



Persistência de Dados

Parte 2

Prof. Ilo Rivero
(ilo@pucminas.br)

O que vamos aprender nessa aula?

- Vamos aprender a realizar a persistência de dados utilizando um banco de dados, o **SQLite**

Introdução

- Um **banco de dados (BD)** é uma coleção de dados relacionados (ELMASRI e NAVATHE, 2019)
- Um BD representa algum aspecto do mundo real, às vezes chamado de minimundo ou de universo de discurso. As mudanças no minimundo são refletidas no BD
- Um BD é uma coleção logicamente coerente de dados com algum significado inerente

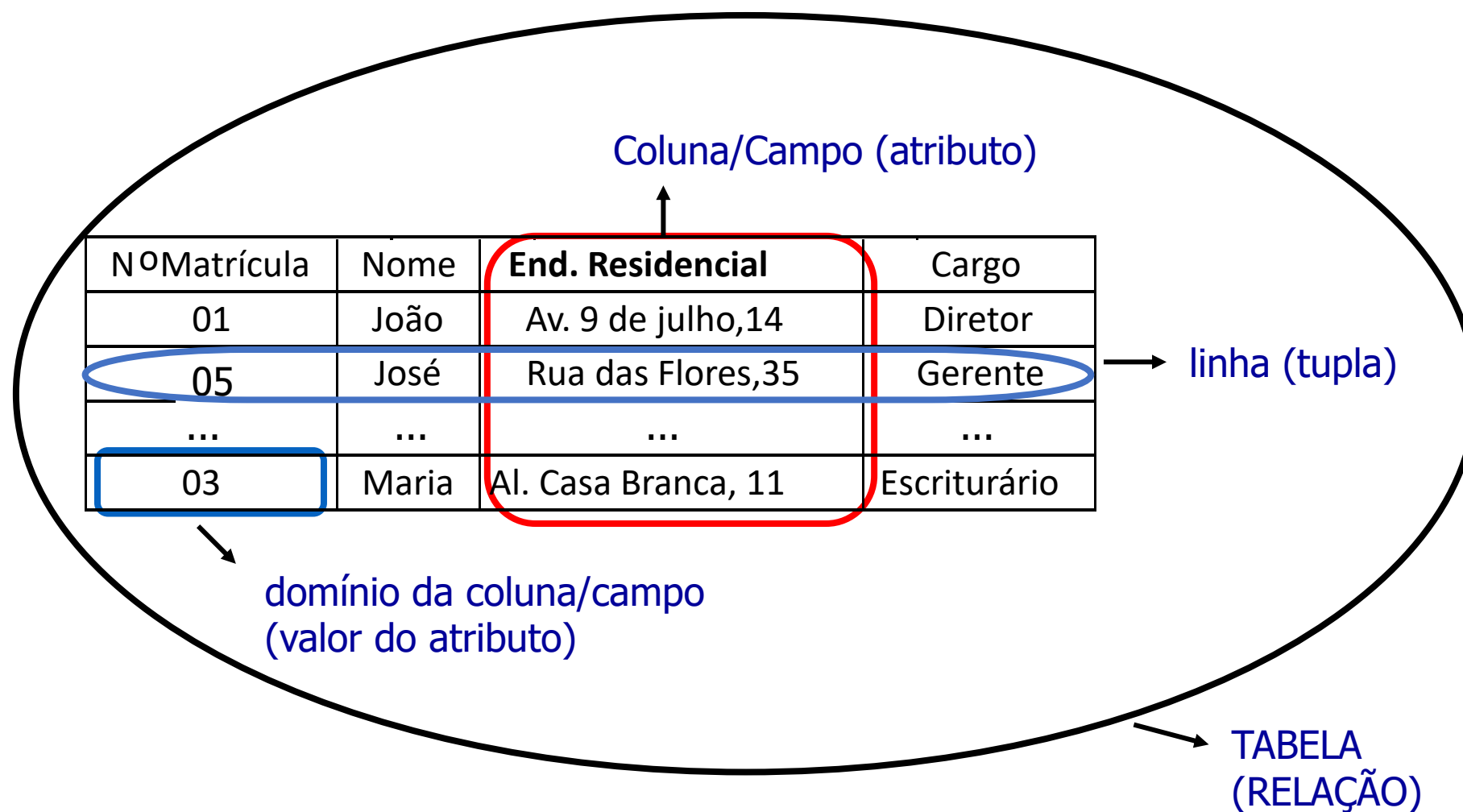
Introdução

- Um BD é projetado, montado e preenchido com dados para uma finalidade específica
- Tem um grupo intencionado de usuários e algumas aplicações previamente concebidas nas quais usuários estão interessados

Modelo Relacional

- As primeiras implementações comerciais do modelo relacional se tornaram disponíveis no início da década de 1980, como o sistema SQL/DS no sistema operacional MVS, da IBM, e o SGBD, da Oracle
- O modelo relacional representa o BD como uma coleção de relações
- Quando uma relação é considerada uma tabela de valores, cada linha na tabela representa uma coleção de valores de dados relacionados

Modelo Relacional



Modelo Relacional

<u>CodEmp</u>	Nome	DataNasc	Cidade	Estado	CodDepto
1	José	21/04/1980	BH	MG	1
2	Alberto	22/04/1980	BH	MG	1
3	Maria	05/08/1970	BH	MG	3
4	Ana	24/04/1980	BH	MG	2
5	Pedro	05/08/1970	BH	MG	1
6	Antônio	06/08/1970	BH	MG	2
7	Maria	07/08/1970	BH	MG	3
8	João	28/04/1980	BH	MG	NULL
9	Carlos	29/04/1980	BH	MG	NULL

Existem dois empregados com o mesmo nome, porém não há problemas, desde que o atributo **Nome** não seja o **Atributo Chave** da relação.

CodEmp é o **Atributo Chave** da relação. Note que seus valores não se repetem. O atributo **CodEmp** garante a unicidade entre as tuplas da relação **Empregado**.

SQL

- Linguagem padrão para lidar com Banco de Dados (BD)
- Structure Query Language (Linguagem Estrutura de Consulta), conhecida com SQL
- Diversos SGBDs (Oracle, SQLServer, MySQL, etc.) que estão no mercado suportam a linguagem SQL
- SQL é uma linguagem não procedural, ou seja, define os resultados desejados mas não a forma como os resultados podem ser alcançados

SQL - CREATE TABLE

- Para criar uma nova tabela é necessário utilizar o comando **CREATE TABLE** (definição de dados)

```
CREATE TABLE Aluno (  
    matricula INTEGER,  
    nome VARCHAR  
);
```

Essa instrução cria uma tabela com duas colunas, *matricula* (chave primária da tabela) e *nome*

SQL - CREATE TABLE

```
CREATE TABLE Aluno (  
    matricula INTEGER PRIMARY KEY AUTOINCREMENT,  
    nome VARCHAR  
);
```

Também podemos definir o campo chave primária da tabela. Este campo é único entre todos os registros da tabela.

SQL - INSERT

- Para popular o banco de dados é necessário o comando **INSERT**

```
INSERT INTO Aluno  
(matricula, nome) VALUES  
(10, 'Maria');
```

Essa instrução adiciona
um linha à tabela Aluno
com o valor 10 na coluna
matricula e o valor 'Maria'
na coluna nome

SQL - DELETE

- Para deletar uma linha o banco de dados é necessário o comando DELETE e o uso da cláusula WHERE para filtrar a linha que será apagada.

```
DELETE FROM Aluno  
WHERE nome = 'Maria';
```

Essa instrução apaga do banco todas as linhas cujo nome é igual a 'Maria'

SQL - UPDATE

- Para atualizar uma linha o banco de dados é necessário o comando UPDATE e o uso da cláusula SET para definir os campos que serão atualizados e WHERE para filtrar a linha que será atualizada.

```
UPDATE Aluno  
SET nome = 'Sandra Maria'  
WHERE matricula = '1';
```

Essa instrução localiza a linha cujo matricula é igual a 1 e atualiza o campo nome para o novo valor "Sandra Maria"

SQL - SELECT

- Para recuperar linhas do banco de dados é necessário o comando **SELECT** para definir os campos que serão recuperados, e o uso da cláusula **FROM** para definir a(s) tabela(s) e **WHERE** para filtrar a(s) linha(s) que será(ão) exibida(s).

```
SELECT matricula, nome  
FROM Aluno  
WHERE matricula = '1';
```

Essa instrução localiza a linha cujo matricula é igual a 1 e projeta (recupera) os campos "matricula" e "nome".

SQLite

- É um banco de dados embutido, open source e escrito na linguagem C
- O SQLite está disponível em todos os dispositivos Android e IOS



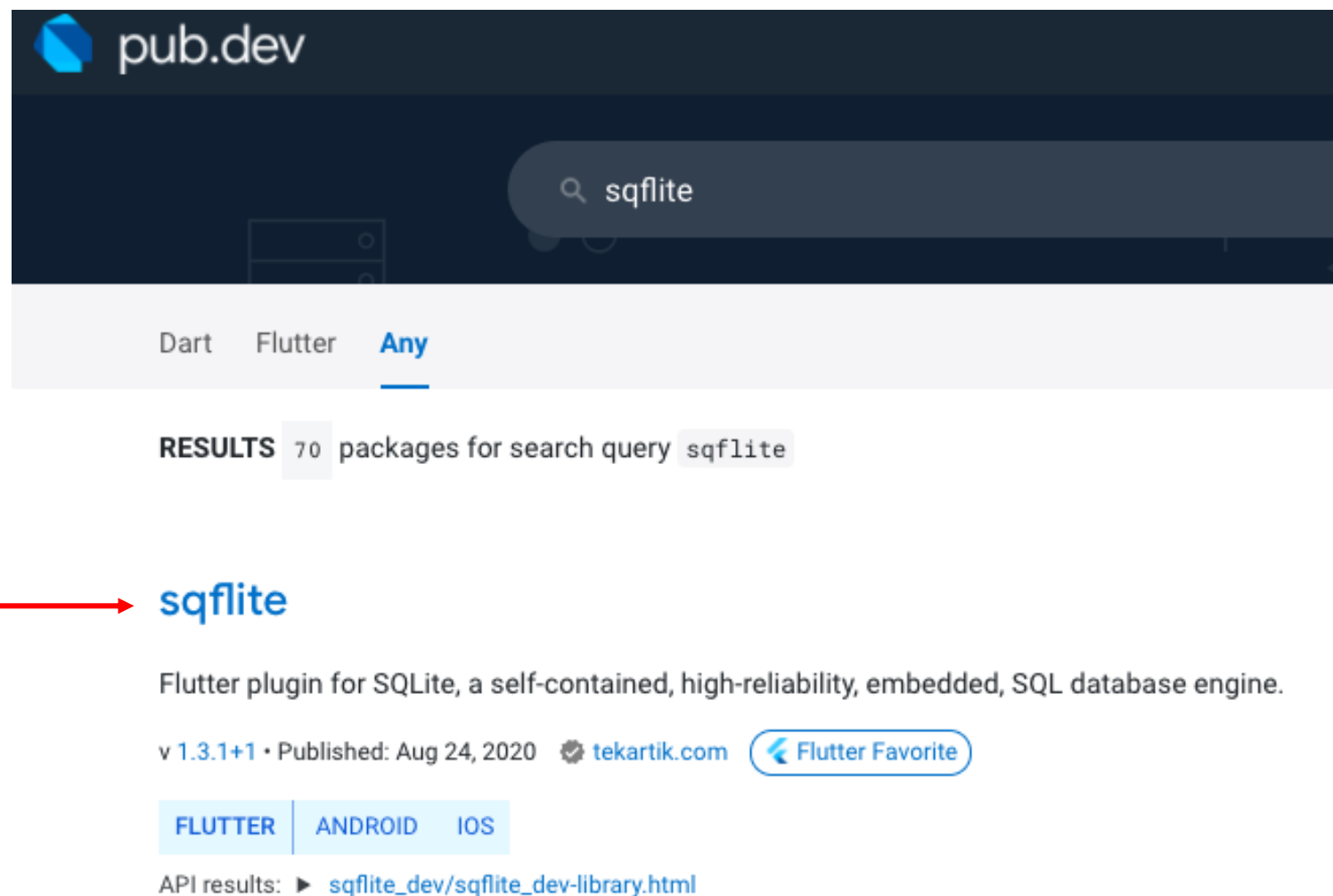
SQLite

- Por ser um BD 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 tipos antes de serem salvos no banco de dados



SQLite

Vamos usar o plugin **sqflite**, que possui suporte tanto para o Android quanto para o iOS



The screenshot shows the pub.dev search results for the 'sqflite' package. The search bar at the top contains the text 'sqflite'. Below the search bar, there are tabs for 'Dart', 'Flutter', and 'Any', with 'Any' currently selected. The results section shows '70 packages for search query sqflite'. The top result is 'sqflite', which is a Flutter plugin for SQLite. The description states it is a self-contained, high-reliability, embedded, SQL database engine. The version is 'v 1.3.1+1', published on 'Aug 24, 2020', by 'tekartik.com'. There is a 'Flutter Favorite' button next to the publisher information. Below the package name, there are tabs for 'FLUTTER', 'ANDROID', and 'IOS', with 'FLUTTER' selected. At the bottom, the API results are listed as 'sqflite_dev/sqflite_dev-library.html'.

pub.dev



sqflite

Dart Flutter **Any**

RESULTS 70 packages for search query sqflite

sqflite

Flutter plugin for SQLite, a self-contained, high-reliability, embedded, SQL database engine.

v 1.3.1+1 • Published: Aug 24, 2020 •  tekartik.com 

FLUTTER | **ANDROID** | **IOS**

API results: ▶ [sqflite_dev/sqflite_dev-library.html](https://pub.dev/documentation/sqflite_dev/sqflite_dev-library.html)

SQLite

sqlite 2.0.0+3

Published Mar 16, 2021 • [tekartik.com](https://pub.dev/packages/sqlite) Null safety

[FLUTTER](#) [ANDROID](#) [IOS](#) [MACOS](#)

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

sqlite

pub v2.0.0 build passing  codecov 57%

SQLite plugin for [Flutter](#). Supports iOS, Android and MacOS.

- Support transactions and batches
- Automatic version management during open
- Helpers for insert/query/update/delete queries
- DB operation executed in a background thread on iOS and Android

sqlite 2.0.0+3

Published Mar 16, 2021 • [tekartik.com](https://pub.dev/packages/sqlite) Null safety

[FLUTTER](#) [ANDROID](#) [IOS](#) [MACOS](#)

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

Use this package as a library

Depend on it

Run this command:

With Flutter:

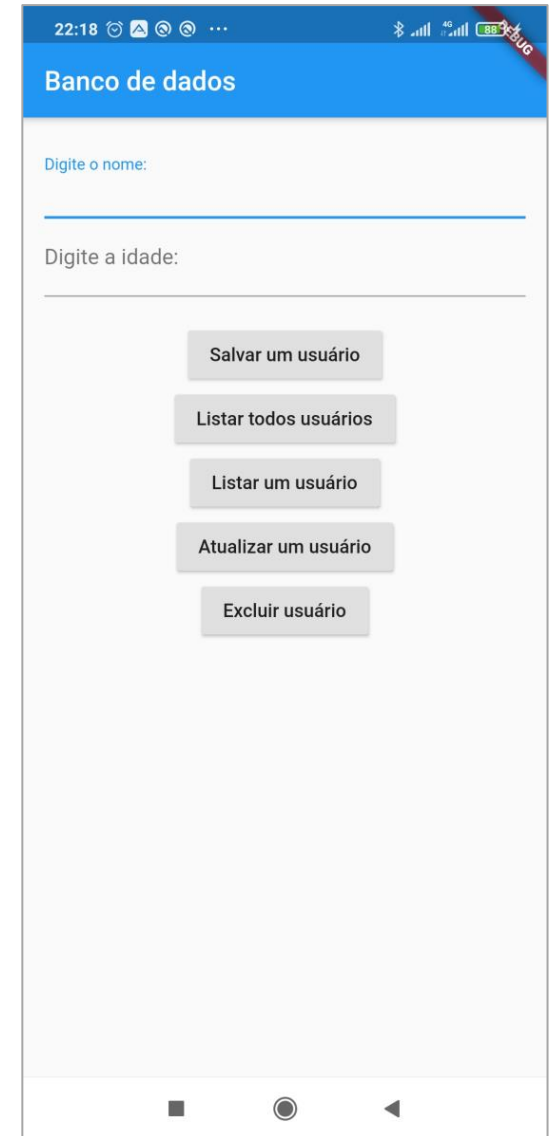
```
$ flutter pub add sqlite
```

This will add a line like this to your package's pubspec.yaml (and run an implicit `dart pub get`):

```
dependencies:  
  sqlite: ^2.0.0+3
```

Exemplo - SQLite

- Vamos fazer um exemplo, salvando, editando e recuperando dados de um banco de dados no dispositivo
- Para usar o BD vamos precisar criar um caminho para o banco de dados (ele precisa ficar armazenado em algum lugar), no caso, ficará no celular do usuário, em algum caminho



Exemplo - SQLite

Configurando o Banco de dados

```
class Home extends StatefulWidget {  
  _recuperarBancoDados() {  
    final caminhoBD = getDatabasesPath();  
    final localBD = join(caminhoBD, "banco.db");  
  }  
}
```

The argument type 'Future<String>' can't be assigned to the parameter type 'String'.
[Open documentation](#)

[Remove unused local variable](#) [More actions...](#)

Para usar o método (join) para passar o caminho do banco de dados, é necessário fazer import de um package => **import 'package:path/path.dart';**

Observe que o método criado para recuperar o banco de dados deverá ser assíncrono, pois o argumento com o caminho do banco de dados, é do tipo Future. Sendo assim, o método deve ser assíncrono e precisamos usar o await antes do getDatabasesPath

Exemplo - SQLite

Configurando o Banco de dados

```
class Home extends StatefulWidget {  
  recuperarBancoDados() async {  
    final caminhoBD = await getDatabasesPath();  
    final localBD = join(caminhoBD, "banco.db");  
  }  
}
```

Método
assíncrono

Usaremos o
await

Aqui podemos
colocar o nome
que quisermos
para o banco de
dados

Exemplo - SQLite

- Usaremos o método `openDatabase` que pode receber vários parâmetros, tais como:
- o **localBD**, que é o nosso banco;
- a **version**, que é a versão do banco que nós indicaremos;
- o **onCreate**, no qual criaremos nossa primeira tabela “usuários”

Configurando o Banco de dados

```
var retorno = await openDatabase(  
    localBD,  
    version: 1,  
    onCreate: (db, dbVersaoRecente){  
        String sql = "CREATE TABLE usuarios ("  
            "id INTEGER PRIMARY KEY AUTOINCREMENT, "  
            "nome VARCHAR, idade INTEGER)";  
        db.execute(sql);  
    }  
);
```

É possível incluir a
instrução SQL diretamente
dentro desse método

Exemplo - SQLite

Configurando o Banco de dados

```
_recuperarBancoDados() async {  
  final caminhoBD = await getDatabasesPath();  
  final localBD = join(caminhoBD, "banco.db");  
  var retorno = await openDatabase(  
    localBD,  
    version: 1,  
    onCreate: (db, dbVersaoRecente) {  
      String sql = "CREATE TABLE usuarios ("  
        "id INTEGER PRIMARY KEY AUTOINCREMENT, "  
        "nome VARCHAR, idade INTEGER)";  
      db.execute(sql);  
    }  
  );  
  print("Aberto " + retorno.isOpen.toString());  
}
```

Vamos criar uma variável (retorno) para receber o retorno do método openDatabase(), apenas para verificar se o banco de dados realmente está sendo criado e aberto e está tudo ok

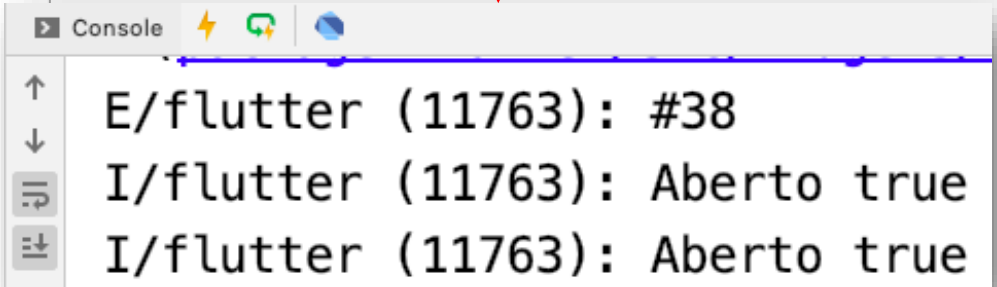
Vamos imprimir o valor do retorno, que basicamente será true ou false

Exemplo - SQLite

Configurando o Banco de dados

```
_recuperarBancoDados() async {  
  final caminhoBD = await getDatabasesPath();  
  final localBD = join(caminhoBD, "banco.db");  
  var retorno = await openDatabase(  
    localBD,  
    version: 1,  
    onCreate: (db, dbVersaoRecente){  
      String sql = "CREATE TABLE usuarios ("  
        "id INTEGER PRIMARY KEY AUTOINCREMENT, "  
        "nome VARCHAR, idade INTEGER)";  
      db.execute(sql);  
    }  
  );  
  print("Aberto " + retorno.isOpen.toString());  
}
```

Ao rodar o APP com apenas uma chamada para `_recuperarBancoDados()`, a tela ficará preta, porém no console deverá ser apresentado o texto "Aberto true", o que significa que deu certo a criação do banco de dados e tabela



The screenshot shows the Flutter console with the following output:

```
E/flutter (11763): #38  
I/flutter (11763): Aberto true  
I/flutter (11763): Aberto true
```

A red arrow points from the text box above to the console output.

Exemplo - SQLite

```
Future<Database> openDatabase(String path,  
    {int version,  
    OnDatabaseConfigureFn onConfigure,  
    OnDatabaseCreateFn onCreate,  
    OnDatabaseVersionChangeFn onUpgrade,  
    OnDatabaseVersionChangeFn onDowngrade,  
    OnDatabaseOpenFn onOpen,  
    bool readOnly = false,  
    bool singleInstance = true}) {
```

O que significa Future?

Um código Dart é executado em uma única thread. Se o código for bloqueado por estar executando uma operação muito longa ou estiver esperando por uma operação de arquivo, o programa inteiro vai 'congelar'.

As operações assíncronas permitem que o seu programa conclua outro trabalho enquanto aguarda a conclusão de uma operação. O Dart usa objetos Future (futures) para representar os resultados de operações assíncronas. Para trabalhar com futures, você pode usar o `async` e o `await` ou a API Future.

Um future é um objeto `Future<T>`, que representa uma operação assíncrona que produz um resultado do tipo T. Se o resultado não for um valor utilizável, o tipo do future será `Future<void>`. Quando uma função que retorna um future é invocada, duas coisas acontecem:

1. A função enfileira o trabalho a ser feito e retorna um objeto Future não concluído;
2. Posteriormente, quando a operação for concluída, o objeto Future será concluído com um valor ou com um erro;

Ao escrever código que depende de um future, você tem duas opções:

1. Use `async` e `await`

2. Use a API Future

Exemplo - SQLite

Salvando dados no Banco de dados

```
_salvarDados(String nome, int idade) async {  
  Database bd = await _recuperarBancoDados();  
  Map<String, dynamic> dadosUsuario = {  
    "nome" : nome,  
    "idade" : idade  
  };  
  int id = await bd.insert("usuarios", dadosUsuario);  
  print("Salvo: $id " );  
}
```

Ao clicar em salvar, os dados digitados no Text Field será armazenado no banco, utilizando o método insert

22:18

Banco de dados

Digite o nome:

Digite a idade:

Salvar um usuário

Listar todos usuários

Listar um usuário

Atualizar um usuário

Excluir usuário

Exemplo - SQLite

`db.insert(table, values)`

Nome da tabela
criada

O values é
um Map<>

O método insert
espera por dois
parâmetros, o nome
da tabela e o values,
que é um map

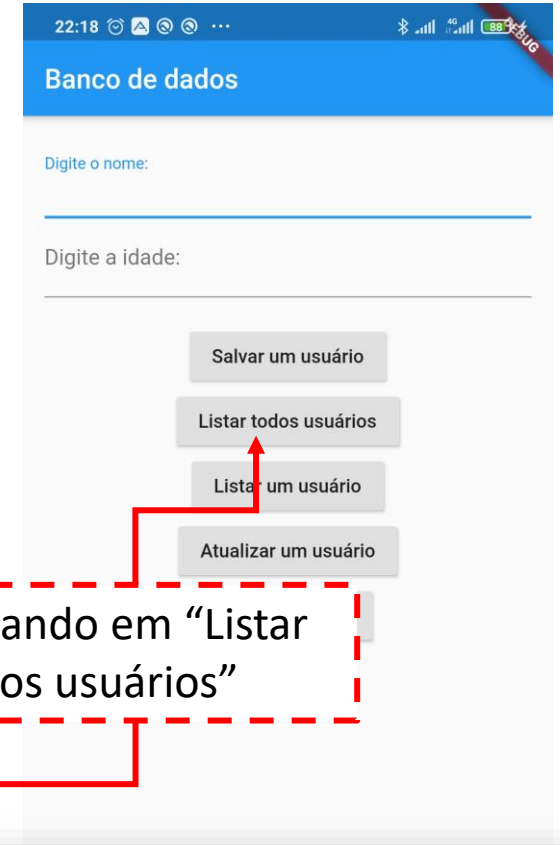
```
/// INSERT helper
```

```
Future<int> insert(String table, Map<String, dynamic> values,  
    {String nullColumnHack, ConflictAlgorithm conflictAlgorithm});
```

Exemplo - SQLite

Listar todos os usuários cadastrados no Banco de dados

```
_listarUsuarios() async{  
  Database bd = await _recuperarBancoDados();  
  String sql = "SELECT * FROM usuarios";  
  List usuarios = await bd.rawQuery(sql);  
  for(var usu in usuarios){  
    print(" id: "+usu['id'].toString() +  
      " nome: "+usu['nome']+  
      " idade: "+usu['idade'].toString());  
  }  
}
```



```
Console  
↑  
↓  
I/flutter ( 804): id: 1 nome: Raquel Ribeiro idade: 26  
I/flutter ( 804): id: 2 nome: Antonio Pedro idade: 35  
I/flutter ( 804): id: 3 nome: Raquel Ribeiro idade: 26  
I/flutter ( 804): id: 4 nome: Raquel Ribeiro idade: 26
```

Exemplo - SQLite

Listar todos os usuários cadastrados no Banco de dados

```
_listarUsuarios() async{  
    Database bd = await _recuperarBancoDados();  
    String sql = "SELECT * FROM usuarios";  
    List usuarios = await bd.rawQuery(sql);  
    for(var usu in usuarios){  
        print(" id: "+usu['id'].toString() +  
            " nome: "+usu['nome']+  
            " idade: "+usu['idade'].toString());  
    }  
}
```

Usando o método
rawQuery é possível
colocar a query que
quisermos do SQL

```
//String sql = "SELECT * FROM usuarios WHERE idade=58";  
//String sql = "SELECT * FROM usuarios WHERE idade >=30 AND idade <=58";  
//String sql = "SELECT * FROM usuarios WHERE idade BETWEEN 18 AND 58";  
//String sql = "SELECT * FROM usuarios WHERE nome='Maria Silva'";
```

Exemplo - SQLite

Listar um usuário apenas cadastrado no Banco de dados

```
_listarUmUsuario(int id) async{  
    Database bd = await _recuperarBancoDados();  
    List usuarios = await bd.query(  
        "usuarios",  
        columns: ["id", "nome", "idade"],  
        where: "id = ?",  
        whereArgs: [id]  
    );  
    for(var usu in usuarios){  
        print(" id: "+usu['id'].toString() +  
            " nome: "+usu['nome']+  
            " idade: "+usu['idade'].toString());  
    }  
}
```

Usamos um caracter curinga ? na cláusula where e na whereArgs indicamos o valor, no caso, estamos usando o id passado por parâmetro

Exemplo - SQLite

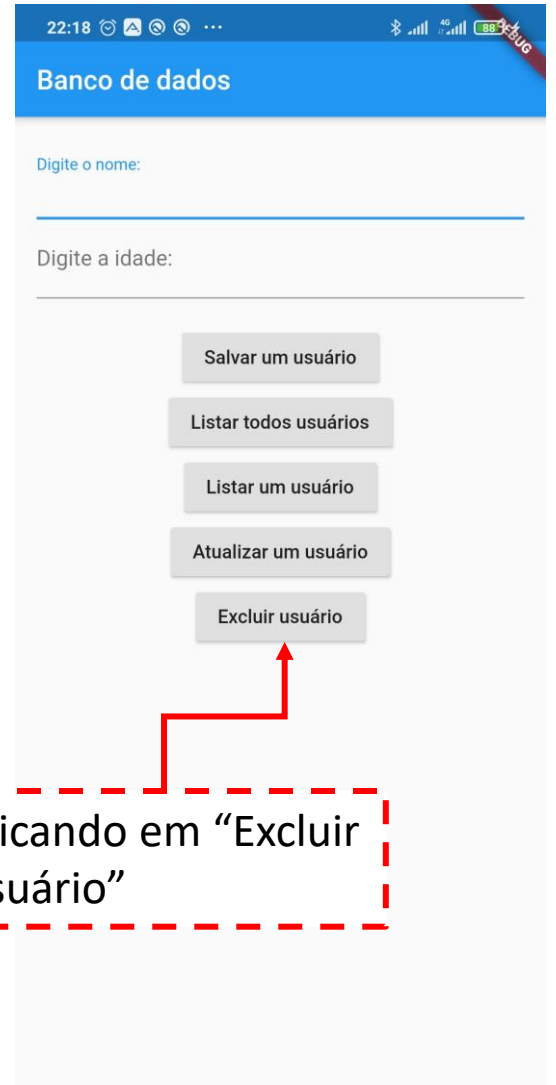
Excluir um usuário apenas cadastrado no Banco de dados

```
_excluirUsuario(int id) async{  
    Database bd = await _recuperarBancoDados();  
    int retorno = await bd.delete(  
        "usuarios",  
        where: "id = ?", // caracter curinga  
        whereArgs: [id]  
    );  
    print("Itens excluidos: "+retorno.toString());  
}
```


Exemplo - SQLite

Excluir um usuário apenas cadastrado no Banco de dados

```
_excluirUsuario() async{  
    Database bd = await _recuperarBancoDados();  
    int retorno = await bd.delete(  
        "usuarios",  
        where: "nome = ? AND idade = ?", //caracter curinga  
        whereArgs: ["Raquel Ribeiro", 26]  
    );  
    print("Itens excluidos: "+retorno.toString());  
}
```



Exemplo - SQLite

Excluir um usuário apenas cadastrado no Banco de dados

```
_excluirUsuario() async{  
    Database bd = await _recuperarBancoDados();  
    int retorno = await bd.delete(  
        "usuarios",  
        where: "nome = ? AND idade = ?", // caracter curinga  
        whereArgs: ["Raquel Ribeiro", 26]  
    );  
    print("Itens excluidos: "+retorno.toString());  
}
```

Exemplo - SQLite

```
Console
I/flutter ( 804): id: 1 nome: Raquel Ribeiro idade: 26
I/flutter ( 804): id: 2 nome: Antonio Pedro idade: 35
I/flutter ( 804): id: 3 nome: Raquel Ribeiro idade: 26
I/flutter ( 804): id: 4 nome: Raquel Ribeiro idade: 26
```

1- Cliquei no botão
listar todos usuários

```
Console
I/flutter ( 804): id: 1 nome: Raquel Ribeiro idade: 26
I/flutter ( 804): id: 2 nome: Antonio Pedro idade: 35
I/flutter ( 804): id: 3 nome: Raquel Ribeiro idade: 26
I/flutter ( 804): id: 4 nome: Raquel Ribeiro idade: 26
I/flutter ( 804): Itens excluidos: 3
```

2- Cliquei em excluir
usuário, usando o filtro
nome "Raquel Ribeiro" e
idade "26"

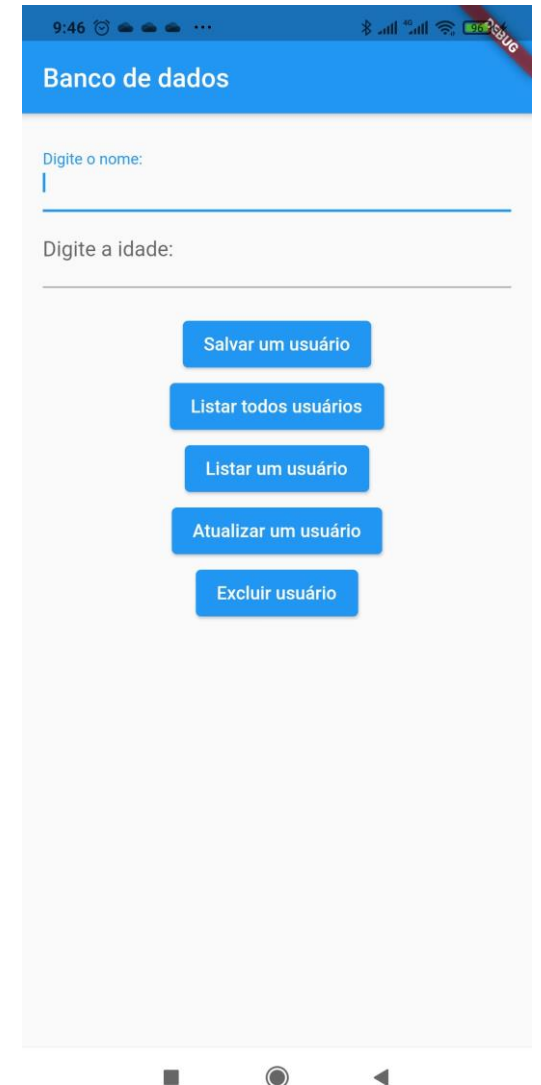
```
I/flutter ( 804): Salvo: 5
I/flutter ( 804): id: 2 nome: Antonio Pedro idade: 35
I/flutter ( 804): id: 5 nome: João Pedro idade: 29
```

3- Salvei um novo
nome e idade,
observe o id

Exemplo - SQLite

Atualizar um usuário apenas cadastrado no Banco de dados

```
_atualizarUsuario(int id) async{  
    Database bd = await _recuperarBancoDados();  
    Map<String, dynamic> dadosUsuario = {  
        "nome" : "Antonio Pedro",  
        "idade" : 35,  
    };  
    int retorno = await bd.update(  
        "usuarios", dadosUsuario,  
        where: "id = ?", // caracter curinga  
        whereArgs: [id]  
    );  
    print("Itens atualizados: "+ retorno.toString());  
}
```



Referências Bibliográficas

- Curso da Udemy – Flutter Essencial do professor Ricardo Lecheta.
- Curso da Udemy - Desenvolvimento Android e IOS com Flutter 2020 – Crie 15 Apps do professor Jamilton Damasceno.
- ELMASRI, Ramez; NAVATHE, Sham. **Sistemas de banco de dados.** 7. ed. São Paulo: Pearson, c2019. E-book. ISBN 9788543025001.
- http://www.macoratti.net/19/08/flut_accsqlite1.htm