O SQLite NO ANDROID

Por que vou estudar o SQLite se eu já tenho muita experiência com MySQL e outros bancos de dados bem mais populares?

Não se engane, pois por motivos de segurança o sistema Android não nos permite realizar conexões diretas com bancos de dados remotos, bancos compartilhados. Bom, o Android não nos fornece APIs para isso, mas é possível, porém de forma alguma recomendado.

Para essa ação, acesso a dados compartilhados em banco de dados remoto, acontecer como esperado pelo sistema Android, nós temos que utilizar alguma API de comunicação com o servidor Web host da base de dados.

Este servidor, junto a uma linguagem de back-end Web, retornará uma resposta à comunicação realizada, resposta entregue à mesma API utilizada na comunicação Android → back-end Web

O **SQLite** é o banco de dados interno e oficial da plataforma Android, com ele é possível modelar uma estrutura de tabelas relacionadas entre si para representar os dados do mundo real.

Se fizermos uma comparação, o **SQLite** é muito parecido com o **MySQL**, porém com algumas limitações por ser um banco de dados muito mais leve e simples.

Neste artigo, vou te mostrar como utilizar o banco de dados **SQLite** no Android de uma forma simples e rápida.

Então, continue lendo este artigo para entender como guardar dados com o SQLite:

O que é o SQLite?

O **SQLite** é um banco de dados de código aberto com recursos de um banco de dados relacional, com sintaxe SQL e transações. Como ele requer memória limitada para ser executado (aproximadamente 250 KByte), ele funciona perfeitamente dentro da plataforma Android.
Por ser um banco de dados simples, os tipos de dados são um **pouco limitados**. Ele suporta os tipos de dados *TEXT*, *INTEGER* e *REAL*. Todos os outros tipos devem ser convertidos em um desses campos antes de serem salvos no banco de dados.

Porém, ele não valida se os tipos adicionados nas colunas são realmente do tipo definido, por exemplo, você pode adicionar um número inteiro em uma coluna de Strings e vice-versa.

Apesar de simples e limitado, ele é um banco de dados leve e rápido, tornando-o perfeito para dispositivos Android.

Ao contrário de outros bancos de dados que ficam em servidores, como Oracle e Microsoft SQL Server, cada banco de dados **SQLite** é armazenado em um único arquivo no disco.

O SQLite no Android

O **SQLite** é incorporado em todos os dispositivos Android, por isso usar-lo no Android não requer nenhuma configuração a mais.

Você só precisa definir as estruturas das tabelas e as operações que serão feitas utilizando os dados armazenados. Posteriormente, o banco de dados é gerenciado automaticamente pela plataforma Android.

Como o acesso a um banco de dados **SQLite** envolve acesso ao sistema de arquivos no disco do Android, isso pode ser lento. Portanto, é recomendável executar operações de banco de dados de forma assíncrona utilizando uma **AsyncTask**.

Se o seu aplicativo criar um banco de dados, este banco de dados é, por padrão, salvo no diretório **DATA/data/APP_NAME/databases/FILENAME**.

As pastas do caminho acima são construídas com base nas seguintes regras:

- **DATA** é o caminho que o método **Environment.getDataDirectory()** retorna.
- **APP_NAME** é o nome do seu aplicativo.
- **FILENAME** é o nome que você especificou no código do seu aplicativo para o banco de dados.

Revisão Sobre Banco de Dados

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

Uma tabela possui colunas com diferentes tipos de dados e cada linha em uma tabela representa uma gravação de dados. Para facilitar, pense em uma tabela como uma planilha do Excel.

Para a programação orientada a objetos, cada tabela em um banco de dados geralmente representa um objeto (representado por uma classe). Cada coluna de tabela representa um atributo de classe. Cada registro em uma tabela representa uma instância específica desse objeto.

Vamos ver um exemplo rápido.

Digamos que você tenha um banco de dados com uma tabela chamada *Empregado* com cinco colunas.

- ID (Integer)
- Nome (String)
- Sobrenome (String)
- Cargo (String)
- Salario (Real)

Você poderia então adicionar um registro à base de dados para um funcionário chamado **João da Silva** e um registro separado para um funcionário chamado **Miguel Veloso**.

As informações dentro de um banco de dados podem manipulados utilizando comandos **SQL**:

- INSERT: Adicionar um novo registro
- UPDATE: Atualizar um resgistro existente
- DELETE: Remover um registro existente

Você pode procurar dados específicos dentro de um banco de dados usando uma consulta **SQL**.

Uma consulta (usando o comando SELECT) pode envolver uma tabela ou múltiplas tabelas. Para criar uma consulta, você deve especificar as tabelas, as colunas e os valores usando comandos **SQL** terminados por um ponto-e-vírgula (;).

SQLite na Prática

Todo aplicativo Android pode criar e fazer uso do bancos de dados **SQLite** para armazenar grandes quantidades de dados estruturados.

Se você tem experiência trabalhando com banco de dados relacionais e está familiarizado com **SQL** e **JDBC**, vai ser muito fácil você utilizar o **SQLite**.

Esquema e Contrato

Um dos conceitos mais importantes de bancos de dados **SQL** é o esquema: basicamente a estrutura de como o banco de dados é organizado.

O esquema é utilizado nas declarações **SQL** e na criação do banco de dados. No Android, é uma boa prática criar uma classe guia, conhecida como **classe de contrato**, que especifica a estrutura do esquema.

Uma **classe de contrato** é onde ficam as constantes que definem os nomes para as URIs, tabelas e colunas. A **classe de contrato** permite usar as mesmas constantes em outras classes no mesmo pacote. Isso permite que você altere o nome da coluna em um local e que a mudança se propague por todo o código.

Uma boa forma de organizar uma **classe de contrato** é colocar as definições que sejam globais para todo o banco de dados no nível raiz da classe.

Por exemplo, este exemplo define os nomes da tabela e das colunas para uma única tabela:

```
public final class PostContract {

private PostContract() {}

public static class PostEntry implements BaseColumns {
public static final String TABLE_NAME = "post";
public static final String COLUMN_NAME_TITLE = "titulo";
public static final String COLUMN_NAME_SUBTITLE = "subtitulo";
}
}
```

Criar o Banco de Dados

Uma vez definida a estrutura do banco de dados, vamos implementar os métodos para criar e alterar as tabelas.

Para criar nosso banco de dados, vamos utilizar as APIs da classe **SQLiteOpenHelper**. Quando você usa essa classe para criar seu banco de dados, o sistema executa as operações de criação e atualização de forma automática quando necessário e não durante a inicialização do aplicativo.

Para usar **SQLiteOpenHelper**, crie uma subclasse e implemente os métodos **onCreate()**, **onUpgrade()** e **onOpen()**. Também é possível implementar **onDowngrade()**, mas não é obrigatório.

```
public class PostDbHelper extends SQLiteOpenHelper {

private static final String TEXT_TYPE = " TEXT";
private static final String COMMA_SEP = ",";
private static final String SQL_CREATE_POSTS =
"CREATE TABLE " + PostEntry.TABLE_NAME + " (" +
PostEntry._ID + " INTEGER PRIMARY KEY," +
PostEntry.COLUMN_NAME_TITLE + TEXT_TYPE + COMMA_SEP +
PostEntry.COLUMN_NAME_SUBTITLE + TEXT_TYPE + " )";

private static final String SQL_DELETE_POSTS =
"DROP TABLE IF EXISTS " + PostEntry.TABLE_NAME;

public static final int DATABASE_VERSION = 1;
public static final String DATABASE_NAME = "FeedReader.db";

public PostDbHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
}
public void onCreate(SQLiteDatabase db) {
db.execSQL(SQL_CREATE_POSTS);
}
```

```
23. public void onUpgrade(SQLiteDatabase db, int oldVersion, int
    newVersion) {
24. db.execSQL(SQL_DELETE_POSTS);
25. onCreate(db);
26. }
27. public void onDowngrade(SQLiteDatabase db, int oldVersion, int
    newVersion) {
28. onUpgrade(db, oldVersion, newVersion);
29. }
30. }
```

Para acessar o banco de dados, instancie a subclasse de **SQLiteOpenHelper**:

```
1. PostDbHelper mDbHelper = new PostDbHelper(getContext());
```

Manipulando os Dados

Agora vamos ver como manipular os dados em nosso SQLite utilizando os métodos do **SQLiteOpenHelper**.

Inserir

Para inserir dados no banco de dados vamos utilizar um objeto **ContentValues** passando para o método **insert()**:

```
1. SQLiteDatabase db = mDbHelper.getWritableDatabase();
2.
3. ContentValues values = new ContentValues();
4. values.put(PostEntry.COLUMN_NAME_TITLE, "Titulo do Post");
5. values.put(PostEntry.COLUMN_NAME_SUBTITLE, "Subtitulo do Post");
6.
7. long newRowId = db.insert(PostEntry.TABLE_NAME, null, values);
```

O primeiro argumento para **insert()** é o nome da tabela.

O segundo argumento indica o que fazer caso o **ContentValues** esteja vazio. Se você especificar **null**, como no exemplo, não será inserido uma linha quando não houver valores.

Ler

Para ler as informações de um banco de dados, use o método **query()** passando os critérios de seleção e as colunas desejadas. Os resultados da consulta são retornados em um objeto **Cursor**.

```
1. SQLiteDatabase db = mDbHelper.getReadableDatabase();
2.
3. String[] projection = {
4. PostEntry._ID,
5. PostEntry.COLUMN_NAME_TITLE,
6. PostEntry.COLUMN_NAME_SUBTITLE
7. };
8.
```

```
String selection = PostEntry.COLUMN_NAME_TITLE + " = ?";
String[] selectionArgs = { "Titulo do Post" };

String sortOrder =
PostEntry.COLUMN_NAME_SUBTITLE + " DESC";

Cursor c = db.query(
PostEntry.TABLE_NAME,
projection,
selection,
selectionArgs,
null,
null,
sortOrder);
```

Para navegar entre os dados, use os métodos do **Cursor**, que sempre deverão ser chamados antes de começar ler os valores.

```
cursor.moveToFirst();
long itemId = cursor.getLong(
cursor.getColumnIndexOrThrow(PostEntry._ID)
);
```

Atualizar

Quando precisar modificar os valores dos dados do seu banco de dados, use o método **update()**.

```
SQLiteDatabase db = mDbHelper.getReadableDatabase();

ContentValues values = new ContentValues();
values.put(PostEntry.COLUMN_NAME_TITLE, "Novo Titulo do Post");

String selection = PostEntry.COLUMN_NAME_TITLE + " LIKE ?";
String[] selectionArgs = { "Titulo do Post" };

int count = db.update(
PostDbHelper.PostEntry.TABLE_NAME,
values,
selection,
selectionArgs);
```

Excluir

Para excluir linhas de uma tabela, utilize os filtros para selecionar as linhas que devem ser excluídas. A API do banco de dados oferece um mecanismo para criar critérios e filtros de seleção.

```
String selection = PostEntry.COLUMN_NAME_TITLE + " LIKE ?";
String[] selectionArgs = { "Titulo do Post" };

```

```
4.  db.delete(PostEntry.TABLE_NAME, selection, selectionArgs);
```