Usando SQLite em Flutter

Prof. Markus Endler e Felipe Carvalho

http://www.inf.puc-rio.br/~endler/courses/Flutter

LAC

Laboratory for Advanced Collaboration

Pontifícia Universidade Católica do Rio de Janeiro

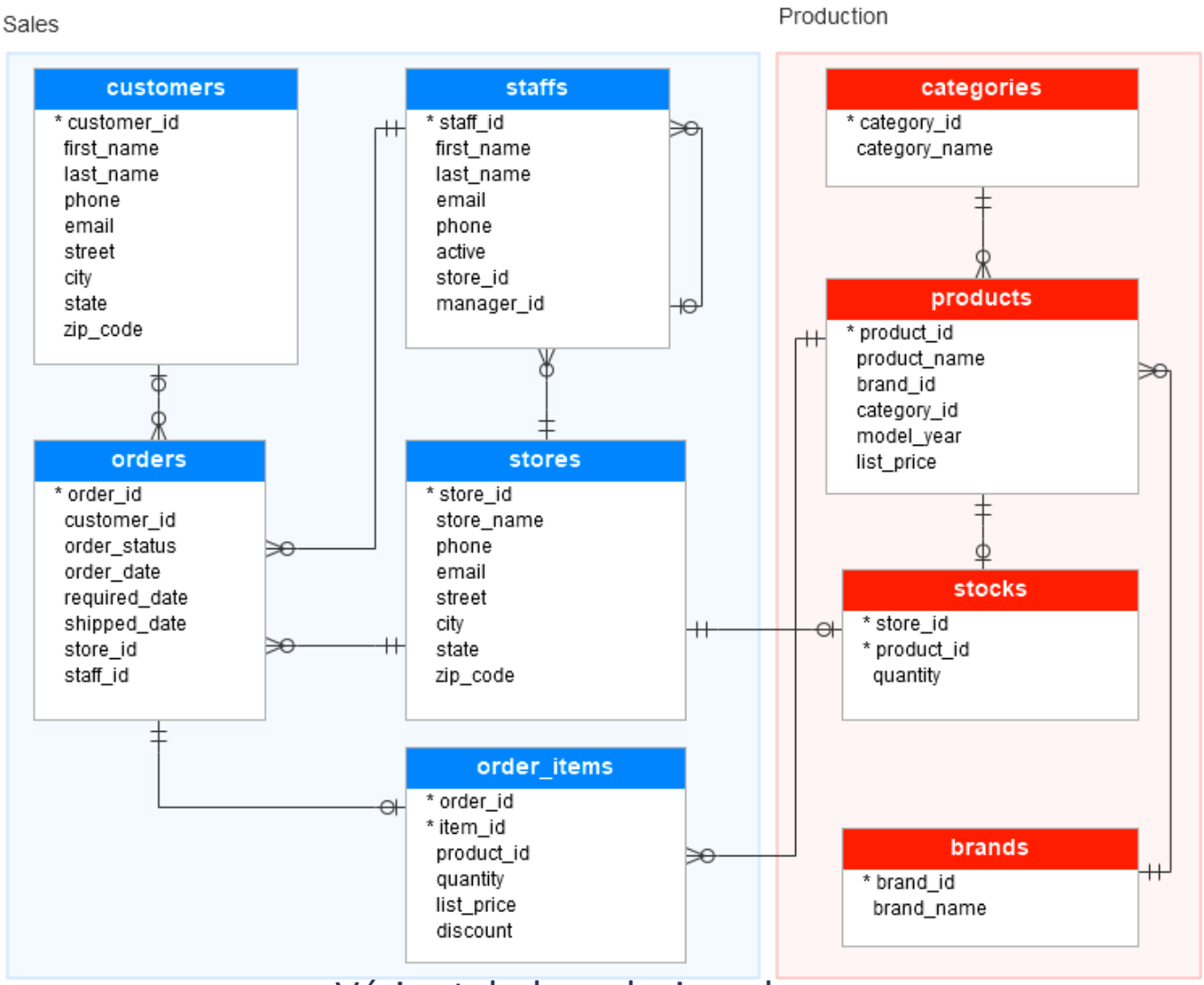
Persistência local de dados

- Persistir dados no aplicativo é muito importante
- Seria muito ruim se o usuário tiversse que digitar dados a cada vez que usasse o aplicativo
- Seria muito custoso esperar o aplicativo carregar os mesmos dados da rede.

ID	Title	ISBN	Author	Publishing...	isAvailable
1	1984	2343454895456	George Orwell	04/12/1979	☒
2	Anna Karenina	1234548485843	Leo Tolstoy	07/11/1998	☒
4	The Adventures of I	3450345345443	Mark Twain	08/11/1999	☒
5	Ulysses	9944933003232	James Joyce	06/05/2010	☒
8	War and Peace	0944344903312	Leo Tolstoy	08/11/2001	☐
11	The Brothers Karan	9003940397271	Doso	04/07/2012	☒
12	On the Road	0459450444310	Jack Kerouac	30/12/2005	☐
15	The Metamorphosi	2003948930545	Franz Kafka	09/03/1976	☒
16	The Illiad	9449039333923	Homer	05/07/1998	☒
17	The Odyssey	8409404850139	Homer	06/08/1999	☒

Unica tabela

Banco de dados complexo



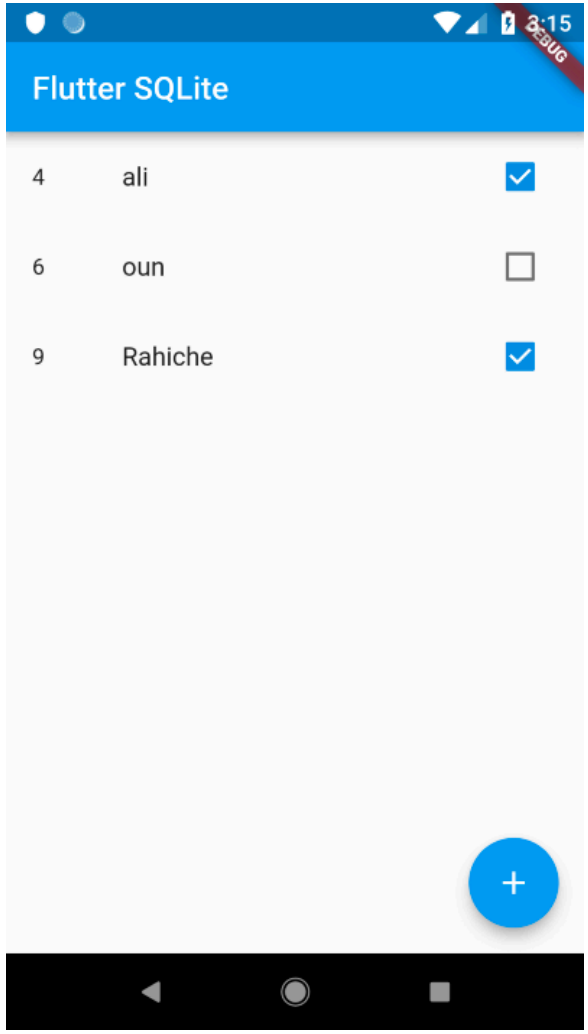
Várias tabelas relacionadas

Persistência local de dados

- O SQLite é uma das formas mais populares de armazenar dados localmente.
- Iremos usar o pacote `sqflite` para conectar com o SQLite.
- É um dos pacotes mais usados e maduros para se conectar a bancos de dados SQLite no Flutter.

Exemplo muito simples:

- Cada entrada no banco de dados terá apenas um Id, dois nomes e um campo binário (checkbox)



Dependências

- Em `pubspec.yaml`, na secão dependencies
- Adicione a última versão de `sqflite` e `path_provider`

```
dependencies:
  flutter:
    sdk: flutter
  sqflite: any
  path_provider: any
```

- Use pacote `path_provider` para obter as localizacões mais comumente usadas:
  - `TemporaryDirectory`,
  - `ApplicationDocumentsDirectory`.

- No seu projeto Flutter crie um novo arquivo, por exemplo, `Database.dart`, onde criamos um singleton (para garantir uma única instância de classe e um ponto global de acesso)

Criando um cliente do banco de dados

- Cria um construtor private que pode ser usado somente dentyro da classe

```
class DBProvider {
  DBProvider._();
  static final DBProvider db = DBProvider._();
}
```

- Iremos criar o objeto de banco de dados e forneceremos um getter através do qual iremos instanciar o banco de dados, se não for (inicialização lazy).

```
static Database _database;
  Future<Database> get database async {
    if (_database != null)
    return _database;

    // if _database is null we instantiate it
    _database = await initDB();
    return _database;
  }
```

- Se não existir um objeto associado ao banco de dados, usaremos `initDB` para criar o banco de dados. Nessa função iremos receber o path para armazenar o banco de dados e criar as tabelas correspondentes.

Criando as tabelas (nesse caso é apenas uma "Client")

- Criamos o banco de dados TestDB.db

```
initDB() async {
    Directory documentsDirectory = await getApplicationDocumentsDirectory();
    String path = join(documentsDirectory.path, "TestDB.db");
    return await openDatabase(path, version: 1, onOpen: (db) {
    }, onCreate: (Database db, int version) async {
      await db.execute("CREATE TABLE Client ("
          "id INTEGER PRIMARY KEY,"
          "first_name TEXT,"
          "last_name TEXT,"
          "blocked BIT"
          ")");
    });
  }
```

- Contém uma única tabela, "Client":
- Tabela é criada com comando SQL CREATE TABLE

  com campos <id, first_name, last_name, blocked>

Criando Classes Modelo

- Os dados no banco de dados serão convertidos usando um Dart map.
- Por isso, precisamos criar classes modelos usando os métodos toMap e fromMap

```
import 'dart:convert';

Client clientFromJson(String str) => Client.fromJson(json.decode(str));
String clientToJson(Client data) => json.encode(data.toJson());

class Client {
    int id;
    String firstName;
    String lastName;
    bool blocked;

    Client({
        this.id,
        this.firstName,
        this.lastName,
        this.blocked,
    });
```

```
    factory Client.fromJson(Map<String, dynamic> json)
=> new Client(
        id: json["id"],
        firstName: json["first_name"],
        lastName: json["last_name"],
        blocked: json["blocked"],
    );

    Map<String, dynamic> toJson() => {
        "id": id,
        "first_name": firstName,
        "last_name": lastName,
        "blocked": blocked,
    };
}
```


As operações CRUD: Create

Criação:

- O pacote sqflite permite duas formas: `rawInsert` e `Insert`

```
newClient(Client newClient) async {
    final db = await database;
    var res = await db.rawInsert(
      "INSERT Into Client (id,first_name)"
      " VALUES (${newClient.id},${newClient.firstName})");
    return res;
  }
```

Opção1: usando o nome da tabela

```
newClient(Client newClient) async {
    final db = await database;
    var res = await db.insert("Client",
newClient.toMap());
    return res;
  }
```

Opção2: usando um map que contém os dados

Consulta (Read)

Exemplo 1: Obter por ID

- a consulta é feita com o id whereArgs. E retorna-se o primeiro resultado se a lista não estiver vazia ou senão, null.

```
getClient(int id) async {
    final db = await database;
    var res =await  db.query("Client", where: "id = ?", whereArgs:
[id]);
    return res.isNotEmpty ? Client.fromMap(res.first) : Null ;
  }
```

- Exemplo 2 : Obter todos dados q satisfazem uma condição – usamos rawQuery e mapeamos a lista de resultados para uma lista de objetos Client:

```
getAllClients() async {
    final db = await database;
    var res = await db.query("Client");
    List<Client> list =
        res.isNotEmpty ? res.map((c) => Client.fromMap(c)).toList() : [];
    return list;
  }
```


Consulta (Read)

- Consulta
- Exemplo 3: obtendo somente os itens que estão com flag BLOCKED=true

```
getBlockedClients() async {
    final db = await database;
    var res = await db.rawQuery("SELECT * FROM Client WHERE blocked=1");
    List<Client> list =
        res.isNotEmpty ? res.toList().map((c) => Client.fromMap(c)) :
null;
    return list;
  }
```


Update

- Dados do novo item (cliente) precisam ser transformados para map

```
updateClient(Client newClient) async {
    final db = await database;
    var res = await db.update("Client", newClient.toMap(),
        where: "id = ?", whereArgs: [newClient.id]);
    return res;
  }
```

- Exemplo concreto de uso: trocando o flag BLOCKED

```
blockOrUnblock(Client client) async {
    final db = await database;
    Client blocked = Client(
        id: client.id,
        firstName: client.firstName,
        lastName: client.lastName,
        blocked: !client.blocked);
    var res = await db.update("Client", blocked.toMap(),
        where: "id = ?", whereArgs: [client.id]);
    return res;
  }
```


Apagando registros (Delete)

- Apagando um único registro, o de id

```
deleteClient(int id) async {
    final db = await database;
    db.delete("Client", where: "id = ?", whereArgs: [id]);
  }
```

- Apagando todos os registros

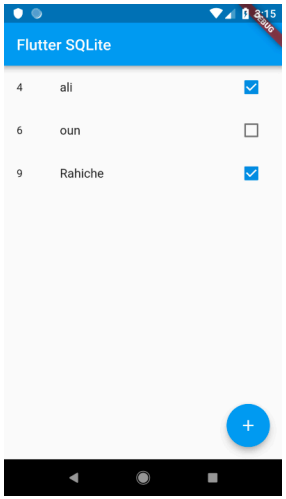
```
deleteAll() async {
    final db = await database;
    db.rawDelete("Delete * from Client");
  }
```


Código Flutter do layout

- `FutureBuilder` obtém dos dados do DB como List

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("Flutter SQLite")),

    body: FutureBuilder<List<Client>>(
      future: DBProvider.db.getAllClients(),
      builder: (BuildContext context, AsyncSnapshot<List<Client>> snapshot) {
        if (snapshot.hasData) {
          return ListView.builder(
            itemCount: snapshot.data.length,
            itemBuilder: (BuildContext context, int index) {
              Client item = snapshot.data[index];
              return ListTile(
                title: Text(item.lastName),
                leading: Text(item.id.toString()),
                trailing: Checkbox(
                  onChanged: (bool value) {
                    DBProvider.db.blockClient(item);
                    setState(() {});
                  },
                  value: item.blocked,
                ),
              );
            },
          );
        } else {
          return Center(child: CircularProgressIndicator());
        }
      },
    ),
  );
}
```



- Se não houver dados

Os dados

- Exemplos de dados”

```
List<Client> testClients = [
    Client(firstName: "Raouf", lastName: "Rahiche", blocked: false),
    Client(firstName: "Zaki", lastName: "oun", blocked: true),
    Client(firstName: "oussama", lastName: "ali", blocked: false),
  ];
```

AsyncSnapshot <T> class

- Representação imutável da interação mais recente com uma computação assíncrona.

Incluindo & removendo registros

- Clicando no FloatingActionButton adiciona um registro aleatório ao banco de dados.

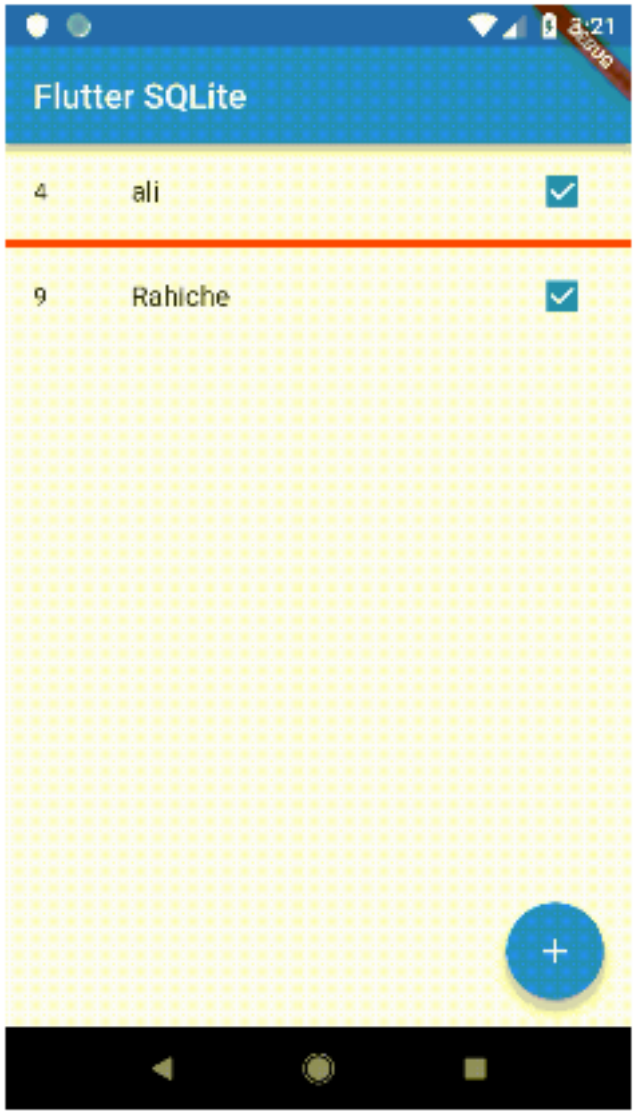
```
 floatingActionButton: FloatingActionButton(
        child: Icon(Icons.add),
        onPressed: () async {
          Client rnd =
testClients[math.Random().nextInt(testClients.length)];
          await DBProvider.db.newClient(rnd);
          setState(() {});
        },
      ),
    );
```

- Se desejo excluir um registro quando o Tile é movido para a esuerda, simplesmente basta envolver o `ListTile` com um widget `Dismissible`:

```
return Dismissible(
   key: UniqueKey(),
   background: Container(color: Colors.red),
   onDismissed: (direction) {
    DBProvider.db.deleteClient(item.id);
   },
    child: ListTile(...),
  );
```



Flutter SQLite
4 ali
6 oun
9 Rahiche


Flutter SQLite
4 ali
9 Rahiche


Q&A

Lembrando...

T1 - INF1300

Desenvolver um app flutter/Dart com:

> Pelo menos um StatefulWidget (mas o menor número possível).
> Navegação (push/pop) entre varias páginas (pelo menos 3).
> Widget que contém um Row, um Column ou ListView.
> Exibe figuras em assets como lista de imagens.
> Usar Future + async + await.
> Fetch de algum JSON da Web (REST) e preenchimento de um objeto local.
> Alguma animação renderizada.

Prazo de entrega: 30/04/2019
Projeto Flutter + Manual de usuário (mais ou menos 4 páginas).

Enviar para: ocfelipe@gmail.com e endler@inf.puc-rio.br