



# Persistir datos con SQLite

## 1. Adicionar dependencias

<https://pub.dev/packages/path#-installing-tab->

<https://pub.dev/packages/sqlite#-installing-tab->

The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows the project structure with files like .history, .idea, android, ios, lib, src, main.dart, test, widget\_test.dart, .gitignore, .metadata, .packages, pubspec.lock, and pubspec.yaml. The main editor shows the pubspec.yaml file with the following content:

```
14
15 dependencies:
16   flutter:
17     sdk: flutter
18
19 # The following adds the Cupertino Icons font to your application.
20 # Use with the CupertinoIcons class for iOS style icons.
21/cupertino_icons: ^0.1.2
22
23 sqlite: ^1.1.5
24 path: ^1.6.2
25
26 dev_dependencies:
27   flutter_test:
28     sdk: flutter
29
```

The bottom panel shows the Flutter output window with the following error message:

```
Because sqlitelab depends on sqlite >=1.1.2 which requires SDK version >=2.1.0 <3.0.0,
version solving failed.

pub get failed (1)
exit code 1
```

## 2. Definir modelo de datos

```
/lib/src/alumno.dart  
class Alumno {  
  final int id;  
  final String nombres;  
  final String apellidos;  
  final String doc_identidad;  
  
  Alumno({this.id, this.nombres, this.apellidos, this.doc_identidad});  
  
  Map<String, dynamic> toMap() {  
    return {  
      'id': id,  
      'nombres': nombres,  
      'apellidos': apellidos,  
      'doc_identidad' : doc_identidad  
    };  
  }  
}
```

## 3. Abrir la base de datos

```
final database = openDatabase(  
  join(await getDatabasesPath(), 'alumnos.db'),  
  onCreate: (db, version) {  
    };  
  },  
  version: 1,  
);
```



## 4. Crear un table en SQLITE

```
final database = openDatabase(  
  join(await getDatabasesPath(), 'alumnos.db'),  
  onCreate: (db, version) {  
    return db.execute(  
      "CREATE TABLE alumnos(id INTEGER PRIMARY KEY AUTOINCREMENT, nombres  
TEXT, apellidos TEXT, doc_identidad TEXT)",  
    );  
  },  
  version: 1,  
);
```

## 5. Insertar datos en una table de SQLITE

```
Future<void> insertaAlumno(Alumno alumno) async {  
  
  final Database db = await database;  
  
  await db.insert(  
    'alumnos',  
    alumno.toMap(),  
    conflictAlgorithm: ConflictAlgorithm.replace,  
  );  
}
```

## 6. Recuperar datos de una tabla de SQLITE

```
Future<List<Alumno>> alumnos() async {  
  
    final Database db = await database;  
  
    final List<Map<String, dynamic>> maps = await db.query('alumnos');  
  
    return List.generate(maps.length, (i) {  
        return Alumno(  
            id: maps[i]['id'],  
            nombres: maps[i]['nombres'],  
            apellidos: maps[i]['apellidos'],  
            doc_identidad: maps[i]['doc_identidad'],  
        );  
    });  
}
```

## 7. Actualizar datos de una tabla de SQLITE

```
Future<void> actualizaalumno(Alumno alumno) async {  
  
    final db = await database;  
  
    await db.update(  
        'alumnos',  
        alumno.toMap(),  
        where: "id = ?",  
        whereArgs: [alumno.id],  
    );  
}
```



## 8. Borrar datos de una tabla de SQLITE

```
Future<void> eliminaAlumno(int id) async {  
  final db = await database;  
  
  await db.delete(  
    'alumnos',  
    where: "id = ?",  
    whereArgs: [id],  
  );  
}
```

/lib/main.dart

```
import 'dart:async';
import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import 'package:sqlitedcrud/alumno.dart';

void main() async {

  final database = openDatabase(
    join(await getDatabasesPath(), 'alumnos.db'),
    onCreate: (db, version) {
      return db.execute(
        "CREATE TABLE alumnos(id INTEGER PRIMARY KEY AUTOINCREMENT, nombres TEXT, apellidos TEXT, doc_identidad TEXT)",
      );
    },
    version: 1,
  );

  Future<void> insertaAlumno(Alumno alumno) async {

    final Database db = await database;

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



```
Future<List<Alumno>> alumnos() async {

    final Database db = await database;

    final List<Map<String, dynamic>> maps = await db.query('alumnos');

    return List.generate(maps.length, (i) {
        return Alumno(
            id: maps[i]['id'],
            nombres: maps[i]['nombres'],
            apellidos: maps[i]['apellidos'],
            doc_identidad: maps[i]['doc_identidad'],
        );
    });
}

Future<void> actualizaalumno(Alumno alumno) async {

    final db = await database;

    await db.update(
        'alumnos',
        alumno.toMap(),
        where: "id = ?",
        whereArgs: [alumno.id],
    );
}

Future<void> eliminaAlumno(int id) async {
    final db = await database;

    await db.delete(
        'alumnos',
        where: "id = ?",
        whereArgs: [id],
    );
}
```

```
var jorge = Alumno(  
  id: 0,  
  nombres: 'Jorge',  
  apellidos: 'Chavez',  
  doc_identidad: '123456'  
);  
  
await insertaAlumno(jorge);  
  
print(await alumnos());  
  
jorge = Alumno(  
  id: jorge.id,  
  nombres: jorge.nombres,  
  apellidos: jorge.apellidos,  
  doc_identidad: '22222222'  
);  
await actualizaalumno(jorge);  
  
print(await alumnos());  
  
await eliminaAlumno(jorge.id);  
  
print(await alumnos());  
}
```





**Preparando para release una app Android**

<https://flutter-es.io/docs/deployment/android>

# **flutter build apk**

**Preparando para release una app iOS**

<https://flutter-es.io/docs/deployment/ios>

# **flutter build ios**