PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS

Prof. Luiz Carlos Querino Filho
luiz.querino@fatec.sp.gov.br

Fatec Garça – 2018

Parte 05

Armazenamento em banco de dados

- A lista de compras desenvolvida em laboratório não armazena os itens entre uma execução e outra do aplicativo.
- Para que os itens da lista persistam entre as execuções do programa, é preciso que estes fiquem gravados permanentemente no dispositivo.
- O Android disponibiliza algumas formas de realizar este armazenamento, como por exemplo:
  - Usar **arquivos de preferências**, armazenando dados no formato "chave" = "valor", por meio da classe `SharedPreferences`
  - Criar um **banco de dados local** utilizando as bibliotecas do **SQLite** embutidas no Android.
- A segunda abordagem é mais poderosa e flexível; será ela que vamos utilizar na Lista de Compras

O banco de dados SQLite

- NÃO É UM SGBD - não requer instalação própria ou programa servidor (*serverless*)
- Na verdade é uma biblioteca compacta, com pouco mais de 300 KB de tamanho.
- Suporta várias linguagens de programação, com API de acesso estruturada ou orientada a objetos.
- Utiliza a sintaxe de consultas em linguagem SQL
- É leve, rápido e suporta grande volume de dados:
  - 50.000 inclusões por segundo, em um computador desktop padrão
  - Suporte a arquivo de banco de dados de até 140 TB (ou no limite do sistema operacional)
- É o banco de dados incluso no Android e no iPhone
- Totalmente livre - domínio público

SQLite – Linguagem SQL

- Criação de tabelas

```
CREATE TABLE nome_da_tabela (
        nome_do_campo  tipo [PRIMARY KEY],
        ...,
);
```

- Campos chave primária serão auto incrementados, se receberem um valor vazio (NULL) na inclusão.
- Tipos de Dados Básicos:
  - INTEGER Um número inteiro
  - REAL Um número com casas decimais
  - TEXT Uma string de texto
  - BLOB Dados binários (arquivo)

Usando o `SQLite` no Android

- O Android vêm com classes prontas para utilização de um banco de dados **SQLite**. Estas classes estão dentro do pacote **`android.database.sqlite`**:
  - **SQLiteOpenHelper**: é usada para a criação e abertura do arquivo de banco de dados. Quando instanciada, passamos o nome do banco de dados que será aberto como parâmetro.
  - **SQLiteDatabase**: um objeto desta classe é retornado pelo método **`getWritableDatabase()`**, de um objeto da classe **`SQLiteOpenHelper`**. Com ele, é possível manipular diretamente o banco de dados, com consultas, inclusões, exclusões e alterações.
- **Cursor**: os resultados obtidos em uma consulta ao banco de dados são gerenciados por um objeto desta classe, que fica dentro do pacote **`android.database`**.

Criando a classe DAO

- Para continuar com nossa prática de dividir a aplicação em camadas bem definidas, não vamos "misturar" o código de manipulação do banco de dados com o das Activities.
- Em aplicações com acesso a banco de dados é interessante colocar estas tarefas em uma classe separada.
- Por essa razão, vamos adicionar ao nosso projeto uma classe com esta finalidade.
- Crie uma nova classe Java denominada **Dao**
- "Dao" é uma sigla para *Data Access Object* (objeto de acesso a dados)

BancoListaOpenHelper.java

- Precisamos criar a classe **BancoListaOpenHelper** antes da **Dao**, pois a primeira será usada pela última.
- A função desta classe é, herdando a classe **SQLiteOpenHelper**, fazer a criação do banco de dados e atualizar sua estrutura, quando necessário.
- Explicação do código-fonte (no próximo slide):
  - **Linhas 15-30:** implementa o método **`onCreate`**. Este método é executado automaticamente quando um banco de dados ainda não existente for utilizado. Contém a definição de uma **String** com um comando SQL para criação da tabela dos produtos da lista. O campo **`_id`** atua como um código único para cada produto. Em seguida à criação é feita a inclusão de dois itens à lista.
  - **Linhas 23 e 24:** contém apenas o "cabeçalho" de um método **`onUpgrade`** (não implementado aqui). Ele pode ser usado posteriormente para modificar a estrutura do banco de dados no caso de uma atualização do sistema.

BancoListaOpenHelper.java

```java
package com.example.alunos.listadecompras;

import ...

public class BancoListaOpenHelper extends SQLiteOpenHelper {

    public BancoListaOpenHelper(Context context, String name,
                                SQLiteDatabase.CursorFactory factory, int version) {
        super(context, name, factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        String comando = "CREATE TABLE itens (" +
                "_id INTEGER PRIMARY KEY AUTOINCREMENT," +
                "descricao TEXT," +
                "quantidade INTEGER," +
                "comprado INTEGER" +
                ")";
        sqLiteDatabase.execSQL(comando);

        comando = "INSERT INTO itens (descricao, quantidade, comprado) " +
                "VALUES ('Banana', 12, 0)";
        sqLiteDatabase.execSQL(comando);

        comando = "INSERT INTO itens (descricao, quantidade, comprado) " +
                "VALUES ('Maçã', 6, 0)";
        sqLiteDatabase.execSQL(comando);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {

    }
}
```

Dao.java

```java
package com.example.alunos.listadecompras;

import ...

public class Dao {

    private BancoListaOpenHelper openHelper;
    private SQLiteDatabase database;

    public Dao(Context contexto) {
        openHelper = new BancoListaOpenHelper(contexto, "lista.sql", null, 1);
    }

    public void abrir() { database = openHelper.getWritableDatabase(); }

    public void fechar() {
        if (database != null) { database.close(); }
    }

    public ArrayList<Item> listarItens() {
        ArrayList<Item> lista = new ArrayList<>();
        abrir();
        Cursor cursor = database.query("itens", null, null, null, null,
                null, "descricao");
        while (cursor.moveToNext()) {
            Item item = new Item();
            item.setId(cursor.getInt(0));
            item.setDescricao(cursor.getString(1));
            item.setQuantidade(cursor.getInt(2));
            item.setComprado(cursor.getInt(3) > 0 ? true : false);
            lista.add(item);
        }
        fechar();
        return lista;
    }
```

```
43
44      public long incluirItem(Item item) {
45          long idDoItemIncluido;
46          ContentValues values = new ContentValues();
47          values.put("descricao", item.getDescricao());
48          values.put("quantidade", item.getQuantidade());
49          values.put("comprado", 0);
50          abrir();
51          idDoItemIncluido = database.insert("itens", null, values);
52          fechar();
53          return idDoItemIncluido;
54      }
55
56      public boolean excluirItem(Item item) {
57          abrir();
58          int registrosExcluidos = database.delete("itens",
59                  "_id = " + item.getId(), null);
60          fechar();
61          if (registrosExcluidos == 0) { return false; }
62          return true;
63      }
64
65      public boolean alterarItem(Item item) {
66          ContentValues values = new ContentValues();
67          values.put("descricao", item.getDescricao());
68          values.put("quantidade", item.getQuantidade());
69          values.put("comprado", item.isComprado() ? 1 : 0);
70          abrir();
71          int registrosAlterados = database.update("itens", values,
72                  "_id = " + item.getId(), null);
73          fechar();
74          if (registrosAlterados == 0) { return false; }
75          return true;
76      }
77  }
```

Dao.java – explicação do código

- **Linhas 12 e 13:** declaramos dois objetos internamente na classe – um para abrir e obter acesso inicial ao banco (**openHelper**) e o outro para realizar as operações de consulta, inclusão, exclusão e alteração nele (**database**).
- **Linhas 15 a 17**: o contrutor da classe **Dao** serve para instanciar o objeto **openHelper**, fazendo isso por meio do construtor de **BancoListaOpenHelper**. A este construtor são passados quatro parâmetros:
  - **context**: o contexto atual de execução, que foi enviado como parâmetro ao construtor de Dao;
  - **lista.sql**: uma String que representa o nome do arquivo de banco de dados
  - **null**: nulo pois não usaremos uma fábrica de cursores
  - **1**: o número da versão do banco de dados.

Dao.java – explicação do código

- **Linha 19:** definem um método chamado **`abrir()`**. Sua função é chamar o método **`getWritableDatabase`** de **`openHelper`**. Este método retorna um objeto **`SQLiteDatabase`**, que é armazenado na variável **`database`**.
- **Linhas 23-25:** definem um método denominado **`fechar()`**. Este método (caso exista um objeto banco válido) chama **`close()`** em **`database`**, fazendo o mesmo ser fechado.
- **Linhas 27-42**: o método **`listarItens()`** faz uma consulta a todos os itens existentes no banco de dados e retorna os mesmos em uma lista (**ArrayList**) de objetos da classe **Item**. A consulta ao banco é feita com o método **`query()`**, do objeto **database**.

Dao.java – explicação do código

- **Linha 27 a 42 (continuação):** O método **`query()`** (linhas 30 e 31) retorna um objeto da classe **Cursor**, que será usado para percorrer o resultado. Os parâmetros especificam como a consulta será feita:
  1. **"itens":** nome da tabela a ser consultada
  2. **null:** quais campos serão retornados; null retorna todos
  3. **null:** condição para a consulta (WHERE); null indica que não haverá condição – todos os registros serão retornados
  4. **null:** argumentos para a condição
  5. **null:** agrupamento (GROUP BY)
  6. **null:** condição para o agrupamento (HAVING)
  7. **"descricao":** campo de ordenação para o resultado

Dao.java – explicação do código

- **Linha 27 a 42 (continuação):** Para criar objetos equivalentes aos registros da tabela, percorremos estes últimos usando o método **`moveToNext()`** do objeto **cursor** dentro de um laço **while** (linhas 32 a 39). Pegamos os valores nos campos do registro apontado pelo cursor por meio de métodos **`get`** específicos para os tipos de cada campo. Deve ser passado ao método **`get`** um valor numérico correspondente ao campo desejado (0 para o primeiro, 1 para o segundo e assim sucessivamente).
- Após colocar cada no valor nos respectivos atributos do objeto **item**, este é adicionado ao **ArrayList** (linha 38).
- Finalizado o laço, fechamos o banco de dados e retornamos o **ArrayList** com os objetos.

Dao.java – explicação do código

- **Linhas 44 a 54:** o método **`incluirItem()`** recebe um objeto Item como parâmetro e inclui os seus dados na tabela.
  - **Linha 46:** cria um objeto da classe **`ContentValues`** chamado **`values`**. Objetos desta classe são usados para mapear uma coluna da tabela do banco de dados para um valor.
  - **Linha 47 a 49:** o método **`put`** de **`values`** recebe dois parâmetros: uma **`String`** com o nome da coluna (o "campo") na tabela do banco de dados e o respectivo valor a ser colocado nela. Se tivéssemos mais campos na nossa tabela de produtos, colocaríamos uma linha desta para cada um deles. Não precisamos colocar valor no campo "_id" da tabela, pois este foi configurado como auto incrementado.

Dao.java – explicação do código

- **Linhas 44 a 54 (continuação)**
  - **Linha 50:** chama o método local que abre o banco de dados.
  - **Linha 51:** executa o método **`insert`** do banco, passando como parâmetros:
    - o nome da tabela onde os dados vão ser inseridos (“**`itens`**”)
    - **`null`**: usada para especificar uma coluna que ficará vazia (nenhuma nesta caso)
    - o objeto da classe **`ContentValues`** com os valores a serem inseridos (**values**).
  - **Linha 52:** fecha o banco de dados.
  - Este método retorna o id do registro que acabou de ser incluído.

Dao.java – explicação do código

- **Linhas 56-63:** remove um produto do banco de dados, usando o método **`delete`**. Passamos para este método os seguintes parâmetros:
  - **`"itens"`**: o nome da tabela
  - **`"_id = " + id`**: a condição (cláusula WHERE) para exclusão – o produto deve possuir o código (id) especificado.
  - **null**: não utilizamos, pois concatenamos o valor do parâmetro antes

Dao.java – explicação do código

- **Linhas 65-76:** definem o método **`alterarItem`**, que recebe como parâmetro um objeto da classe **Item** com seus novos dados (mas com o mesmo id de antes).
  - **Linha 66:** Cria um objeto chamado **`values`**, da classe **ContentValues** para armazenar os valores modificados.
  - **Linha 67 a 69:** Armazena os novos valores para os campos (exceto id, pois é a chave primária e não deve ser modificado).
  - **Linha 70:** Abre o banco de dados
  - **Linhas 71 e 72:** executa o método **`update`** de banco, que recebe como parâmetros:
    - **`"itens"`:** o nome da tabela em questão
    - **`values`**: o objeto **`ContentValues`** com os novos valores
    - **`"_id= " + item.getId()`**: a condição de modificação – só será mudado o itemcujo código for igual ao código (id) recebido como parâmetro deste método (funciona como o WHERE da linguagem SQL)
    - **`null`**: parâmetros da condição – não usado neste caso;
  - **Linha 73**: fecha o banco de dados

Usando a classe Dao

- Depois de implementada a classe **Dao** é muito simples adaptar o código para utilizar o banco de dados.
- Veja como deverá ficar sua **MainActivity** nos próximos slides. As linhas a serem acrescentadas estão em destaque.

```
1   package com.example.alunos.listadecompras;
2
3   import ...
16
17  public class MainActivity extends ActionBarActivity {
18
19      private ArrayList<Item> lista;
20      private ArrayAdapter<Item> adapter;
21      private EditText edtDesc, edtQtd;
22
23      private Dao dao;
24
25      @Override
26      protected void onCreate(Bundle savedInstanceState) {
27          super.onCreate(savedInstanceState);
28          setContentView(R.layout.activity_main);
29
30          dao = new Dao(this);
31          lista = dao.listarItens();
32
33          adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_checked, lista);
34
35          edtDesc = (EditText)findViewById(R.id.edtDesc);
36          edtQtd = (EditText)findViewById(R.id.edtQtd);
37
38          final ListView listView = (ListView)findViewById(R.id.listView);
39          listView.setAdapter(adapter);
40          listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
41
```

Usando a classe Dao (parte 2)

```
42        listView.setOnItemClickListener((parent, view, position, id) → {
45                Item itemTocado = lista.get(position);
46                itemTocado.setComprado(!itemTocado.isComprado());
47                String verbo = "devolveu";
48                if (itemTocado.isComprado()) {
49                    verbo = "comprou";
50                }
51                Toast.makeText(MainActivity.this,
52                        "Você " + verbo + " " + lista.get(position).getDescricao(),
53                        Toast.LENGTH_SHORT)
54                        .show();
55
56                dao.alterarItem(itemTocado);
57
58                adapter.notifyDataSetChanged();
59        });
61
62        listView.setOnItemLongClickListener((adapterView, view, position, id) → {
65                Item itemTocado = lista.get(position);
66
67                dao.excluirItem(itemTocado);
68
69                lista.remove(position);
70                adapter.notifyDataSetChanged();
71                return true;
72        });
74    }
```

Usando a classe Dao (parte 3)

```
76      public void adicionar(View view){
77          Item item = new Item();
78          item.setDescricao(edtDesc.getText().toString());
79          item.setQuantidade(Integer.parseInt(edtQtd.getText().toString()));
80
81          lista.add(item);
82
83          dao.incluirItem(item);
84
85          Collections.sort(lista);
86
87          adapter.notifyDataSetChanged();
88          edtDesc.setText("");
89          edtQtd.setText("");
90          edtDesc.requestFocus();
91      }
```

BIBLIOGRAFIA

- QUERINO FILHO, L. C. Desenvolvendo seu Primeiro Aplicativo Android. Novatec Editora. 2013.
- DEITEL, H. et al. Android for Programmers: An App-Driven Approach. 2012. Pearson Education.