**CSC 4360/6360 — Mobile App Development**

**In\_class 10 - Local storage part 1**

**Due: October 15, 2025, 11:59 PM**

**Objective:** Create a simple Flutter app that integrates a local SQLite database using the sqflite package. This will give you practical experience with local database integration in Flutter.

**What to Submit**

1. **Main Dart File**: The primary file for your Flutter app.
2. **Helper File**: A file that contains helper functions or classes for database operations.
3. **Yaml File**: Your pubspec.yaml file with the necessary dependencies.

**Steps to Get Started**

1. **Set Up Your Project**:
   * Create a new Flutter project.
   * Add the sqflite and path\_provider packages to your pubspec.yaml file.
2. **Configure SQLite**:
   * Create a database helper class to manage database operations (e.g., opening the database, creating tables, CRUD operations).
3. **Implement UI**:
   * Design a simple UI to interact with the database (e.g., adding, displaying, and deleting records).
   * Include functionality to handle images if required.
4. **Test Your App**:
   * Ensure all database operations work correctly.
   * Test the app on an emulator or physical device.

### **Minimal Flutter App with Enhanced UI Design**

**Here is the minimal Flutter app that you will be creating. Add some border and padding to make the UI cleaner and more visually appealing.**

A screenshot of a cell phone

Description automatically generated

**The Function:** Pressing each button will perform the related operation on the database using SQFlite.

**For this minimal example, you will do the following steps:**

* **Add the dependencies**
* **Make a database helper class**
* **Make your app layout with the button logic**
* **Start by creating a new Flutter project called flutter\_database.**

**Dependencies**

Open the pubspec.yaml file and add the following lines to the dependencies section:

***sqflite: ^2.2.2***

***path\_provider: ^2.0.11***

***path: ^1.8.2***

You can find the current versions of these packages here:

1. [**sqflite**](https://pub.dartlang.org/packages/sqflite)
2. [**path\_provider**](https://pub.dartlang.org/packages/path_provider)
3. [**path**](https://pub.dev/packages/path)

**About the Database helper**

**Create a new file in the lib folder and call it database\_helper.dart**

**Your project should look something like this in VS Code:**

my\_flutter\_app/

├── android/

├── ios/

├── lib/

│ ├── services/

│ │ └── database\_helper.dart ← YOUR HELPER FILE

│ └── main.dart ← YOUR MAIN FILE

├── pubspec.yaml

└── ...

**Copy in the following code:**

|  |  |  |
| --- | --- | --- |
| import 'package:path/path.dart'; | | |
|  | import 'package:sqflite/sqflite.dart'; |
|  | import 'package:path\_provider/path\_provider.dart'; |
|  |  |
|  | class DatabaseHelper { |
|  | static const \_databaseName = "MyDatabase.db"; |
|  | static const \_databaseVersion = 1; |
|  |  |
|  | static const table = 'my\_table'; |
|  |  |
|  | static const columnId = '\_id'; |
|  | static const columnName = 'name'; |
|  | static const columnAge = 'age'; |
|  |  |
|  | late Database \_db; |
|  |  |
|  | // this opens the database (and creates it if it doesn't exist) |
|  | Future<void> init() async { |
|  | final documentsDirectory = await getApplicationDocumentsDirectory(); |
|  | final path = join(documentsDirectory.path, \_databaseName); |
|  | \_db = await openDatabase( |
|  | path, |
|  | version: \_databaseVersion, |
|  | onCreate: \_onCreate, |
|  | ); |
|  | } |
|  |  |
|  | // SQL code to create the database table |
|  | Future \_onCreate(Database db, int version) async { |
|  | await db.execute(''' |
|  | CREATE TABLE $table ( |
|  | $columnId INTEGER PRIMARY KEY, |
|  | $columnName TEXT NOT NULL, |
|  | $columnAge INTEGER NOT NULL |
|  | ) |
|  | '''); |
|  | } |
|  |  |
|  | // Helper methods |
|  |  |
|  | // Inserts a row in the database where each key in the Map is a column name |
|  | // and the value is the column value. The return value is the id of the |
|  | // inserted row. |
|  | Future<int> insert(Map<String, dynamic> row) async { |
|  | return await \_db.insert(table, row); |
|  | } |
|  |  |
|  | // All of the rows are returned as a list of maps, where each map is |
|  | // a key-value list of columns. |
|  | Future<List<Map<String, dynamic>>> queryAllRows() async { |
|  | return await \_db.query(table); |
|  | } |
|  |  |
|  | // All of the methods (insert, query, update, delete) can also be done using |
|  | // raw SQL commands. This method uses a raw query to give the row count. |
|  | Future<int> queryRowCount() async { |
|  | final results = await \_db.rawQuery('SELECT COUNT(\*) FROM $table'); |
|  | return Sqflite.firstIntValue(results) ?? 0; |
|  | } |
|  |  |
|  | // We are assuming here that the id column in the map is set. The other |
|  | // column values will be used to update the row. |
|  | Future<int> update(Map<String, dynamic> row) async { |
|  | int id = row[columnId]; |
|  | return await \_db.update( |
|  | table, |
|  | row, |
|  | where: '$columnId = ?', |
|  | whereArgs: [id], |
|  | ); |
|  | } |
|  |  |
|  | // Deletes the row specified by the id. The number of affected rows is |
|  | // returned. This should be 1 as long as the row exists. |
|  | Future<int> delete(int id) async { |
|  | return await \_db.delete( |
|  | table, |
|  | where: '$columnId = ?', |
|  | whereArgs: [id], |
|  | ); |
|  | } |
|  | } |

**App code**

Open your **main.dart** file and replace it with the following code:

|  |  |  |
| --- | --- | --- |
| import 'package:flutter/material.dart'; | | |
|  | import 'database\_helper.dart'; |
|  |  |
|  | // Here we are using a global variable. You can use something like |
|  | // get\_it in a production app. |
|  | final dbHelper = DatabaseHelper(); |
|  |  |
|  | Future<void> main() async { |
|  | WidgetsFlutterBinding.ensureInitialized(); |
|  | // initialize the database |
|  | await dbHelper.init(); |
|  | runApp(const MyApp()); |
|  | } |
|  |  |
|  | class MyApp extends StatelessWidget { |
|  | const MyApp({super.key}); |
|  |  |
|  | @override |
|  | Widget build(BuildContext context) { |
|  | return MaterialApp( |
|  | title: 'SQFlite Demo', |
|  | theme: ThemeData( |
|  | primarySwatch: Colors.blue, |
|  | ), |
|  | home: const MyHomePage(), |
|  | ); |
|  | } |
|  | } |
|  |  |
|  | class MyHomePage extends StatelessWidget { |
|  | const MyHomePage({super.key}); |
|  |  |
|  | // homepage layout |
|  | @override |
|  | Widget build(BuildContext context) { |
|  | return Scaffold( |
|  | appBar: AppBar( |
|  | title: const Text('sqflite'), |
|  | ), |
|  | body: Center( |
|  | child: Column( |
|  | mainAxisAlignment: MainAxisAlignment.center, |
|  | children: <Widget>[ |
|  | ElevatedButton( |
|  | onPressed: \_insert, |
|  | child: const Text('insert'), |
|  | ), |
|  | const SizedBox(height: 10), |
|  | ElevatedButton( |
|  | onPressed: \_query, |
|  | child: const Text('query'), |
|  | ), |
|  | const SizedBox(height: 10), |
|  | ElevatedButton( |
|  | onPressed: \_update, |
|  | child: const Text('update'), |
|  | ), |
|  | const SizedBox(height: 10), |
|  | ElevatedButton( |
|  | onPressed: \_delete, |
|  | child: const Text('delete'), |
|  | ), |
|  | ], |
|  | ), |
|  | ), |
|  | ); |
|  | } |
|  |  |
|  | // Button onPressed methods |
|  |  |
|  | void \_insert() async { |
|  | // row to insert |
|  | Map<String, dynamic> row = { |
|  | DatabaseHelper.columnName: 'Bob', |
|  | DatabaseHelper.columnAge: 23 |
|  | }; |
|  | final id = await dbHelper.insert(row); |
|  | debugPrint('inserted row id: $id'); |
|  | } |
|  |  |
|  | void \_query() async { |
|  | final allRows = await dbHelper.queryAllRows(); |
|  | debugPrint('query all rows:'); |
|  | for (final row in allRows) { |
|  | debugPrint(row.toString()); |
|  | } |
|  | } |
|  |  |
|  | void \_update() async { |
|  | // row to update |
|  | Map<String, dynamic> row = { |
|  | DatabaseHelper.columnId: 1, |
|  | DatabaseHelper.columnName: 'Mary', |
|  | DatabaseHelper.columnAge: 32 |
|  | }; |
|  | final rowsAffected = await dbHelper.update(row); |
|  | debugPrint('updated $rowsAffected row(s)'); |
|  | } |
|  |  |
|  | void \_delete() async { |
|  | // Assuming that the number of rows is the id for the last row. |
|  | final id = await dbHelper.queryRowCount(); |
|  | final rowsDeleted = await dbHelper.delete(id); |
|  | debugPrint('deleted $rowsDeleted row(s): row $id'); |
|  | } |
|  | } |

**When you are done!!**

When you finish, you should be able to run the app. Press the buttons and observe the logged output from the print statements. In Android Studio, ensure you have the Run tab selected.

Review the code comments to understand the functionality of different parts.

For a production app, it's recommended not to mix your UI code with your database code. Implement a state management layer to handle UI events and a service layer for your database operations.

**What to Do in Part 2:**

This task will enhance the app's functionality by allowing users to find a specific record by its ID and delete all records from the database.

**Hint:** Create a function to query a specific row from the database based on its ID.

1. **Query by ID:**
   * Create a function to search for a specific record in the database using its ID number.
   * Add this function to the DatabaseHelper class.
2. **Delete All Records:**
   * Create a function to remove all records from the database.

**How to View Debug Output and Interact with Your App:**

**View Debug Output:**

* The terminal will display the debug output, including any debug Print statements from your code. You can see the results of your \_insert, \_query, \_update, and \_delete methods here.

**Interacting with the App:**

* If you have a connected device or emulator, you can interact with your app and press the buttons to trigger the database operations. The results will be printed in the terminal.

Setting up your architecture this way makes it easier to swap out another database solution without breaking your app.

Here are some popular alternatives:

?

* [isar](https://pub.dev/packages/isar)
* [hive](https://pub.dev/packages/hive)
* [moor](https://pub.dev/packages/moor)
* [floor](https://pub.dev/packages/floor)

**Resources**

* [SQFlite documentation](https://pub.dartlang.org/packages/sqflite)
* [Using sqflite in your Flutter application effectively](https://steemit.com/programming/@tstieff/using-sqflite-in-your-flutter-applicaiton-effectively)
* [Flutter SQLite example — CRUD operations with sqflite plugin](https://grokonez.com/flutter/flutter-sqlite-example-crud-sqflite-example)
* [SQFlite Database in Flutter](https://medium.com/@rajajawahar77/sqflite-database-in-flutter-c0b7be83bcd2)