Usar persistência de dados no Android com SQLite.

Para criar uma persistência no Android precisamos criar em primeiro ligar um objeto para carregar nossos dados, ou seja um dto.

DTO, um objeto simples usado para transferir dados de um local a outro na aplicação, sem lógica de negócios em seus objetos e comumente associado à transferência de dados entre uma camada de visão (view layer) e outra de persistência dos dados (model layer).

Leia mais em:

Diferença entre os patterns PO, POJO, BO, DTO e VO:
http://www.devmedia.com.br/diferenca-entre-os-patterns-po-pojo-bo-dto-e-vo/28162#ixzz3bX0Pvcqv

Vamos criar nosso objeto, com o nome de:

Contato.java

```
public class Contato {
   int _id;
   String nome;
   String telefone;
   // construtor vazio
   public Contato(){
   }
   public Contato(int id, String nome, String telefone){
      this._id = id;
      this.nome = nome;
      this.telefone = telefone;
   }
   public Contato(String nome, String telefone){
      this.nome = nome;
      this.telefone = telefone;
   }
```

```
  // gets e sets

 }
```

Para guardar objetos precisamos criar e configurar o banco do SQLite no android. O Android já nos da uma classe de ajuda que é a SQLiteOpenHelper.

Essa classe tem vários métodos de ajuda para manipulação do banco de dados:

http://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html

Para criamos nossas manipulações têm que criar uma classe para criar o banco as tabelas e suportar a leitura dos nossos objetos:

Vamos criar nossa classe:

Com o botão direto nos pacotes do seu projeto, crie uma classe chamada DatabaseHandler, logo após estenda sua classe do SQLiteOpenHelper.

Sua classe ficará assim:

```
public class DatabaseHandler extends SQLiteOpenHelper {

}
```

Depois de extender a classe SQLiteOpenHelper, precisamos reescrever dois métodos obrigatórios para criação do banco, os métodos, onCreate() and onUpgrage().

- onCreate() – Este método é necessário para criar, escrever nas tabelas e gerenciar suas conexões. O metodo é chamado quando a tabela é criada.
- onUpgrade() – Este metodo é chamado quando o database é atualizado, quando a estrutura da tabela é atualizada, adicionado contraints e outras atualizações nesse banco de dados.

Exemplo da classe pronta:

```
public class DatabaseHandler extends SQLiteOpenHelper {

   // Para criar o banco precisamos definir uma versão.

   private static final int DATABASE_VERSION = 1;

   // Nome do banco

   private static final String DATABASE_NAME = "controlecontatos.db";

   // definição do nome da tabela.

   private static final String TABLE_CONTATO = "tbl_contato";
```

```
// Nome das colunas para ser criadas;
private static final String KEY_ID = "id";
private static final String KEY_NOME = "nome";
private static final String KEY_TELEFONE = "phone_number";

public DatabaseHandler(Context context) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
}

// criação das tabelas.
@Override
public void onCreate(SQLiteDatabase db) {
   String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTATO + "("
         + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NOME + " TEXT,"
         + KEY_TELEFONE + " TEXT" + ")";
   db.execSQL(CREATE_CONTACTS_TABLE);
}

// update do banco
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   // deleta a tabela se ela exitir.
   db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTATO);

   // recria a tabela depois de deletar.
   onCreate(db);
```

```
    }
}
```

Agora precisamos criar nosso DAo:

DAO - Data Access Object. É basicamente um o objeto que abstrai tudo que for referente ao acesso a dados da aplicação que serão persistidas.

Por exemplo, é onde ficam o acesso ao banco e métodos de CRUD em uma aplicação que use Banco de Dados.

CRUD é um acrônimo. E a representação para as operações de criação, leitura, alteração e remoção.

Para criação da nossa classe de DAO, precisamos escrever métodos que lidam com operações de gravação, leitura, atualização de deleção:

Agora vamos criar uma classe chamada de contatoDAO, com os seguintes métodos:

```
// adiciona novo contato
public void addContato(Contato contato) {}

// paga um simples contato por id
public Contato getContato(int id) {}

// pega todos os contatos
public List<Contato> getAllContatos() {}

// pega o numero de contatos
public int getContatosCount() {}
```

```
// atualizando o contato

public int updateContato(Contato contato) {}


// deletando o contato

public void deleteContato(Contato contato) {}
```

Insirindo novos contatos:

O metodo addContato() aceita os valores do objeto contato como parametros.

Você precisa criar um ContentValues com parametros com um conjuto de par chaves e valores, usand o objeto contato.

Após, quando nós inserimos dados na database precisamos fechar a operação após essa inserção.

```
addContato()

   // Adicionar o contato


public void addContato(Contato contato) {

   SQLiteDatabase db = this.getWritableDatabase(); //abre o banco para leitura e gravação


   ContentValues values = new ContentValues(); //cria o objeto para preencher valores nas tabelas

   values.put(KEY_NOME, contatoo.getNome()); // Contato Nome

   values.put(KEY_TELEFONE, contato.getTelefone()); // Contato Telefone


   // Inserindo a linha

   db.insert(TABLE_CONTATO, null, values);

   db.close(); // Fechando a conexão do banco.

}
```

Pega todos os contatos:

getAllContatos()

Este método retornara todos os contatos inseridos no banco de dados, retornado um array. Você precisa escrever os dados no objeto e inserir na lista a cada interação do loop.

```
public List<Contato> getAllContatos() {
  List<Contato> contatoList = new ArrayList<Contato>();

String selectQuery = "SELECT  * FROM " + TABLE_CONTATO;

  SQLiteDatabase db = this.getWritableDatabase(); //abre o banco para leitura e gravação
  Cursor cursor = db.rawQuery(selectQuery, null);

  // Pega os vales resultantes e seta no objeto, movendo sempre para o próximo cursor.
  if (cursor.moveToFirst()) {
     do {
        Contato contato = new Contato();
        contato.setID(Integer.parseInt(cursor.getString(0)));
        contato.setNome(cursor.getString(1));
        contato.setPhoneNumber(cursor.getString(2));
        // adiciona o contato na lista de contatos.
        contatoList.add(contato);
     } while (cursor.moveToNext());
  }

  // retorna a lista de contato
```

```
    return contatoList;
}
```

getContatosCount() esse método retorna a quantidade total de contatos inseridos nos bancos.

```
getContatosCount()
            //pega soma total dos contatos
    public int getContatosCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTATO;
        SQLiteDatabase db = this.getReadableDatabase(); //abre o banco para leitura
        Cursor cursor = db.rawQuery(countQuery, null); //passa a query para ser exetutada
        cursor.close();

        // retorna o soma total dos contatos.
                return cursor.getCount();
    }
```

Atualizando registros:

updateContato(), faz atualização de um registro passando um parâmetro único como Id os dados atualizados.

```
updateContato()
    // atualiza um contato simples
public int updateContato(Contato contato) {
    SQLiteDatabase db = this.getWritableDatabase(); //abre o banco para leitura e gravação

    ContentValues values = new ContentValues();
    values.put(KEY_NOME, contato.getNome());
    values.put(KEY_TELEFONE, contato.getPhoneNumber());
```

```
    // faz update no registo
    return db.update(TABLE_CONTATO, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contato.getID()) });
}
```

?Deleting Record

deleteContato() will delete single contato from database.

```
deleteContato()
    // Deletando o retrito por ID
public void deleteContato(Contato contato) {
    SQLiteDatabase db = this.getWritableDatabase();  //abre o banco para leitura e gravação
    db.delete(TABLE_CONTATO, KEY_ID + " = ?",
            new String[] { String.valueOf(contato.getID()) });
    db.close();
}
```

A classe completa, DatabaseHandler.java

```
package default;

import java.util.ArrayList;
import java.util.List;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
```

```
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHandler extends SQLiteOpenHelper {

    // Todas as variáveis
    // Versão do Database
    private static final int DATABASE_VERSION = 1;

    // Nome do banco
    private static final String DATABASE_NAME = "contatosManager";

    // tabela de contatos
    private static final String TABLE_CONTATO = "contatos";

    // colunas da tabela de contatos.
    private static final String KEY_ID = "id";
    private static final String KEY_NOME = "name";
    private static final String KEY_TELEFONE = "phone_number";

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Criando as tabelas
    @Override
    public void onCreate(SQLiteDatabase db) {
```

```
    String CREATE_CONTACTS_TABLE = "CREATE TABLE " + TABLE_CONTATO + "("
            + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NOME + " TEXT,"
            + KEY_TELEFONE + " TEXT" + ")";
    db.execSQL(CREATE_CONTACTS_TABLE);
}

// Atualizando as tabelas
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // Deletando a tabela se ela existe
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTATO);

    // Recriando a nova tabela
    onCreate(db);
}

/**
 * Todos operações de CRUD(Create, Read, Update, Delete).
 */

// Adding new contato
void addContato(Contato contato) {
    SQLiteDatabase db = this.getWritableDatabase();  //abre o banco para leitura e gravação

    ContentValues values = new ContentValues(); // cria o objeto para criar as querys
    values.put(KEY_NOME, contato.getNome()); // Contato Nome
    values.put(KEY_TELEFONE, contato.getPhoneNumber()); // Contato Phone
```

```
    // Inserindo dados
    db.insert(TABLE_CONTATO, null, values);
    db.close(); // fechando a conexao dos dados
}

// Pegando um simples contat
Contato getContato(int id) {
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.query(TABLE_CONTATO, new String[] { KEY_ID,
            KEY_NOME, KEY_TELEFONE }, KEY_ID + "=?",
            new String[] { String.valueOf(id) }, null, null, null, null);
    if (cursor != null)
        cursor.moveToFirst();

    Contato contato = new Contato(Integer.parseInt(cursor.getString(0)),
            cursor.getString(1), cursor.getString(2));
    return contato;
}

// Retornando todos os contatos
public List<Contato> getAllContatos() {
    List<Contato> contatoList = new ArrayList<Contato>();
    // Query para pegar todos sos contatos
    String selectQuery = "SELECT  * FROM " + TABLE_CONTATO;
```

```
    SQLiteDatabase db = this.getWritableDatabase();
    Cursor cursor = db.rawQuery(selectQuery, null);

    if (cursor.moveToFirst()) {
        do {
            Contato contato = new Contato();
            contato.setID(Integer.parseInt(cursor.getString(0)));
            contato.setNome(cursor.getString(1));
            contato.setPhoneNumber(cursor.getString(2));

                                        contatoList.add(contato);
        } while (cursor.moveToNext());
    }

    return contatoList;
}

// Atualizando contatos
public int updateContato(Contato contato) {
    SQLiteDatabase db = this.getWritableDatabase();

    ContentValues values = new ContentValues();
    values.put(KEY_NOME, contato.getNome());
    values.put(KEY_TELEFONE, contato.getPhoneNumber());

    return db.update(TABLE_CONTATO, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contato.getID()) });
```

```
    }

    // Deletando contatos
    public void deleteContato(Contato contato) {
        SQLiteDatabase db = this.getWritableDatabase();
        db.delete(TABLE_CONTATO, KEY_ID + " = ?",
                new String[] { String.valueOf(contato.getID()) });
        db.close();
    }


    // pega soma total de registros
    public int getContatosCount() {
        String countQuery = "SELECT  * FROM " + TABLE_CONTATO;
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery(countQuery, null);
        cursor.close();

        return cursor.getCount();
    }

}
```

Como usar, vamos criar uma Activity como as gerenciamento de persistência do banco de dados.

AndroidSQLiteTutorialActivity

```
package com.androidhive.androidsqlite;

import java.util.List;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;

public class AndroidSQLiteTutorialActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        DatabaseHandler db = new DatabaseHandler(this);

        /**
         * CRUD Operations
         * */
        // Inserindo Contatos
        Log.d("Insert: ", "Inserindo ..");
        db.addContato(new Contato("Ravi", "9100000000"));
        db.addContato(new Contato("Srinivas", "9199999999"));
        db.addContato(new Contato("Tommy", "9522222222"));
        db.addContato(new Contato("Karthik", "9533333333"));
```

```
        // Reading all contatos

        Log.d("Reading: ", "Reading all contatos..");

        List<Contato> contatos = db.getAllContatos();



        for (Contato cn : contatos) {

            String log = "Id: "+cn.getID()+" ,Nome: " + cn.getNome() + " ,Phone: " +
cn.getPhoneNumber();

                // Writing Contatos to log

        Log.d("Nome: ", log);

    }

    }

}
```