

Persistencia de datos con SQLite | FLUTTER

Siempre que necesite almacenar los datos en el local, debe usar SQLite para conservar los datos del usuario.

**¿Por qué persistencia de datos?**

La persistencia de datos es importante para los usuarios, ya que sería un inconveniente para ellos escribir su información cada vez o esperar a que la red cargue los mismos datos nuevamente. En situaciones como esta, sería mejor guardar sus datos localmente.

**¿Por qué usamos SQLite?**

SQLite es una opción popular como software de base de datos integrada para almacenamiento local / de cliente en software de aplicación, como los navegadores web.

**¿Cómo usar SQLite en un flutter?**

Antes de usar SQLite, debes saber que

SQLite no está disponible en un SDK de aleteo como Android, pero tenemos un plugin sqflite que realiza exactamente todas las operaciones en la base de datos, al igual que en

**Android e iOS.**

El complemento Flutter es el contenedor del código nativo escrito en Kotlin o java para Android y swift o object-c para iOS.

P.S: el complemento también se puede crear solo con el código de dardo

Si es nuevo en SQLite, consulte el sitio del Tutorial de SQLite aquí.

Implementación

**Paso 1: Añade las dependencias**



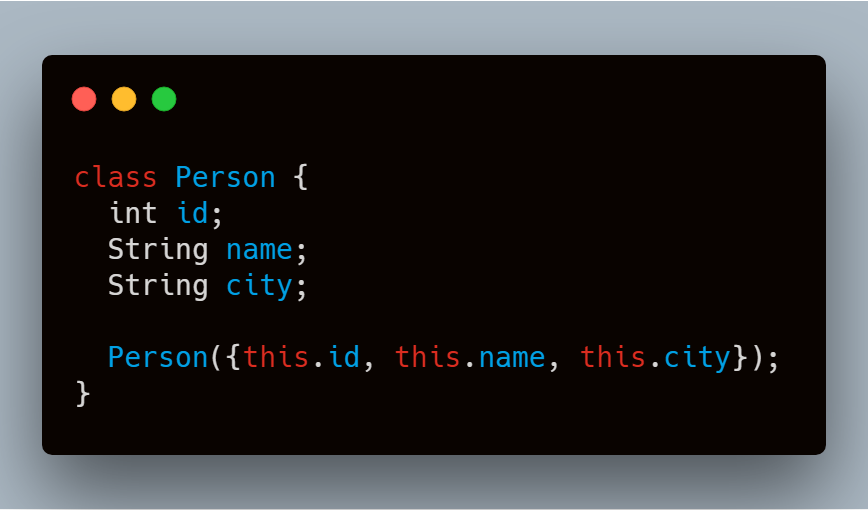
Para trabajar con bases de datos SQLite, importe los paquetes sqflite y path.

1. El paquete sqflite proporciona clases y funciones que le permiten interactuar con una base de datos SQLite.

2. El paquete path\_provider proporciona funciones que le permiten definir correctamente la ubicación para almacenar la base de datos en un disco como TemporaryDirectory y ApplicationDocumentsDirectory.

**Step 2: Create a Model class**

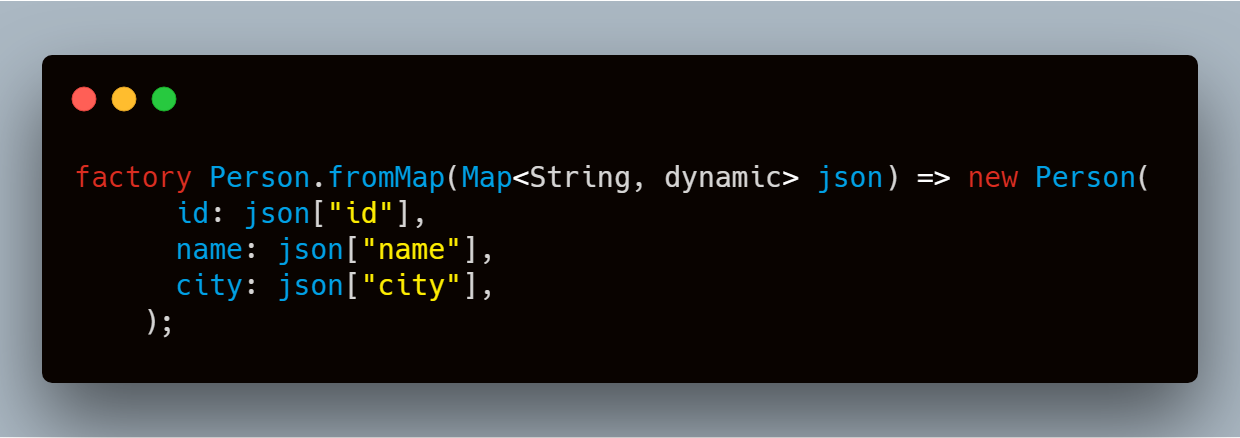
SQLite creates a table for the model class, the fields in the class correspond to columns in the table. Therefore, the classes tend to be small model classes that don’t contain any logic. Our Person class represents the model for the data in the database.



If we want to insert into the database then we need to convert the Person into a Map



And if we want to retrieve from the database then we need to convert the Map into the Person



This is how our PODO class will look like

|  |  |
| --- | --- |
|  | class Person { |
|  | int id; |
|  | String name; |
|  | String city; |
|  |  |
|  | Person({this.id, this.name, this.city}); |
|  |  |
|  | factory Person.fromMap(Map<String, dynamic> json) => new Person( |
|  | id: json["id"], |
|  | name: json["name"], |
|  | city: json["city"], |
|  | ); |
|  |  |
|  | Map<String, dynamic> toMap() => { |
|  | "id": id, |
|  | "name": name, |
|  | "city": city, |
|  | }; |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/d74ea7944b53f193832793c1fe5f54a9/raw/e8f72ec91ec8bd8627b9acf31282b162c0c01041/person.dart)[**person.dart**](https://gist.github.com/ashishrawat2911/d74ea7944b53f193832793c1fe5f54a9#file-person-dart) hosted with  by **[GitHub](https://github.com/)**

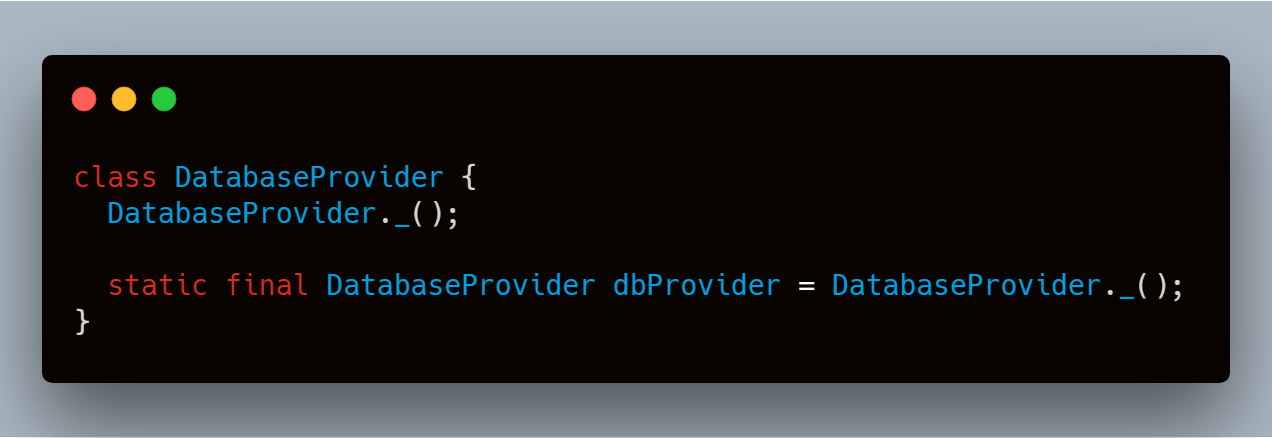
**Step 3: Create a database**

We will make a separate class as database.dart to make the code more modular and add all the requirements meths for managing the database that can be accessed anywhere in the app.

Create a singleton class DatabaseProvider

**Why Singleton?**We use the singleton pattern to ensure that we have only one class instance and provide global point access to it.

**How to create a Singleton in Dart?**Create a private constructor that can be used only inside the class.



**Step 4: Open the Database**

Before you read and write data to the database, you need to open a connection to the database. This involves two steps:

1. Define the path to the database file using the getDatabasesPath from the sqflite package combined with the pathfunction from the path package  
2. Open the database with the openDatabase function from sqflite



**Step 5: Create the table**

You need to create a table to store information.  
For example, In our case, we create a table called Person that defines the data that can be stored. In this case, each Person contains an id, name, and city. Therefore, these will be represented as three columns in the Person table.

1. The id is a Dart int, and will be stored as an INTEGER SQLite Datatype. It is also good practice to use an id as the primary key for the table to improve query and update times.

2. The name is a Dart String, and will be stored as a TEXT SQLite Datatype

3. The city is also a Dart String, and will be stored as an TEXT Datatype



To prevent from opening a connection to the database again and again we can use this:

|  |  |
| --- | --- |
|  | Future<Database> get database async { |
|  | if (\_database != null) return \_database; |
|  | \_database = await getDatabaseInstance(); |
|  | return \_database; |
|  | } |
|  |  |
|  | Future<Database> getDatabaseInstance() async { |
|  | Directory directory = await getApplicationDocumentsDirectory(); |
|  | String path = join(directory.path, "person.db"); |
|  | return await openDatabase(path, version: 1, |
|  | onCreate: (Database db, int version) async { |
|  | await db.execute("CREATE TABLE Person (" |
|  | "id integer primary key AUTOINCREMENT," |
|  | "name TEXT," |
|  | "city TEXT" |
|  | ")"); |
|  | }); |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/a59ab9ece9649d1de3dc9527b870c99b/raw/8f0a603e5146396480cc146706036af9a7451d2f/createdatabase.dart)[**createdatabase.dart**](https://gist.github.com/ashishrawat2911/a59ab9ece9649d1de3dc9527b870c99b#file-createdatabase-dart) hosted with  by **[GitHub](https://github.com/)**

**Step 6: Managing the data**

Now we are going to show you how you can perform an operation on the SQLite database.

**Query:**

|  |  |
| --- | --- |
|  | //return all persons from the database |
|  | Future<List<Person>> getAllPersons() async { |
|  | final db = await database; |
|  | var response = await db.query("Person"); |
|  | List<Person> list = response.map((c) => Person.fromMap(c)).toList(); |
|  | return list; |
|  | } |
|  |  |
|  | //return single person with id |
|  | Future<Person> getPersonWithId(int id) async { |
|  | final db = await database; |
|  | var response = await db.query("Person", where: "id = ?", whereArgs: [id]); |
|  | return response.isNotEmpty ? Person.fromMap(response.first) : null; |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/333a6b650568b9b59468a2e70e33175b/raw/b19e0777aed123605e61d4998e7bd87d50771e64/query.dart)[**query.dart**](https://gist.github.com/ashishrawat2911/333a6b650568b9b59468a2e70e33175b#file-query-dart) hosted with  by **[GitHub](https://github.com/)**

getAllPersons() will return all the person from the SQLite database if available.

**Insert:**

|  |  |
| --- | --- |
|  | addPersonToDatabase(Person person) async { |
|  | final db = await database; |
|  | var raw = await db.insert( |
|  | "Person", |
|  | person.toMap(), |
|  | conflictAlgorithm: ConflictAlgorithm.replace, |
|  | ); |
|  | return raw; |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/2ed7bcdf77d253d4940bcf0681afbc6d/raw/cf251b569c1098c02a54138ec06babeecd07d71f/insert.dart)[**insert.dart**](https://gist.github.com/ashishrawat2911/2ed7bcdf77d253d4940bcf0681afbc6d#file-insert-dart) hosted with  by **[GitHub](https://github.com/)**

**Delete:**

|  |  |
| --- | --- |
|  | //delete person with id |
|  | deletePersonWithId(int id) async { |
|  | final db = await database; |
|  | return db.delete("Person", where: "id = ?", whereArgs: [id]); |
|  | } |
|  |  |
|  | //delete all persons |
|  | deleteAllPersons() async { |
|  | final db = await database; |
|  | db.delete("Person"); |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/5969e225083a5c8ff995618fdab95a01/raw/52e73642ba5236700ca73541bae99267f47590f9/delete.dart)[**delete.dart**](https://gist.github.com/ashishrawat2911/5969e225083a5c8ff995618fdab95a01#file-delete-dart) hosted with  by **[GitHub](https://github.com/)**

If you see we have two methods, one deletes the row with particular id and other deletes all data present in the table, you can change all the query according to your need.

**Update:**

|  |  |
| --- | --- |
|  | updatePerson(Person person) async { |
|  | final db = await database; |
|  | var response = await db.update("Person", person.toMap(), |
|  | where: "id = ?", whereArgs: [person.id]); |
|  | return response; |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/f25acd6599b120bfa3e1d68a58b4430d/raw/3a6afd86bf5a13b534279b17fed800b0bce3f4e6/update.dart)[**update.dart**](https://gist.github.com/ashishrawat2911/f25acd6599b120bfa3e1d68a58b4430d#file-update-dart) hosted with  by **[GitHub](https://github.com/)**

If these small code snippets still confuse you we have the complete code of the database class:

|  |  |
| --- | --- |
|  | import 'dart:async'; |
|  | import 'dart:io'; |
|  |  |
|  | import 'package:path/path.dart'; |
|  | import 'package:path\_provider/path\_provider.dart'; |
|  | import 'package:flutter\_sqlite/person.dart'; |
|  | import 'package:sqflite/sqflite.dart'; |
|  |  |
|  | class PersonDatabaseProvider { |
|  | PersonDatabaseProvider.\_(); |
|  |  |
|  | static final PersonDatabaseProvider db = PersonDatabaseProvider.\_(); |
|  | Database \_database; |
|  |  |
|  | Future<Database> get database async { |
|  | if (\_database != null) return \_database; |
|  | \_database = await getDatabaseInstance(); |
|  | return \_database; |
|  | } |
|  |  |
|  | Future<Database> getDatabaseInstance() async { |
|  | Directory directory = await getApplicationDocumentsDirectory(); |
|  | String path = join(directory.path, "person.db"); |
|  | return await openDatabase(path, version: 1, |
|  | onCreate: (Database db, int version) async { |
|  | await db.execute("CREATE TABLE Person (" |
|  | "id integer primary key AUTOINCREMENT," |
|  | "name TEXT," |
|  | "city TEXT" |
|  | ")"); |
|  | }); |
|  | } |
|  |  |
|  | addPersonToDatabase(Person person) async { |
|  | final db = await database; |
|  | var raw = await db.insert( |
|  | "Person", |
|  | person.toMap(), |
|  | conflictAlgorithm: ConflictAlgorithm.replace, |
|  | ); |
|  | return raw; |
|  | } |
|  |  |
|  | updatePerson(Person person) async { |
|  | final db = await database; |
|  | var response = await db.update("Person", person.toMap(), |
|  | where: "id = ?", whereArgs: [person.id]); |
|  | return response; |
|  | } |
|  |  |
|  | Future<Person> getPersonWithId(int id) async { |
|  | final db = await database; |
|  | var response = await db.query("Person", where: "id = ?", whereArgs: [id]); |
|  | return response.isNotEmpty ? Person.fromMap(response.first) : null; |
|  | } |
|  |  |
|  | Future<List<Person>> getAllPersons() async { |
|  | final db = await database; |
|  | var response = await db.query("Person"); |
|  | List<Person> list = response.map((c) => Person.fromMap(c)).toList(); |
|  | return list; |
|  | } |
|  |  |
|  | deletePersonWithId(int id) async { |
|  | final db = await database; |
|  | return db.delete("Person", where: "id = ?", whereArgs: [id]); |
|  | } |
|  |  |
|  | deleteAllPersons() async { |
|  | final db = await database; |
|  | db.delete("Person"); |
|  | } |
|  | } |

[**view raw**](https://gist.github.com/ashishrawat2911/16c4d20501f8bec519b914ee6e49480d/raw/01e012a4712dbacf264f21aeafa5ee8018db8174/database.dart)[**database.dart**](https://gist.github.com/ashishrawat2911/16c4d20501f8bec519b914ee6e49480d#file-database-dart) hosted with  by **[GitHub](https://github.com/)**

**Step 7: Using the data**

In order to use the database we need to create the instance of the database and use the method present in the database class

DatabaseProvider.*db*

This will help us to perform an operation on the database.

like if we want to get  the person in the database we will use the method that we have defined in our DatabaseProvider class

DatabaseProvider.*db*.getAllPersons()

And if I want to display it in the list then I’ll use FututeBuilder :

|  |  |
| --- | --- |
|  | FutureBuilder<List<Person>>( |
|  | future: PersonDatabaseProvider.db.getAllPersons(), |
|  | builder: (BuildContext context, AsyncSnapshot<List<Person>> snapshot) { |
|  | if (snapshot.hasData) { |
|  | return ListView.builder( |
|  | physics: BouncingScrollPhysics(), |
|  | itemCount: snapshot.data.length, |
|  | itemBuilder: (BuildContext context, int index) { |
|  | Person item = snapshot.data[index]; |
|  | return Dismissible( |
|  | background: Container(color: Colors.red), |
|  | onDismissed: (direction) { |
|  | PersonDatabaseProvider.db.deletePersonWithId(item.id); |
|  | }, |
|  | child: ListTile( |
|  | title: Text(item.name), |
|  | subtitle: Text(item.city), |
|  | leading: CircleAvatar(child: Text(item.id.toString())), |
|  | ), |
|  | ); |
|  | }, |
|  | ); |
|  | } else { |
|  | return Center(child: CircularProgressIndicator()); |
|  | } |
|  | }, |
|  | ), |

[**view raw**](https://gist.github.com/ashishrawat2911/fdf814c9638976a57d440329a318257f/raw/7e74bfe5ebf379296b7fe03c3d753aa310a08b7c/bodysql.dart)[**bodysql.dart**](https://gist.github.com/ashishrawat2911/fdf814c9638976a57d440329a318257f#file-bodysql-dart) hosted with  by **[GitHub](https://github.com/)**

That it for SQLite in flutter.