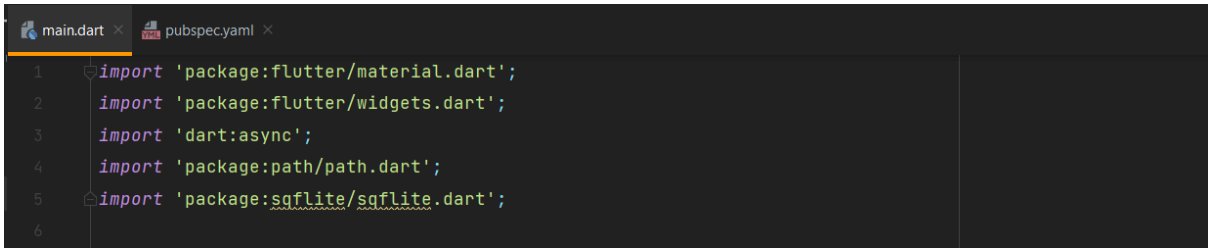
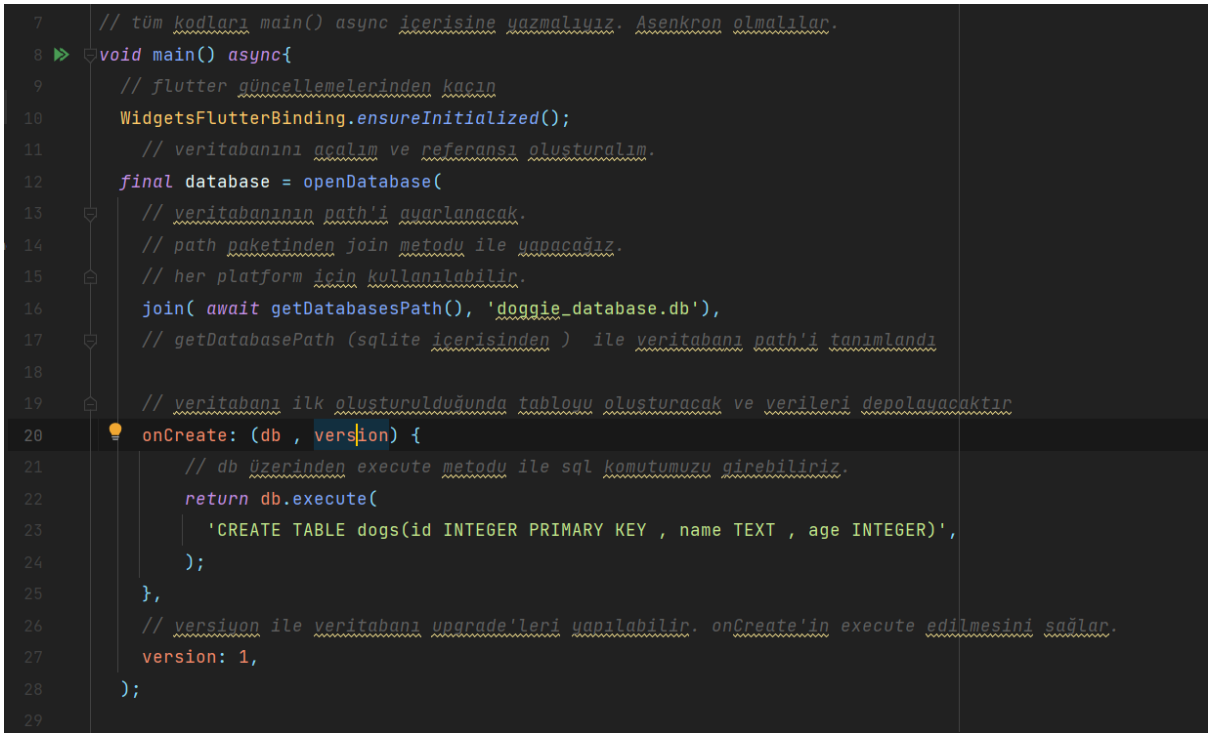
# Ödev 9

| | |
|---|---|
| Created | @December 29, 2021 1:55 PM |
| Açıklama | |
| Link | |

Uygulamanın Youtube Linki :




import etmemiz gereken paketleri ekleyelim.



```dart
// tüm kodları main() async içerisine yazmalıyız. Asenkron olmalılar.
void main() async{
  // flutter güncellemelerinden kaçın
  WidgetsFlutterBinding.ensureInitialized();
    // veritabanını açalım ve referansı oluşturalım.
  final database = openDatabase(
    // veritabanının path'i ayarlanacak.
    // path paketinden join metodu ile yapacağız.
    // her platform için kullanılabilir.
    join( await getDatabasesPath(), 'doggie_database.db'),
    // getDatabasePath (sqlite içerisinden )  ile veritabanı path'i tanımlandı

    // veritabanı ilk oluşturulduğunda tabloyu oluşturacak ve verileri depolayacaktır
    onCreate: (db , version) {
        // db üzerinden execute metodu ile sql komutumuzu girebiliriz.
        return db.execute(
          'CREATE TABLE dogs(id INTEGER PRIMARY KEY , name TEXT , age INTEGER)',
        );
    },
    // versiyon ile veritabanı upgrade'leri yapılabilir. onCreate'in execute edilmesini sağlar.
    version: 1,
  );

```

main fonksiyonumuz asenkron olmalı.

Widgets.FlutterBinding.ensureInitialized ile  flutter upgrade'lerinden kaçınmış olduk.

openDatabase metodu sqflite ile gelen bir metod ve yeni veritabanı oluşturmamıza yarar. Bu veritabanını database referansı ile tutuyoruz.

Veritabanını join metodu ile path'ini ayarlayacağız. ( join ,  path paketinden ) . getDatabasesPath metodu sqflite içerisinden ve bu metod ile path tanımlanmış oldu.

onCreate özelliğine db ve versiyon parametresi tanımlandı. db üzerinden veritabanına sorgu yazabileceğiz. Bu sorguyu db.execute ile tablo oluşturarak yazdık.

version bilgisi ise upgrade işlemlerine yaramaktadır. Ayrıca onCreate'ın execute edilmesini sağlar.

```dart
// Dog nesnelerini veritabanına ekleyen fonksiyon
Future<void> insertDog(Dog dog) async {
  // veritabanı referansı
  final db = await database;
  // doğru tabloya eklenmek için insert işlemi yapılır.
  // conflictAlgorithm, aynı nesne iki kere eklenmesi durumu için
  // bu durumda önceki verileri değiştirecek
  await db.insert(
    'dogs',
    dog.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

insertDog metodumuz Future tipinde asenkron bir metod. Dog tipinde nesneler alacak. Bu metod sayesinde veritabanına nesneleri ekleyebileceğiz.

database referansımızı db ile tutuyoruz. await ile işlemlerin senkronizasyonunu sağlıyoruz.

db.insert metodu ile verileri tabloya doğru şekilde , map türünde ,  ekleyebileceğiz. conflictAlgorithm sayesinde tekrarlı verilerin düzeltilmesi sağlanacak.

```dart

Future<List<Dog>> dogs() async {
  // referans tanımı
  final db = await database;
  // tüm 'dogs' için tabloyu sorgular.
  // bunu query ile kısaca yapmış olur.
  final List<Map<String, dynamic>> maps = await db.query('dogs');
  // dönen nesne kadar döngü oluşturulur
  return List.generate(maps.length, (i){
      return Dog(
        id   : maps[i]['id'],
        name : maps[i]['name'],
        age  : maps[i]['age'],
      );  // Dog
  });  // List.generate
}

```

Yine Future tipinde , Dog nesnelerini döndürecek listemizi oluşturan , dogs metodu asenkron bir şekilde tanımlandı.

db ile database referansı alındı.

db.query( ) metodu ile tabloya sorgu işlemleri yapabileceğiz. Bizim için kolaylık sağlayacak. Geri dönen nesne veya nesneler bir map yapısında olacağı için tipi map olan bir liste tanımladık.

List.generate ile bu listenin uzunluğu kadar bir döngü oluşturduk. Her bir iterasyon için Dog döndürülecek ve parametreleri liste içinden o iterasyondaki sıradan alınacak.

```dart
Future<void> updateDog(Dog dog) async {
  // referans tanımı
  final db = await database;

  await db.update(
    'dogs',
    dog.toMap(),
    // buradaki yapıyı SQL INJECTİON 'dan korunmak için kullanıyoruz.
    // !!!   Sunu kullanma : where: "id = ${dog.id}"   !!!
    where     : 'id = ?',
    whereArgs : [dog.id],
  );
}

```

Şimdi uptadeDog metodumuz var. Bu metotta yine asenkrondur. Parametre olarak verilen Dog nesnesini veritabanında güncellememizi sağlar.

Yine db ile database referansımızı aldık.

db üzerinden update( ) metodu ile veritabanı güncelleme işlemiini yapabileceğiz.

Burada önemli olan nokta where parametresine direk veriyi vermememiz gerektiği. Bu şekilde yaparsak SQL Injection ‘a açık bir sistem olabilir. O yüzden whereArgs özelliği üzerinden işlemimizi yapıyoruz

```dart
Future<void> deleteDog(int id) async {
  // referans tanımı
  final db = await database;
  // veritabanından veriyi sil

  await db.delete(
    'dogs',
    where     : 'id = ?',
    // yine sql injection'u engellemek için dog id bilgisini whereArgs olarak iletiriz.
    whereArgs : [id],
  )
}
```

Sırada deleteDog metoduuz var. Bu metod veritabanından veri silmemizi sağlar . Asenkron metoddur. Parametre olarak verilen id bilgisini siler.

db referansımızı tanımladık.

db.delete ile hangi verinin silineceğini tanımlayabiliriz.

Update işleminde olduğu gibi SQL Injection olmaması için where kısmına direk veriyi vermiyoruz. whereArgs üzerinden id bilgisini veriyoruz.

```
86
87      var fido = Dog(
88        id:0,
89        name: 'Fido',
90        age:10,
91      );
92      await insertDog(fido);  // fido nesnesi eklenir
93      print(await dogs());    // Simdi bastıralım
94
```

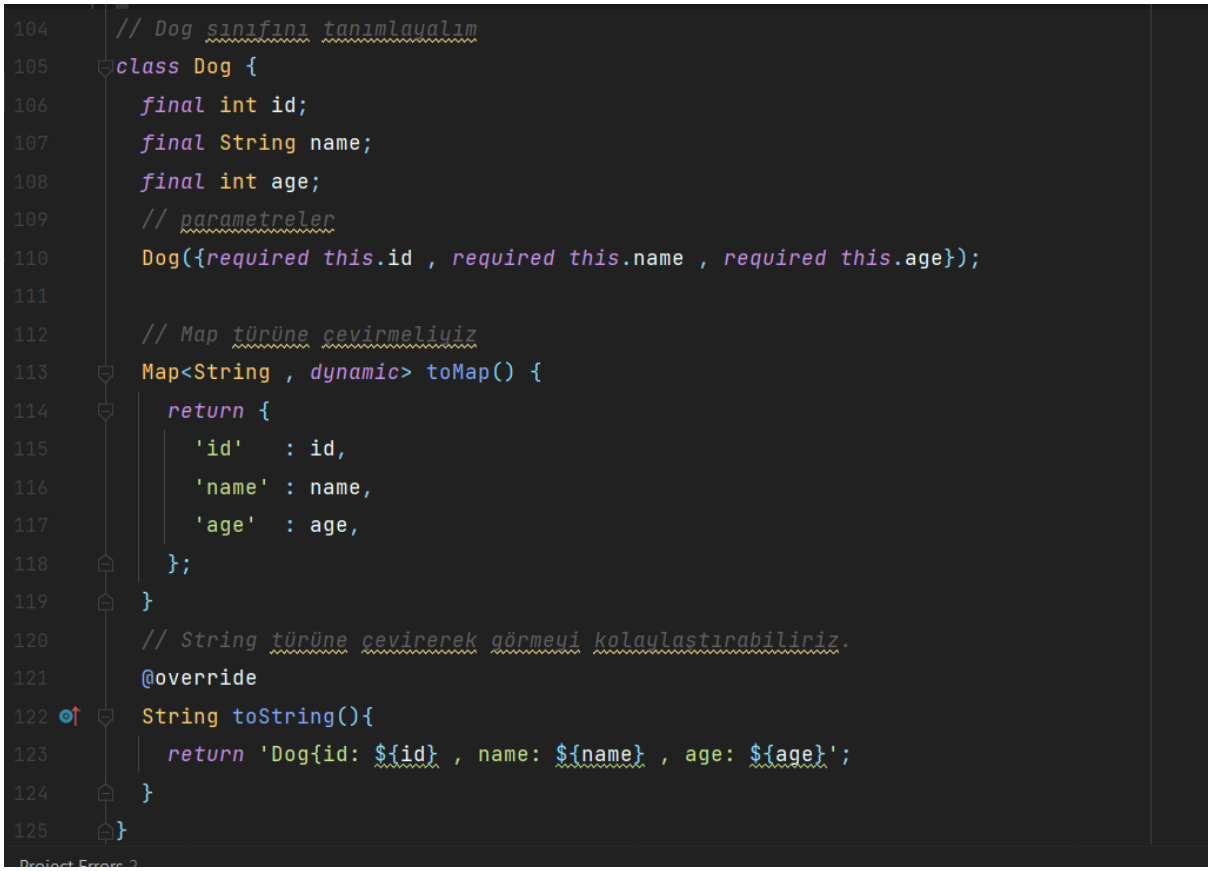
fido nesnemiz yaratıldı . Dog tipinde ( daha sonra tanımlanacak ) .

Gerekli parametreleri verildi.

insertDog metodumuz ile nesne veritabanına eklendi  ve son olarak ekrana bastırıldı.

```
94
95      fido = Dog(
96        id    : fido.id,
97        name  : fido.name,
98        age   : fido.age + 7,
99      );
100     await updateDog(fido);  // fido nesnesini güncelleyelim
101     print(await dogs())     // bastıralım
102
103 }
```

fido nesnemiz id ve name bilgileri aynı kalırken , age bilgisi değiştirildi ve işlem updateDog metodumuz ile veritabanında güncellendi. Son olarak ekrana bastırıldı.



```dart
// Dog sınıfını tanımlayalım
class Dog {
  final int id;
  final String name;
  final int age;
  // parametreler
  Dog({required this.id , required this.name , required this.age});

  // Map türüne çevirmeliyiz
  Map<String , dynamic> toMap() {
    return {
      'id'   : id,
      'name' : name,
      'age'  : age,
    };
  }
  // String türüne çevirerek görmeui kolaylaştırabiliriz.
  @override
  String toString(){
    return 'Dog{id: ${id} , name: ${name} , age: ${age}';
  }
}
```

Son olarak Dog sınıfımız. id , name ve age parametrelerine sahiptir ve constructor içerisinde bunları dışarıdan alır.

Hemen altında bir Map türünde return yapan toMap metodumuz var. Bu metod ile verilerimizi map türünde tutabileceğiz. Düzenli durması için yapılabilir.

En altta ise String türünde olan toString metodunu override ederek güncelledik. Verileri Map türünden tekrar String şeklinde görebilmemize yarayacak.