# Sumário

# 1. Informativo

Autor(a): Helena Strada

Data de criação: jan/2018

## 2. Pré-Requisitos

Java: Orientação a Objetos, APIs e Bibliotecas;

Android: Activity, Intent, Ciclo de Vida, Views, ListView/RecyclerView, EditText;

Recomendação: Básico de SQL.

# 3. Objetivo

Mostrar as funcionalidades básicas do SQLite e os primeiros passos para criar tabelas e estruturas SQL.

# 4. Versão

## 4.1. minSdkVersion

15

## 4.2. targetSdkVersion

26

# 5. SQLite

SQLite é uma biblioteca de software que implementa um Banco de Dados SQL transacional autônomo, sem servidor e sem nenhuma configuração.

## 5.1. Introdução

O SQLite possui as características de um banco de dados relacional. O SQLite armazena as tabelas, views e índices em apenas um arquivo do disco, e é nesse arquivo que a leitura e a escrita serão realizadas. Com o SQLite, se você reiniciar o celular, ou a aplicação, não iremos perder as informações que foram previamente inseridas. Essa informação só será removida do celular se a aplicação for desinstalada.

Porém, por ser um pouco mais simples do que bancos de dados como o MySQL ou SQL Server, ele possui algumas limitações. Por exemplo, os tipos de dados podem ser somente do tipo *Text, Integer, Real, Blob e Null.* Além disso, ele não valida se o tipo de dado que está sendo inserido, corresponde ao tipo de dado da coluna.

## 5.2. SQLite no Android

O SQLite é incorporado em todos os dispositivos Android. Como dito anteriormente, não precisamos de nenhuma configuração a mais.

### 5.2.1. Armazenamento

Por padrão, nosso arquivo de banco de dados será salvo no diretório:

DATA/data/<Nome-Aplicacao>/databases/<Nome-BD>

DATA: caminho que o método Environment.getDataDirectory() retorna;

Nome-Aplicacao: é o nome do seu aplicativo;

Nome-BD: nome especificado para o seu banco de dados.

# 6. Banco de Dados

Um banco de dados é uma maneira estrutura de armazenar dados de forma persistente em formato de tabelas.

A tabela é composta por colunas e linhas. As colunas representam os tipos de dados que serão inseridos e as linhas representam os dados em si que foram armazenados. Pensamos em uma tabela do excel.

| int | String | string |
|---|---|---|
| ID | Nome | Fabricante |
| 1 | God of War | Sony |

Se fizermos a comparação para a nossa programação orientada a objetos, cada coluna irá representar o nosso atributo da classe e cada registro representaria uma instância específica deste objeto.

```
Public class Jogo {

        private Long id;

        private String nome;

        private String fabricante;

        // getters e setters

}

Jogo jogo = new Jogo();

jogo.setId(2);

jogo.setNome("GTA");

jogo.setFabricante("Saint Monica Studios");
```

Para o nosso banco de dados, o registro que seria adicionado a nossa base, seria o registro deste jogo que criamos com suas respectivas propriedades.

| ID | Nome | Autor |
|---|---|---|
| 1 | God of War | Sony |
| 2 | GTA | Saint Monica Studios |

A grande situação aqui é que para manipularmos do Java ou de outra linguagem de programação para o nosso banco de dados, nós temos comandos específicos para o banco de dados. Ou seja, se precisamos inserir um novo registro no banco de dados por exemplo, nós temos um comando específico para essa ação.

Eu quero *inserir* uma nova linha na tabela de *jogos* com os respectivos dados de *id* = 3, *nome* = "FIFA", *fabricante* = "EA".

No SQL:

Insert into *jogos* (id, nome, fabricante) values (3, ‘FIFA’, ‘EA’);

INSERT: para inserir um novo registro no bd;

UPDATE: para atualizar um registro já existente no bd;

SELECT: para selecionar um registro no bd;

DELETE: para deletar um registro existente no bd.

Para que estes sejam dados sejam manipulados, precisamos criar as nossas tabelas com as respectivas colunas e os tipos de dados que queremos armazenar. Para isto, temos comandos específicos também do SQL. Neste caso, utilizaremos o CREATE.

# 7. SQL na Prática

Salvar nossos dados em um banco de dados é ideal para dados estruturados e para uma grande quantidade de informações que deverão ser armazenadas.

## 7.1. Esquema e Contrato

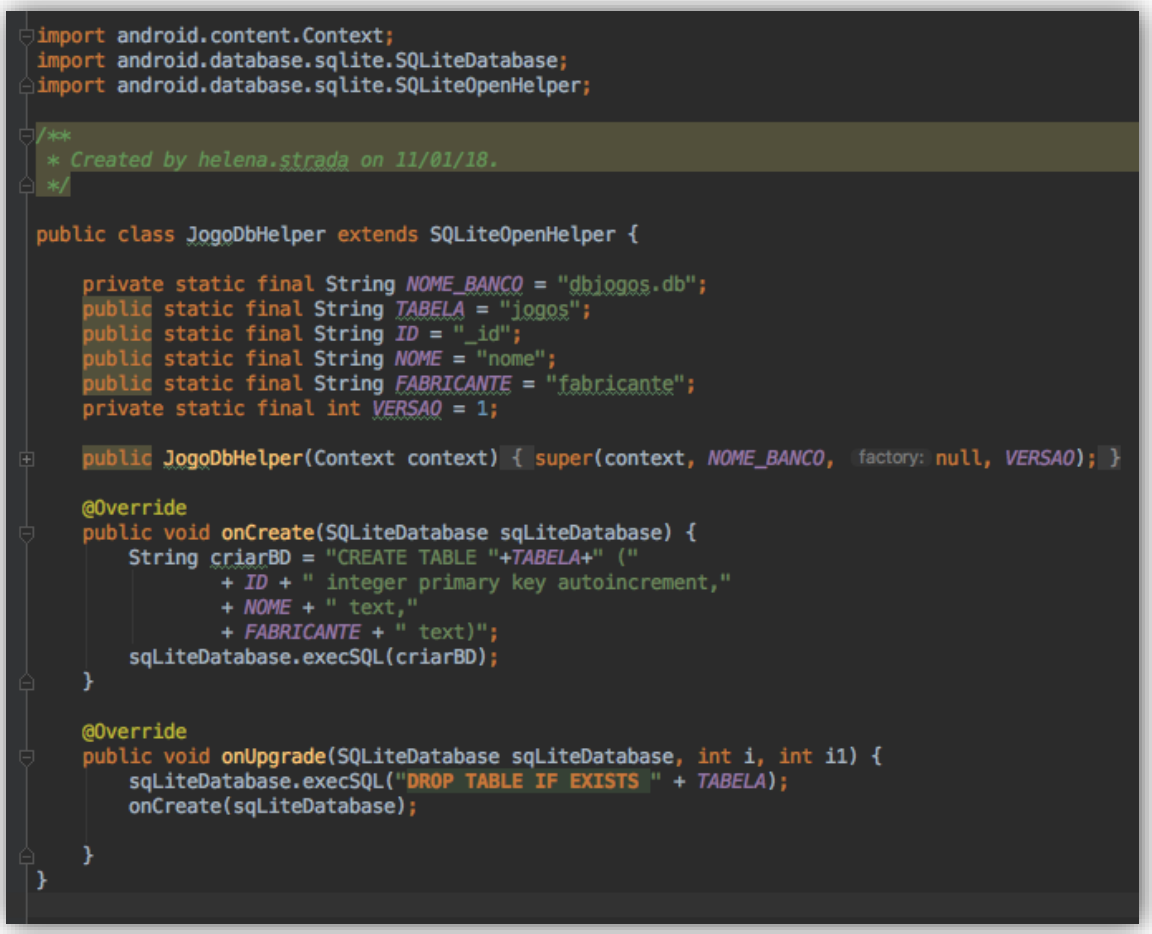
Um dos princípios mais importante de bancos de dados é o esquema: é a declaração de como o nosso banco de dados será organizado. Além disso, é recomendado criar um classe de contrato que irá especificar a estrutura do esquema.

Essa classe de contrato é aonde colocaremos as definições globais para o banco de dados, como por exemplo, nossas propriedades para o nosso Jogo.

## 7.2. Criando o Banco de Dados

Uma vez que definimos a estrutura das nossas tabelas, podemos de fato criá-las. Para criar nosso banco, vamos utilizar um conjunto útil de APIs que está disponível para nós na classe SQLiteOpenHelper.

Assim, podemos chamar somente quando necessitamos das informações e não durante a inicialização do aplicativo. Utilizaremos o getWritableDatabase() e o getReadableDatabase(). Além disso, devemos ter em mente que as chamadas podem ser de longas duração, devemos assim utilizar AsyncTask, por exemplo.



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

/**
 * Created by helena.strada on 11/01/18.
 */

public class JogoDbHelper extends SQLiteOpenHelper {

    private static final String NOME_BANCO = "dbjogos.db";
    public static final String TABELA = "jogos";
    public static final String ID = "_id";
    public static final String NOME = "nome";
    public static final String FABRICANTE = "fabricante";
    private static final int VERSAO = 1;

    public JogoDbHelper(Context context) { super(context, NOME_BANCO,  factory: null, VERSAO); }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        String criarBD = "CREATE TABLE "+TABELA+" ("
                + ID + " integer primary key autoincrement,"
                + NOME + " text,"
                + FABRICANTE + " text)";
        sqLiteDatabase.execSQL(criarBD);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABELA);
        onCreate(sqLiteDatabase);

    }
}
```

*Figura 1 – Definindo a estrutura da nossa tabela.*

Para nós criarmos um novo banco de dados em nossa aplicação, a primeira coisa que devemos fazer é criar uma classe que herde de SQLiteOpenHelper para que nós tenhamos acesso a dois métodos principais, OnCreate e OnUpgrade.

OnCreate nós definimos qual o script de criação das nossas tabelas. OnUpgrade nós podemos atualizar uma versão do nosso banco de dados (atualizando estrutura, alterando versão).

Nesta classe, definimos também o construtor padrão. Ela irá definir em qual banco de dados queremos realizar todo o nosso script.

## 7.3. Manipulando os Dados

Vamos aprender agora a como manipular os dados em nosso SQLite. Existem algumas maneiras de escrevermos os nossos scripts. Uma é utilizando comandos nativos do próprio SQL e o outro é através de métodos específicos do SQLiteDatabase.

### 7.3.1. Insert

Podemos notar essa diferença com o insert, por exemplo. Queremos inserir um novo registro em nosso banco de dados. Recebemos os valores de um novo objeto e queremos armazenar essa informação no banco de dados.

```
long resultado;
ContentValues values = new ContentValues();
values.put(JogoDbHelper.NOME, jogo.getNome());
values.put(JogoDbHelper.FABRICANTE, jogo.getFabricante());
resultado = db.insert(JogoDbHelper.TABELA,
        nullColumnHack: null,
        values);
if (resultado == -1) {
    Log.d( tag: "Erro ao inserir",  msg: "Erro");
    return "Erro ao inserir registro";
} else {
    Log.d( tag: "Sucesso ao inserir",  msg: "Sucesso");
    return "Registro inserido com sucesso";
}
```

*Figura 2 – Inserindo um novo registro através dos comandos do próprio SQLite.*

```
String inserir = "insert into "
        + JogoDbHelper.TABELA
        + " (nome, fabricante) values (?, ?)";
db.execSQL(inserir, new Object[]{jogo.getNome(), jogo.getFabricante()});
db.close();
```

*Figura 3 – Inserindo o mesmo registro, porém utilizando os comandos nativos SQL.*

### 7.3.2. Ler

```java
public List<Jogo> getLista() {
    List<Jogo> jogos = new LinkedList<>();
    String rawQuery = "SELECT _id, nome, fabricante FROM " +
            JogoDbHelper.TABELA;
    SQLiteDatabase db = dbo.getReadableDatabase();
    Cursor cursor = db.rawQuery(rawQuery,  selectionArgs: null);
    Jogo jogo = null;
    if (cursor.moveToFirst()) {
        do {
            jogo = new Jogo();
            jogo.setId(cursor.getLong( i: 0));
            jogo.setNome(cursor.getString( i: 1));
            jogo.setFabricante(cursor.getString( i: 2));
            jogos.add(jogo);
        } while (cursor.moveToNext());
    }
    return jogos;
}
```

*Figura 4 – Para buscar todos os registros do banco de dados.*

Para buscarmos todos os registros do banco de dados e retornarmos em uma lista, precisamos de um cursor. Para cada registro da nossa lista, ele irá buscar os dados correspondentes desse jogo e adicioná-lo a nossa lista. Para manipularmos a lista em Java.

Um detalhe importante é que para o SQLite, ao realizarmos a consulta ao banco, retorne um resultado para nós, é utilizarmos o *rawQuery.*

### 7.3.3. Atualizar.

```
public void atualizar(Jogo jogo) {

    SQLiteDatabase db = dbo.getWritableDatabase();

    /* long resultado;
    ContentValues values = new ContentValues();
    values.put(JogoDbHelper.NOME, jogo.getNome());
    values.put(JogoDbHelper.FABRICANTE, jogo.getFabricante());
    resultado = db.update(JogoDbHelper.TABELA, values, JogoDbHelper.ID + "=" + jogo.getId(), null);

    if (resultado == -1) {
        Log.d("Erro ao inserir", "Erro");
        return "Erro ao inserir registro";
    } else {
        Log.d("Sucesso ao inserir", "Sucesso");
        return "Registro inserido com sucesso";
    } */

    String update = "update " + JogoDbHelper.TABELA + " set nome = ?, fabricante = ? where _id = ?";
    db.execSQL(update, new Object[]{jogo.getNome(), jogo.getFabricante(), jogo.getId()});
    Log.d( tag: "sql: ", update);
    db.close();
}
```

*Figura 5 – Atualizando um registro no banco de dados*

### 7.3.4. Deletar

```
public void remover(Jogo jogo) {
    SQLiteDatabase db = dbo.getWritableDatabase();

    /* String where = JogoDbHelper.ID + "=" + jogo.getId();
    db.delete(JogoDbHelper.TABELA, where, null); */
    String deletar = "delete from " + JogoDbHelper.TABELA + " where _id = ?";
    db.execSQL(deletar, new Object[]{jogo.getId()});
    db.close();

}
```

*Figura 6 – Deletar um registro do banco de dados.*

Para deletarmos um registro do nosso banco de dados, realizamos com o comando do próprio sql ou nativo do SQLite.

# 8. Projeto

Para este projeto, iremos utilizar uma estrutura já criada com RecyclerView, CardView e todos os métodos para realizar um CRUD, mas salvando e atualizando apenas uma lista em Java. Para este projeto vamos utilizar a base do projeto: "*android-rv-cv-notify-crud*".

Vamos copiar e colar o projeto e alterar o nome para "*android-rv-cv-notify-crud-sqlite*".

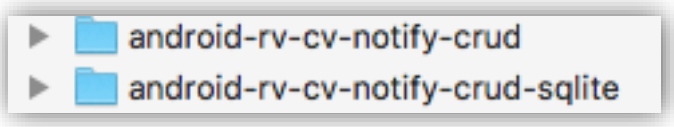



*Figura 7 – Projeto duplicado.*

Não iremos mexer na estrutura e nem no layout deste projeto. Tudo o que precisamos fazer é alterar o local aonde os nossos dados serão armazenados.

A primeira coisa que precisamos fazer é criar uma classe auxiliar que ficará responsável por determinar o nome do nosso banco de dados, e os scripts iniciais para a criação das nossas tabelas.

Essa classe terá o nome de "*JogoDbHelper*"e ficará dentro da pasta de dao.

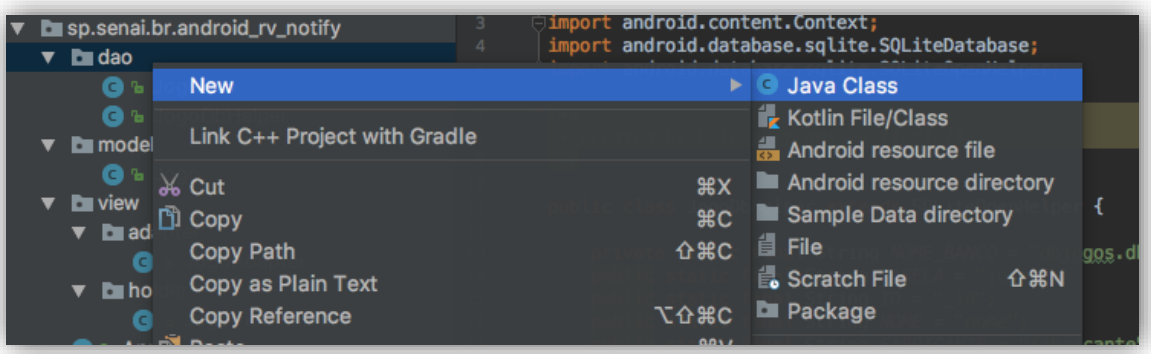



*Figura 8 – Criar uma nova classe dentro de dao chamada JogoDbHelper.*

```java
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

/**
 * Created by helena.strada on 11/01/18.
 */

public class JogoDbHelper extends SQLiteOpenHelper {

    // propriedades para o nosso banco de dados
    private static final String NOME_BANCO = "dbjogos.db";
    public static final String TABELA = "jogos";
    public static final String ID = "_id";
    public static final String NOME = "nome";
    public static final String FABRICANTE = "fabricante";
    private static final int VERSAO = 1;

    // definimos o nome do nosso banco e qual script queremos executar no início da nossa aplicação
    public JogoDbHelper(Context context) { super(context, NOME_BANCO,  factory: null, VERSAO); }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        // script para a criação da nossa tabela
        String criarBD = "CREATE TABLE "+TABELA+" ("
                + ID + " integer primary key autoincrement,"
                + NOME + " text,"
                + FABRICANTE + " text)";
        sqLiteDatabase.execSQL(criarBD);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int i, int i1) {
        // script para atualização
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS " + TABELA);
        onCreate(sqLiteDatabase);

    }
}
```

*Figura 9 – Nossa classe auxiliar para realizar a criação do banco de dados.*

O nosso Dao ainda permanecerá com os métodos que foram criados, porém, nós trocaremos o local aonde os dados serão armazenados.

```java
public class JogoDao {

    private SQLiteDatabase db;
    private JogoDbHelper dbo;

    public JogoDao (Context context) {
        dbo = new JogoDbHelper(context);
    }

    public void salvar(Jogo jogo) {

        SQLiteDatabase db = dbo.getWritableDatabase();

        /* long resultado;
        ContentValues values = new ContentValues();
        values.put(JogoDbHelper.NOME, jogo.getNome());
        values.put(JogoDbHelper.FABRICANTE, jogo.getFabricante());
        resultado = db.insert(JogoDbHelper.TABELA,
                null,
```

```
                values);
        if (resultado == -1) {
            Log.d("Erro ao inserir", "Erro");
            return "Erro ao inserir registro";
        } else {
            Log.d("Sucesso ao inserir", "Sucesso");
            return "Registro inserido com sucesso";
        } */

        String inserir = "insert into "
                + JogoDbHelper.TABELA
                + " (nome, fabricante) values (?, ?)";
        db.execSQL(inserir, new Object[]{jogo.getNome(), jogo.getFabricante()});
        db.close();

    }

    public List<Jogo> getLista() {
        List<Jogo> jogos = new LinkedList<>();
        String rawQuery = "SELECT _id, nome, fabricante FROM " +
                JogoDbHelper.TABELA;
        SQLiteDatabase db = dbo.getReadableDatabase();
        Cursor cursor = db.rawQuery(rawQuery, null);
        Jogo jogo = null;
        if (cursor.moveToFirst()) {
            do {
                jogo = new Jogo();
                jogo.setId(cursor.getLong(0));
                jogo.setNome(cursor.getString(1));
                jogo.setFabricante(cursor.getString(2));
                jogos.add(jogo);
            } while (cursor.moveToNext());
        }
        return jogos;
    }

    public void atualizar(Jogo jogo) {

        SQLiteDatabase db = dbo.getWritableDatabase();

        /* long resultado;
        ContentValues values = new ContentValues();
        values.put(JogoDbHelper.NOME, jogo.getNome());
        values.put(JogoDbHelper.FABRICANTE, jogo.getFabricante());
        resultado = db.update(JogoDbHelper.TABELA, values, JogoDbHelper.ID + "=" +
jogo.getId(), null);

        if (resultado == -1) {
            Log.d("Erro ao inserir", "Erro");
            return "Erro ao inserir registro";
        } else {
            Log.d("Sucesso ao inserir", "Sucesso");
            return "Registro inserido com sucesso";
        } */

        String update = "update " + JogoDbHelper.TABELA + " set nome = ?, fabricante = ? where
_id = ?";
        db.execSQL(update, new Object[]{jogo.getNome(), jogo.getFabricante(), jogo.getId()});
        Log.d("sql: ", update);
        db.close();
```

```
    }

    public Jogo localizar(Long id) {
        SQLiteDatabase db = dbo.getWritableDatabase();
        String query = "SELECT _id, nome, fabricante FROM " + JogoDbHelper.TABELA + "
WHERE _id = ?";
        Cursor cursor = db.rawQuery(query, new String[]{String.valueOf(id)});
        cursor.moveToFirst();
        Jogo jogoA = new Jogo();
        jogoA.setId(cursor.getLong(0));
        jogoA.setNome(cursor.getString(1));
        jogoA.setFabricante(cursor.getString(2));
        db.close();
        return jogoA;
    }

    public void remover(Jogo jogo) {
        SQLiteDatabase db = dbo.getWritableDatabase();

        /* String where = JogoDbHelper.ID + "=" + jogo.getId();
        db.delete(JogoDbHelper.TABELA, where, null); */
        String deletar = "delete from " + JogoDbHelper.TABELA + " where _id = ?";
        db.execSQL(deletar, new Object[]{jogo.getId()});
        db.close();

    }

}
```

# 9. Resumo

A ideia principal deste arquivo é mostrar como os comandos básicos do SQL e como utilizar o SQLite no Android. E como informação extra, aprendemos a como utilizar o Stetho para verificar as informações que foram inseridas, bem como o banco que fora criado e as suas respectivas tabelas.

# 10. Referências


http://pythonclub.com.br/guia-rapido-comandos-sqlite3.html

https://www.tutorialspoint.com/sqlite/

http://facebook.github.io/stetho/

https://developer.android.com/training/basics/data-storage/databases.html?hl=pt-br

https://developer.android.com/reference/android/database/sqlite/package-summary.html