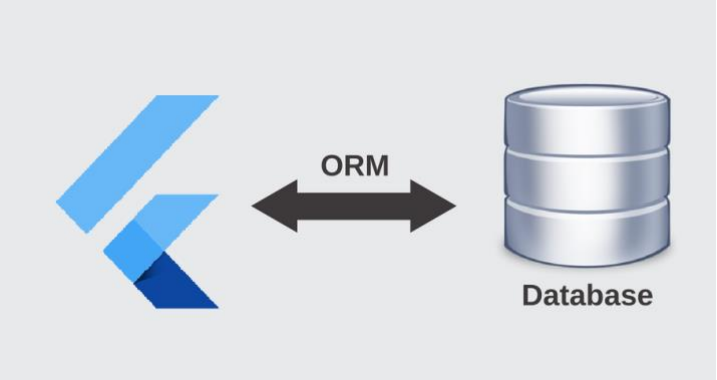
Floor Database: Saving Data in Local Database

Floor provides a neat SQLite abstraction for your Flutter applications inspired by the Room persistence library. It comes with automatic mapping between in-memory objects and database rows while still offering full control of the database with the use of SQL. As a consequence, it's necessary to have an understanding of SQL and SQLite in order to harvest Floor's full potential.

- null-safe
- typesafe
- reactive
- lightweight
- SQL centric
- no hidden magic
- no hidden costs
- iOS, Android, Linux, macOS, Windows



- ORM is very important to when you are using a database in your application, I started looking to ORM available in flutter and I found sqfentity, moor, floor, etc.
- I started looking into them and I found the floor is quite easy if you are familiar with Room android ORM, that's why I decide to work on the floor ORM and I found it is easy to implement with minimum code.

STEP 1

Add the runtime dependency `floor` as well as the generator`floor_generator` to your `pubspec.yaml`. The third dependency is `build_runner` which has to be included as a dev dependency just like the generator.

- `floor` holds all the code you are going to use in your application.
- `floor_generator` includes the code for generating the database classes.
- `build_runner` enables a concrete way of generating source code files.

```
dependencies:
  flutter:
    sdk: flutter
  floor: ^0.13.0 dev_dependencies:
  floor_generator: ^0.13.0
  build_runner: ^1.7.3
```

STEP 2

Create an **Entity**

It will represent a database table as well as the scaffold of your business object. `@entity` marks the class as a persistent class. It's required to add a primary key to your table. You can do so by adding the `@primaryKey` annotation to an `int` property. There is no restriction on where you put the file containing the entity.

```
import 'package:floor/floor.dart'; @entity
class Todo
{
@PrimaryKey(autoGenerate: true)
final int id;
final String task;
final String time;
final String scheduleTime;
@ignore
bool isSelect = false; Todo(this.id,this.task,this.time,this.scheduleTime);
}
```

STEP 3

Create a **DAO (Data Access Object)**

This component is responsible for managing access to the underlying SQLite database. The abstract class contains the method signatures for querying the database which have to return a `Future` or `Stream`.

- You can define queries by adding the `@Query` annotation to a method. The SQL statement has to get added in parenthesis. The method must return a `Future` or `Stream` of the `Entity` you're querying for.

- `@insert` marks a method as an insertion method.

```
import 'package:floor/floor.dart';
import 'package:todolist/src/model/todo.dart';
@dao
abstract class TodoDao
{
@Query('SELECT * FROM todo')
Future<List<Todo>> findAllTodo();

@Query('Select * from todo order by id desc limit 1')
Future<Todo> getMaxTodo();

@Query('SELECT * FROM todo order by id desc')
Stream<List<Todo>> fetchStreamData();

@insert
Future<void> insertTodo(Todo todo);

@insert
Future<List<int>> insertAllTodo(List<Todo> todo);

@Query("delete from todo where id = :id")
Future<void> deleteTodo(int id);

@delete
Future<int> deleteAll(List<Todo> list);}
```

STEP 4

Create the **Database**

It has to be an abstract class which extends `FloorDatabase`. Furthermore, it's required to add `@Database()` to the signature of the class. Make sure to add the created entity to the `entities` attribute of the `@Database` annotation.

In order to make the generated code work, it’s required to also add the listed imports.

```
import 'dart:async';
import 'package:floor/floor.dart';
import 'package:todolist/src/dao/todo_dao.dart';
import 'package:todolist/src/model/todo.dart';
import 'package:sqflite/sqflite.dart' as sqflite;
import 'package:path/path.dart'; part 'app_database.g.dart'; // the generated code will be there @Database(version: 1,entities: [Todo])
abstract class AppDatabase extends FloorDatabase
{
  TodoDao get todoDao;
}
```

part ‘app_database.g.dart’;

It will auto-generate the file which contains all the queries of the dao.

For generating the file run below in terminal.

1. flutter packages pub run build_runner build

2. flutter packages pub run build_runner watch

Run the generator with `flutter packages pub run build_runner build`. To automatically run it, whenever a file changes, use `flutter packages pub run build_runner watch`

STEP 5

use the Dao's functions for insert and delete.

```
final database=$FloorAppDatabase.databaseBuilder('tododatabase.db').build();
database.then((onValu){
// find the dao here
onValu.todoDao.getMaxTodo().then((onValue){
// max id
int maxId = onValue
}});
```

From implementing the above 5 steps you can able to create/edit database in a flutter.

There are some profit and the loss I found on it.

Profit

- less code.
- if you are familiar with the room database easy to understand.

Loss

- every time change in database/entity you have run generator code in the terminal.

- it will take time to improve with a large-size database currently it is fine with a small number of tables.

GitHub link for above code:

https://github.com/khambhaytajaydip/Floor_flutter_ORM.git

More references: GitHub link: https://github.com/samsunk/demo_database1.git