Görsel Programlama

SQLite ile lokal veritabanına bağlanma

Emir ÖZTÜRK

Veritabanı

- Dosyalama
- Verilerin saklanması
- Veri bütünlüğü
- Veri doğruluğu
- Verinin modellenmesi

Veriler dosyalarda saklanabilir ve yine dosyalar kullanılarak verilere erişilebilir. Fakat özellikle parçalı ve yapısal verinin dosyalarda saklanmasının bazı problemleri bulunmaktadır. İmplementasyonun zorluğu, veri bütünlüğü ve doğruluğunun sağlanması ve modellenmiş verinin saklama formatının belirlenmesi işlemlerinin her biri kendi içerisinde çözülmesi gereken sorunlar taşımaktadır.

İlişkisel Veritabanları

- Veriyi ayırt edici özellikler
- Özellikleri tanımlayan id
- Tablolar
- Tablolar arası ilişki
  - 1-1
  - 1-n
  - n-m

Veritabanlarında verilerin atomik olması gerekmektedir. Her veri grubunu tekil olarak belirlemek için id alanı belirlenebilir. Bu id alanlarının kullanılması ile belirli ilişkiler kurularak veritabanı tabloları mantıksal olarak bağlanabilmektedir.

Lokal veritabanı

- Veritabanı yönetim sistemleri
- Verinin veritabanı prensipleri ile lokalde saklanması
- Fazla büyük olmayan veriler
- Veri güvenliği

Veritabanları lokalde, servis olarak veya bulutta çalışabilirler. Veritabanı mantığının bulunduğu fakat servis olarak çalışmak yerine bir dosyaymış gibi çevrimdışı saklanan veritabanları da bulunmaktadır. Bu veritabanları fazla büyük olmayan yapısal verilerin güvenli bir şekilde saklanmasını amaçlarlar.

Lokal veritabanı

- MS Access
- SQL Server Express
- LevelDB
- SQLite

Lokal veritabanı örnekleri.

Lokal veritabanı

- SQLite bağlantısı için sqflite
- Dependency ekleme
  - sqflite

Flutter ile SQLite veritabanına bağlantı sağlanabilmektedir. Bunun için dependency olarak sqflite paketi eklenmelidir.

Veritabanı Bağlantısı
Model Oluşturma

- Veritabanı tablolarına karşılık gelen modeller
- Sınıf
- Id içermeli

```
class Kisi {
  int? id;
  String? ad;
  String? soz;
  Kisi(this.id, this.ad, this.soz);
  Kisi.fromMap(Map<String, dynamic> veri) {
    id = veri["id"];
    ad = veri["ad"];
    soz = veri["soz"];
  }
  Map<String,dynamic> toMap(){
    return {
        "id":id,
        "ad":ad,
        "soz":soz
    };
  }
}
```

Veritabanı alanlarına karşılık gelen sınıflar model olarak tanımlanır ve bu modeller veritabanından okuma ve veritabanına yazma esnasında kullanılabilmektedirler. Bu modellerin id içermesi gerekmektedir.

Veritabanı Bağlantısı
Veritabanı bağlantısını açma

- openDatabase()
- getDatabasesPath()
  - await edilmeli
- join(getDatabasesPath(),"Veritabani_dosya_adi.db")

```
static Future<Database> open() async {
  var db = await openDatabase("sozler.db");
```

openDatabase metodu ile veritabanı açılabilir. Açılan veritabanı erişim için bir değişken adına atanır. Ayrıca openDatabase async bir metottur. Bu veritabanının kapanması için Close() metodu kullanılır. Veritabanı yolunu almak için getDatabasesPath() metodu kullanılabilir. Bu metot cihazların veritabanını varsayılan olarak kaydettikleri yolu döndürmektedir.

Veritabanı Bağlantısı
Tablo Oluşturma

- Bir tablo birden fazla kayıt
- Kayıtların saklanacağı şemanın veritabanında oluşturulması
- DbFirst
- CodeFirst

Tablolar ve sql komutları kod içerisinden belirli metotlar ile çalıştırılabilmektedir.

Veritabanı Bağlantısı
Tablo Oluşturma

- openDatabase() içerisinde
- onCreate: (db,version){ return db.execute("SQL KOMUTU");}
- CREATE komutu
- CREATE TABLE tablo_adı(alan TÜR [PRIMARY KEY],alan TÜR, ...)
- CREATE TABLE kisiler(id INTEGER PRIMARY KEY,isim TEXT);

```
static Future<Database> open() async {
  var db = await openDatabase("sozler.db");
  //var db = await databaseFactoryFfi.openDatabase("sozler.db");
  await db.execute(
    "CREATE TABLE if not exists kisiler(id INTEGER PRIMARY KEY AU
  );
  return db;
}
```

Uygulama açıldığında eğer veritabanı yoksa oluştur demek için onCreate kullanılabilir. onCreate ile veritabanı içerisinde tablo oluşturma kodları çalıştırılır. Böylece eğer uygulama açıldığında veritabanı bulunmuyorsa onCreate tetiklenir.

Veritabanı Bağlantısı
Ekleme

- Ekleme işlemleri için sınıfta düzenleme
- Sınıfa toMap metodu eklenmesi
- Alan adlarına karşılık gelen değerlerin oluşturduğu bir map

```
Map<String,dynamic> toMap(){
  return {
    "id":id,
    "ad":ad,
    "soz":soz
  };
}
```

Sınıfların veritabanı kayıtlarına dönüştürülmesi veya tam tersi için modellerin içine toMap ve fromMap adlı iki metot tanımlanmalıdır. Map metotlarına veritabanı alan isimlerine karşılık gelen isimler atanmalıdır.

Veritabanı Bağlantısı
Ekleme

- Ekleme metodu
- Metot içerisinde db.insert("tablo_adı",nesne.toMap(),conflictAlgorithm);
- ConflictAlgorithm
  - Rollback
  - Abort
  - Fail
  - Ignore
  - Replace

```
static Future<int> ekle(Kisi eklenecek) async {
  var db = await open();
  return db.insert("kisiler", eklenecek.toMap());
}
```

Ekleme işlemi için insert metodu kullanılabilir. Bu metoda ilk parametre olarak tablo adı, ikinci parametre olarak da nesne verilir. Nesne verilmeden önce veritabanına yazılabilmesi için toMap metodu çağırılır. Conflictalgorithm ile herhangi bir çakışma ya da problemde ne yapacağı seçilir.

Veritabanı Bağlantısı
Güncelleme

- Db nesnesinin eldesi
  - Db = await database
- db.update metodu
  - Tablo adı
  - Yeni nesnenin toMap metodu
  - where: 'id=?'
  - whereArgs: [nesne.id]

```
static Future<int> update(Kisi kisi) async {
  var db = await open();
  return db.update("kisiler", kisi.toMap(),
      where: "id=?",
      whereArgs: [kisi.id],
      conflictAlgorithm: ConflictAlgorithm.ignore);
}
```

Güncelleme işlemi için insert yerine update metodu kullanıllır. İlk iki parametre insert ile aynı olurken 3. Parametre where ile hangi kaydın güncelleneceği koşulu verilir. Where alanına sql sorgusunda karşılık gelecek eşleme alanı (örneğin id=x'se) verilir. Bu alanda eşitliğe uygulamadan gelecek parametrenin yerini vermek için ? Karakteri kullanılır. whereArgs'da ise bu ? işaretlerine sırası ile karşılık gelecek elemanları içeren bir dizi verilir.

Veritabanı Bağlantısı
Silme

- Update'de olduğu gibi db nesnesinin eldesi
- db.delete()
- Tablo_adi
- where: 'alan= ?'
- whereArgs: [alan]

```
static Future<void> sil(int id) async {
  var db = await open();
  db.delete("kisiler", where: "id=?", whereArgs: [id]);
}
```

Veritabanından bir kayıt silmek için delete kullanılır. Update ile aynı parametreleri almaktadır.

Veritabanı Bağlantısı
Elde etme

- db.query ile map eldesi
- Query içerisine istenilen tablonun yazılması
- Dönen map listesini list.generate ile sınıf listesine çevirme

```
static Future<List<Kisi>> getAll() async {
  var db = await open();
  var kayitlar = await db.query("kisiler");
  return kayitlar.map((e) => Kisi.fromMap(e)).toList();
}
```

Query metodu ile veritabanına sorgu atılabilir. Dönen map değerlerinin nesne listesine döndürülmesi için list.generate kullanılabilir. Ayrıca veritabanından gelen değer Map<String,dynamic> olacağı için bunu nesneye dönüştürecek bir frommap metodu yazılırsa liste.map metodu içerisinde kullanılabilir.