DB_NAME = "mydb";  
    private static final int DB_VERSION = 1;  
    private static final String DB_TABLE = "goods";  
    private final Context mContext;  
    private DBHelper mDBHelper;  
    private SQLiteDatabase mDB;  
  
    public DB(Context ctx) {  
        mContext = ctx;  
    }  
}
```


Класс DB

```
public DB(Context ctx) {  
    mCtx = ctx;  
}  
  
// открыть подключение  
public void open() {  
    mDBHelper = new DBHelper(mCtx, DB_NAME, null,  
DB_VERSION);  
    mDB = mDBHelper.getWritableDatabase();  
}  
  
// закрыть подключение  
public void close() {  
    if (mDBHelper!=null) mDBHelper.close();  
}
```

Класс DB

// заполнить таблицу исходными данными при необходимости

```
public void write() {  
    int i=0;  
    while (i<25) {  
        i++;  
        addRec("My good №" + i, i, i);  
    }  
}
```

// получить все данные из таблицы DB_TABLE

```
public Cursor getAllData() {  
    return mDB.query(DB_TABLE, null, null, null, null, null, null);  
}
```

Класс DB

// добавить запись в DB_TABLE

```
public void addRec(String name, int price, int count) {  
    ContentValues cv = new ContentValues();  
    cv.put("name", name);  
    cv.put("price", price);  
    cv.put("count", count);  
    mDB.insert(DB_TABLE, null, cv);  
}
```

// обновить запись в DB_TABLE

```
public void update(int id, String name, int price, int count) {  
    ContentValues cv = new ContentValues();  
    cv.put("name", name);  
    cv.put("price", price);  
    cv.put("count", count);  
    mDB.update(DB_TABLE, cv, "id = ?",  
        new String[]{String.valueOf(id)});  
}
```

Класс DB

```
// удалить запись из DB_TABLE
public void delRec(long id) {
    mDB.delete(DB_TABLE, "id = " + id, null);
}

// удалить все записи из DB_TABLE
public void delAll() {
    mDB.delete(DB_TABLE, null, null);
}
}
```

Можно продолжать код посредством добавления
необходимых Вам методов...

Можно продолжать код посредством добавления
необходимых Вам методов...

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

Примеры поиска по имени, по стоимости:

```
mDB.query.query(DB_TABLE, null, " name = ?",  
    new String[] { "Samsung Galaxy" }, null, null, null);  
mDB.query(DB_TABLE, null, "count = ?",  
    new String[] { Integer.toString(100) }, null, null, null);
```

В метод `query()` передают семь параметров. Если какой-то параметр для запроса не имеет значения — указывают `null`:

`table` — имя таблицы, к которой передается запрос;

`String[] columnNames` — список имен возвращаемых полей (массив). При передаче `null` возвращаются все столбцы.

`String whereClause` — параметр, формирующий выражение WHERE (исключая сам оператор WHERE). Значение `null` возвращает все строки. Например: `id = 19 and summary = ?`

`String[] selectionArgs` — значения аргументов фильтра. Вы можете включить `?` в `"whereClause"`. Подставляется в запрос из заданного массива;

`String[] groupBy` — фильтр для группировки, формирующий выражение GROUP BY (исключая сам оператор GROUP BY).

`String[] having` — фильтр для группировки, формирующий выражение HAVING (исключая сам оператор HAVING). Если не нужен, передается `null`;

`String[] orderBy` — параметр, формирующий выражение ORDER BY (исключая сам оператор ORDER BY). При сортировке по умолчанию передается `null`.

MainActivity

```
public class MainActivity extends ActionBarActivity implements  
View.OnClickListener{
```

```
    final String LOG_TAG = "myLogs";
```

```
    private Context context;
```

```
    private DB db;
```

```
    private Cursor cursor;
```

```
    private int idColIndex;
```

```
    private int nameColIndex;
```

```
    private int priceColIndex;
```

```
    private int countColIndex;
```

```
    private EditText etName, etPrice, etCount, etId;
```

```
    private Button btnAdd, btnRead, btnClear, btnUpdate, btnDelete;
```

```
    private String name_temp;
```

```
    private int id_temp, price_temp, count_temp;
```

MainActivity

@Override

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    initView();  
    initDB();  
}  
  
private void initView() {  
    etName = (EditText) findViewById(R.id.etName);  
    etPrice = (EditText) findViewById(R.id.etPrice);  
    etCount = (EditText) findViewById(R.id.etCount);  
    etId = (EditText) findViewById(R.id.etId);  
    btnAdd = (Button) findViewById(R.id.btnAdd);  
    btnAdd.setOnClickListener(this);  
    btnRead = (Button) findViewById(R.id.btnRead);  
    btnRead.setOnClickListener(this);  
    btnClear = (Button) findViewById(R.id.btnClear);  
    btnClear.setOnClickListener(this);  
    btnUpdate = (Button) findViewById(R.id.btnUpdate);  
    btnUpdate.setOnClickListener(this);  
    btnDelete = (Button) findViewById(R.id.btnDelete);  
    btnDelete.setOnClickListener(this);  
}
```


MainActivity

```
private void initDB() {  
    // открываем подключение к БД  
    db = new DB(this);  
    db.open();  
    //db.delAll(); // если нужно все вернуть к исходному состоянию  
    db.write(); // при каждом запуске дописываем данные из write()  
}
```

@Override

```
public void onClick(View view) {  
    switch (view.getId()){  
        case R.id.btnAdd:  
            name_temp = etName.getText().toString();  
            price_temp = Integer.parseInt(etPrice.getText().toString());  
            count_temp = Integer.parseInt(etCount.getText().toString());  
            db.addRec(name_temp, price_temp, count_temp);  
            break;
```

MainActivity

```
case R.id.btnRead:
```

```
    cursor = db.getAllData();
```

```
    // ставим позицию курсора на первую строку выборки
```

```
    if (cursor.moveToFirst()) {
```

```
        int idColIndex = cursor.getColumnIndex("id");
```

```
        int nameColIndex = cursor.getColumnIndex("name");
```

```
        int priceColIndex = cursor.getColumnIndex("price");
```

```
        int countColIndex = cursor.getColumnIndex("count");
```

```
        do {
```

```
            // получаем значения по номерам столбцов и пишем в лог
```

```
            Log.d(LOG_TAG,
```

```
                "ID = " + cursor.getInt(idColIndex) +
```

```
                ", name = " + cursor.getString(nameColIndex) +
```

```
                ", price = " + cursor.getInt(priceColIndex) +
```

```
                ", count = " + cursor.getInt(countColIndex));
```

```
        } while (cursor.moveToNext());
```

```
    } else {
```

```
        Log.d(LOG_TAG, "0 rows");
```

```
    }
```

```
    Log.d(LOG_TAG, "cursor.getCount()=" + String.valueOf(cursor.getCount()));
```

```
    cursor.close();
```

```
    break;
```

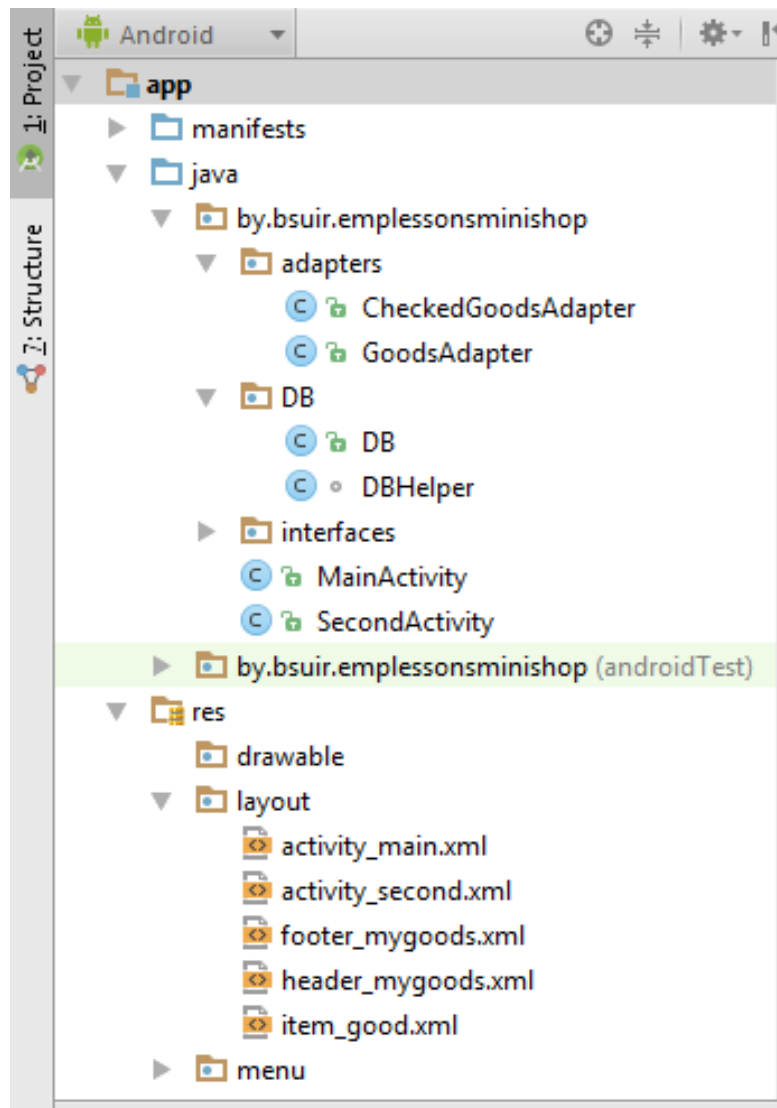
MainActivity

```
case R.id.btnClear:
    db.delAll();
    break;
case R.id.btnUpdate:
    id_temp = Integer.parseInt(etId.getText().toString());
    name_temp = etName.getText().toString();
    price_temp = Integer.parseInt(etPrice.getText().toString());
    count_temp = Integer.parseInt(etCount.getText().toString());
    db.update(id_temp, name_temp, price_temp, count_temp);
    break;
case R.id.btnDelete:
    id_temp = Integer.parseInt(etId.getText().toString());
    db.delRec(id_temp);
    break;
}
}
```

MainActivity

```
protected void onDestroy() {  
    super.onDestroy();  
    // закрываем подключение к базе данных при выходе  
    db.close();  
}  
}
```

Пример работы со списком



Далее приведен пример реализации класса `GoodsAdapter` для решения задачи приложения `MiniShop`, ранее реализуемой посредством предоставления адаптеру набора объектов класса `Good`. В примере реализованы паттерны проектирования `CursorLoader` (для выгрузки информации из базы данных в **фоновом** режиме) и `ViewHolder` (внутренний класс для удобной работы с графическим представлением одного элемента списка).

EMPLesson4

My Goods List

1	My good №1	<input type="checkbox"/>
2	My good №2	<input checked="" type="checkbox"/>
3	My good №3	<input checked="" type="checkbox"/>
4	My good №4	<input type="checkbox"/>
5	My good №5	<input type="checkbox"/>
6	My good №6	<input type="checkbox"/>
7	My good №7	<input type="checkbox"/>
8	My good №8	<input type="checkbox"/>
9	My good №9	<input type="checkbox"/>
10	My good №10	<input type="checkbox"/>
11	My good №11	<input type="checkbox"/>
12	My good №12	<input type="checkbox"/>
13	My good №13	<input type="checkbox"/>

for personal use

EMPLesson4

11	My good №11	<input type="checkbox"/>
12	My good №12	<input type="checkbox"/>
13	My good №13	<input type="checkbox"/>
14	My good №14	<input type="checkbox"/>
15	My good №15	<input type="checkbox"/>
16	My good №16	<input type="checkbox"/>
17	My good №17	<input type="checkbox"/>
18	My good №18	<input type="checkbox"/>
19	My good №19	<input checked="" type="checkbox"/>
20	My good №20	<input type="checkbox"/>
21	My good №21	<input type="checkbox"/>
22	My good №22	<input type="checkbox"/>
23	My good №23	<input type="checkbox"/>
24	My good №24	<input type="checkbox"/>
25	My good №25	<input type="checkbox"/>

SHOW CHECKED ITEMS

Count of goods = 3

for personal use

```
public class MainActivity extends ActionBarActivity implements  
OnChangeListener, View.OnClickListener,  
LoaderManager.LoaderCallbacks<Cursor> {
```

```
    private ListView listView;  
    private DB db;  
    private Cursor cursor;  
    private View view_header, view_footer;  
    private LayoutInflater inflater;  
    private Button btnShow;  
    private TextView tv_count;  
    private GoodsAdapter goodsAdapter;  
    private int count_checked_goods = 0;
```

@Override

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
    initView();  
    initDB();  
    createMyListView();  
}
```

```
private void initView() {  
    listView = (ListView) findViewById(R.id.listView);  
}
```

```
private void initDB() {  
    db = new DB(this);  
}
```



```
private void createMyListView() {  
    inflater = LayoutInflater.from(this);  
    view_header = inflater.inflate(R.layout.header_mygoods, null);  
    view_footer = inflater.inflate(R.layout.footer_mygoods, null);  
    btnShow = (Button) view_footer.findViewById(R.id.btnShow);  
    btnShow.setOnClickListener(this);  
    tv_count = (TextView) view_footer.findViewById(R.id.tv_count);  
    getSupportLoaderManager().initLoader(0, null, this);  
}
```

```
protected void onDestroy() {  
    super.onDestroy();  
    db.close();  
}
```

@Override

```
public void onClick(View view) {  
    Intent intent = new Intent(this, SecondActivity.class);  
    startActivity(intent);  
}
```

@Override

```
public void onDataChange(int id, String name, int price, int count) {  
    db.update(id, name, price, count);  
    getSupportLoaderManager().getLoader(0).forceLoad();  
    if (count==0) {  
        count_checked_goods--;  
    } else {  
        count_checked_goods++;  
    }  
    tv_count.setText("Count of goods = " + count_checked_goods + "");  
}
```

@Override

```
public Loader<Cursor> onCreateLoader(int id, Bundle args) {  
    return new MyCursorLoader(this, db);  
}
```

@Override

```
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {  
    if (goodsAdapter == null) {  
        goodsAdapter = new GoodsAdapter(MainActivity.this, data, this);  
        listView.addHeaderView(view_header);  
        listView.addFooterView(view_footer);  
        listView.setAdapter(goodsAdapter);  
    } else {  
        goodsAdapter.refreshCursor(data);  
    }  
}
```

@Override

```
public void onLoaderReset(Loader<Cursor> loader) {  
}
```

```
static class MyCursorLoader extends CursorLoader {  
  
    DB db;  
  
    public MyCursorLoader(Context context, DB db) {  
        super(context);  
        this.db = db;  
    }  
  
    @Override  
    public Cursor loadInBackground() {  
        Cursor cursor = db.getAllData();  
        return cursor;  
    }  
}
```

```
public class GoodsAdapter extends BaseAdapter implements
CompoundButton.OnCheckedChangeListener {

    private Context context;
    private LayoutInflater inflater;
    private DB db;
    private Cursor cursor;
    private int idColIndex;
    private int nameColIndex;
    private int priceColIndex;
    private OnChangeListener onChangeListener;
```

```
public GoodsAdapter(Context context, Cursor cursor, OnChangeListener
onChangeListener) {
    this.context = context;
    this.inflater = LayoutInflater.from(context);
this.cursor = cursor;
    this.onChangeListener = onChangeListener;
    idColIndex = cursor.getColumnIndex("id");
    nameColIndex = cursor.getColumnIndex("name");
    priceColIndex = cursor.getColumnIndex("price");
    }
}

@Override
public int getCount() {
    return cursor.getCount();
}
```

@Override

```
public Cursor getItem(int i) {  
    cursor.moveToPosition(i);  
    return cursor;  
}
```

@Override

```
public long getItemId(int i) {  
    return i;  
}
```

@Override

```
public View getView(int position, View view, ViewGroup parent) {  
    if (view == null) {  
        view = inflater.inflate(R.layout.item_good, null);  
    }  
    cursor.moveToPosition(position);  
    ViewHolder vh = new ViewHolder();  
    vh.initViewHolder(view);  
    vh.tv_goodPrice.setText(cursor.getInt(priceColIndex)+"");  
    vh.tv_goodName.setText(cursor.getString(nameColIndex)+"");  
    if (cursor.getInt(countColIndex) == 0) {  
        vh.cb_good.setChecked(false);  
    } else {  
        vh.cb_good.setChecked(true);  
    }  
    vh.cb_good.setOnCheckedChangeListener(this);  
    vh.cb_good.setTag(position);  
    return view;  
}
```


@Override

```
public void onCheckedChanged(CompoundButton compoundButton,
    boolean isChecked) {
    if (compoundButton.isShown()) {
        int i = (int) compoundButton.getTag();
        cursor.moveToPosition(i);
        int id = cursor.getInt(idColIndex);
        String name = cursor.getString(nameColIndex);
        int price = cursor.getInt(priceColIndex);
        int check = 0;
        if (isChecked){
            check = 1;
        }
        onChangeListener.onDataChanged(id, name, price, check);
    }
}
```

```
public void refreshCursor(Cursor data) {
    cursor = data;
}
```

```
public class ViewHolder {

    private TextView tv_goodPrice;
    private TextView tv_goodName;
    private CheckBox cb_good;

    public ViewHolder() {
    }

    public void initViewHolder(View view) {
        tv_goodPrice = (TextView) view.findViewById(R.id.tv_goodPrice);
        tv_goodName = (TextView) view.findViewById(R.id.tv_goodName);
        cb_good = (CheckBox) view.findViewById(R.id.cb_good);
    }
}
```