Flutter :Sqlite

Rogério B. de Andrade

SqlLite: Introdução

- O uso do SQLite como fonte de dados em um aplicativo permite armazenar e acessar dados de forma eficiente, mesmo offline.
- O SQLite é um banco de dados embutido de código aberto, que oferece uma solução leve e eficiente para armazenamento de dados em aplicativos móveis. Está disponível de forma nativa nos dispositivos Android e IOS.
- Características e benefícios para o desenvolvimento de aplicativos móveis: portabilidade, facilidade de uso, economia de recursos, suporte a transações ACID, entre outros.

SqlLite: Introdução

- O Flutter oferece suporte nativo para integração com o SQLite, permitindo a criação de aplicativos poderosos e eficientes.
- O SQLite permite que os aplicativos armazenem dados localmente no dispositivo do usuário, possibilitando o acesso a informações mesmo quando não há conexão com a internet.
- O SQLite é conhecido por sua velocidade e eficiência na recuperação e manipulação de dados, o que é essencial para garantir um desempenho suave no desenvolvimento de aplicativos

SqlLite: Configuração

- O SQLite é um banco de dados relacional que segue os princípios ACID (Atomicidade, Consistência, Isolamento, Durabilidade), garantindo a integridade dos dados e a consistência das operações.
- Adicionando a Dependência do SQLite (pubespc.yaml):

```
dependencies:
flutter:
    sdk: flutter
  sqflite: ^2.0.0
  path: ^1.8.0
```

SqlLite: Implementação

› Bibliotecas a serem importadas para uso do Sqlite:

```
import 'package:flutter/material.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
```

SqlLite: Definindo a base de dados

```dart
// método para criar/abrir o banco de dados
Future<void> initializeDatabase() async {
  // obtém path (local) de armazenamento do banco de dados
  final String databasesPath = await getDatabasesPath();
  // associa local com nome da base de dados
  final String path = join(databasesPath, 'my_database.db');
  // cria/abre banco de dados e cria tabela se não existe
  _database = await openDatabase(
    path,
    version: 1,
    onCreate: (db, version) async {
      await db.execute('''
        CREATE TABLE IF NOT EXISTS users (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          name TEXT,
          address TEXT
        )
      ''');
    },
  );
}
```

- O código ao lado abre ou cria a base dados e posteriormente cria a tabela users (se não existir) definindo os campos id, name e adress

SqlLite: Inserindo registros em uma tabela

```dart
// cria um novo registro na base de dados
Future<void> createUser() async {
  //obtém as informações do formulário
  final String name = _nameController.text;
  final String address = _addressController.text;
  // define um objeto que encapsula as informações do formulário
  final row = {
    'name': name,
    'address': address,
  };
  // insere as informações na tabela
  final id = await _database.insert('users', row);
  // limpa formulário
  setState(() {
    _nameController.text = '';
    _addressController.text = '';
  });
  // verifica inserção obteve êxito
  if (id != null) {
    debugPrint('User created successfully with id: $id');
  } else {
    debugPrint('Failed to create user');
  }
}
```

- O código utiliza o método insert para inserir um novo registro na tabela do banco de dados. O primeiro parâmetro é o nome da tabela onde as informações serão inseridas e o segundo parâmetro é um objeto que contém as informações a serem inseridas.

SqlLite: Recuperando informações

```dart
// lê informações armazenadas em tabela de banco de dados
Future<void> readUsers() async {
  // cria uma lista e recupera as informações inseridasd
  final List<Map<String, dynamic>> users = await _database.query('users');
  // associa state _users ao resultado da query
  setState(() {
    _users = users;
  });
  // se há informações, mostra no terminal com debugPrint
  if (_users.isNotEmpty) {
    for (final user in _users) {
      final name = user['name'];//acessa campo da tabela
      final address = user['address'];
      debugPrint('Name: $name, Address: $address');
    }
  } else {
    debugPrint('No users found');
  }
}
```

- O método query retorna as informações armazenas na tabela e armazena na variável users (lista). O conteúdo é copiado no state _users.

- Após verificar se há informações (_users.isNotEmpty), as informações são acessadas utilizando um laço de repetição, que permite o acesso individual aos campos (entre []).

SqlLite: Atualizando informações

```dart
// atualizando um registro
Future<void> updateUser() async {
  // obtém as informações do formulário
  final String name = _nameController.text;
  final String address = _addressController.text;
  // encapsula as informações
  final row = {'name': name,'address': address,};
  //atualiza
  final updatedRows = await _database.update(
    'users',row,// recebe informações
    where: 'name = ?',//critério pelo nome
    whereArgs: [name],// associa critério com parâmetro
  );
  // limpa formulário
  setState(() {
    _nameController.text = '';
    _addressController.text = '';
  });
  if (updatedRows > 0) {  // verifica êxito da atualização
    debugPrint('User updated successfully');
  } else {
    debugPrint('Failed to update user');
  }
}
```

- O método update permite atualizar informações de uma tabela. O primeiro parâmetro é o nome da tabela, o segundo objeto que contém as informações a serem atualizadas, o terceiro é o critério de atualização, o último os argumentos do critério.

SqlLite: Excluindo informações

```dart
// excluindo informações da tabela
Future<void> deleteUser() async {
  // obtém informação do formulário
  final String name = _nameController.text;
  // excluí registro da tabela
  final deletedRows = await _database.delete(
    'users', // define tabela
    where: 'name = ?',// critério da exclusão
    whereArgs: [name],// parâmetro da exclusão
  );
  // limpa formulário
  setState(() {
    _nameController.text = '';
    _addressController.text = '';
  });
  // verifica êxito da exclusão
  if (deletedRows > 0) {
    debugPrint('User deleted successfully');
  } else {
    debugPrint('Failed to delete user');
  }
}
```

- O método delete remove um registro do banco de dados.
- Os parâmetros são: tabela, critério e argumentos do critério.
- O retorno da exclusão na variável deleteRows permite verificar se houve êxito na exclusão (valor >0)

SqlLite: Atividade

- Modifique o exemplo visto anteriormente, acrescentando mais 4 campos (à sua escolha). Na exclusão, modifique o código para que o parâmetro de exclusão seja a chave primária (e não o nome). Para isto, ao consultar uma informação, armazene a chave primaria. Aprimore as mensagens de retorno em relação ao êxito ou não das funcionalidades Crud: informe o resultado em um Text ou na forma de janela de diálogo.
- Link do exemplo:
- https://docs.google.com/document/d/1na6gtUuay9mUxDEO7q9vJo_EYkSpQNag2oUSztrRVwQ/edit?usp=sharing