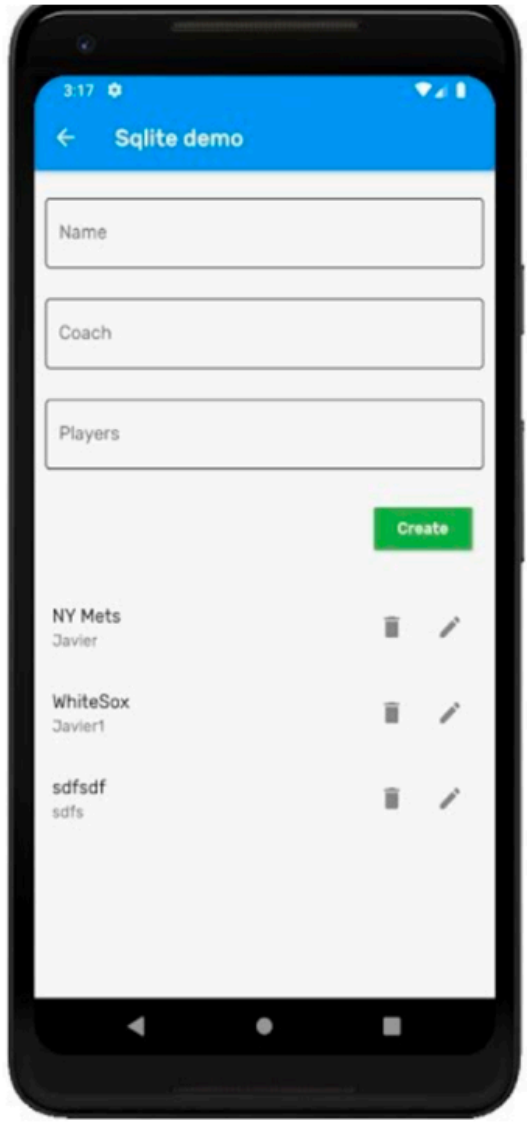
Resource Software Solution

Persist data with SQLite

Introduction

- If you are writing an app that needs to persist and query large amounts of **data** on the **local device**. In general, databases provide faster **inserts**, **updates**, and **queries** compared to other local persistence solutions.
- Flutter apps can make use of the SQLite databases via the **sqflite** plugin. This lesson demonstrates the basics of using sqflite to **insert**, **read**, **update**, and **remove** data.



Introduction

- This lesson uses the following steps:
  1. Add the dependencies.
  2. Define the data model.
  3. Open the database and create table.
  4. Insert.
  5. Select.
  6. Update.
  7. Delete.

Add the dependencies

- To work with SQLite databases, import the **sqflite** package.
- The **sqflite** package provides classes and functions to **interact with a SQLite database**.

```
dependencies:
  flutter:
    sdk: flutter

  english_words: ^4.0.0

  sqflite: ^2.1.0+1
```

Define the Student data model

- To define the **data** that needs to be stored.
- For this example, define a **Student** class that contains three pieces of data: A unique **id**, the **name**, and the **age** of each student.

```dart
class Student {
  final int id;
  final String name;
  final int age;

  const Student(this.id, this.name, this.age);

  Map<String, dynamic> toMap() {
    return {
      'id' : id,
      'name' : name,
      'age' : age,
    };
  }
}
```

Open the database

- Before reading and writing data to the database, open a connection to the database.
- Open the database with the **openDatabase()** function.

```
final database = openDatabase(
  'demo.db',
   version: 1,
  // When the database is first created, create a table to store student.
  onCreate: (db, version) {
    // Run the CREATE TABLE statement on the database.
    return db.execute(
      'CREATE TABLE student(id INTEGER PRIMARY KEY, name TEXT, age INTEGER)',
    );
  },
);
```

Datatypes in SQLite

- Each value stored in an SQLite database (or manipulated by the database engine) has one of the following storage classes:
  1. **INTEGER**. The value is a signed integer.
  2. **REAL**. The value is a floating point value.
  3. **TEXT**. The value is a text string

CREATE TABLE statement

- To create a new table in SQLite, you use **CREATE TABLE** statement using the following syntax:

```
CREATE TABLE table_name (
    column_1 data_type PRIMARY KEY,
    column_2 data_type NOT NULL,
    column_3 data_type DEFAULT VALUE,
    …
);
```

Insert

- Use the **insert()** method to store the Map in the student table.

```
// Define a function that inserts student into the database
Future<void> insertStudent(Student student) async {
  // Get a reference to the database.
  final db = await database;

  await db.insert(
    'student',
    student.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

Select

- Use the **query()** method. This returns a **List<Map>**

```
// A method that retrieves all the students from the student table.
Future<List<Map<String, dynamic>> getStudents() async {
  // Get a reference to the database.
  final db = await database;

  // Query the table for all the Students.
  return db.query('student');
}
```

Update

- Using the **update()** method from the sqflite library. Use a **where** clause to ensure you update the correct Student.

```
Future<void> updateStudent(Student student) async {
  // Get a reference to the database.
  final db = await database;

  await db.update(
    'student',
    student.toMap(),
    where: 'id = ?', // Ensure that the Student has a matching id.
    whereArgs: [student.id],
  );
}
```

Delete

- To delete data, use the **delete()** method from the sqflite library.

```
Future<void> deleteStudent(int id) async {
  // Get a reference to the database.
  final db = await database;

  // Remove the Student from the database.
  await db.delete(
    'student',
    where: 'id = ?',
    // Pass the Student's id as a whereArg.
    whereArgs: [id],
  );
}
```