30.10.2025 Ders Notları

Aşağıdaki notlar, **Meta Android Developer Professional Certificate** programındaki derslerden derlenmiştir. Çeşitli konuları kapsayan, kendi anladığım dille, hızlıca tuttuğum notlardır.

Tarih: 30.10.2025
İçerdiği Konular: Veritabanı Temelleri, SQL, Tablolar, İlişkisel Veritabanları, CRUD Operasyonları, Grafik Türleri, SQL Pratik

---

Veritabanına Giriş

- **Veri (Data)** aslında her şeyin gerçekleri ve rakamları. Mesela bir kişi hakkındaki veri; adı, yaşı, email'i, doğum tarihi olabilir. Ya da bir online satışla ilgili sipariş no, açıklama, miktar falan.
- **Veritabanı (Database)**, bu verilerin elektronik ve sistematik bir şekilde saklandığı yer. Manuel dosyalama sisteminin dijital hali. Daha yönetilebilir, verimli ve güvenli olması için kullanılıyor.
- Gerçek hayatta her yerdeler: Bankalar (müşteri hesapları), hastaneler (hasta kayıtları)...
- Veritabanı dediğin temelde düzenli bir **tablo**ya benziyor. Excel sayfası gibi.

Varlıklar (Entities) ve Nitelikler (Attributes)

- Sistematik derken, her verinin tanımlanabilir **nitelikleri** var. Örneğin bir "kişi" entity'si, "ad, soyad, doğum tarihi, email" gibi niteliklere sahip.
- Bu **entity'ler** (varlıklar) aslında tabloların ta kendisi. Satırlar ve sütunlar var.
- Entity'ler fiziksel olabilir (Çalışan, Müşteri, Ürün) ya da kavramsal olabilir (Sipariş, Fatura). Bu kavramı oturtmak lazım.
- İlişkisel veritabanı dünyasında:
  - Entity = **Relation** ya da **Table**
  - Attributes = **Columns** (Sütunlar)
  - Her bir satır = O entity'nin bir **instance'ı** (örneği)

Farklı Veritabanı Türleri

- Sadece ilişkisel veritabanları yokmuş, başka türleri de var:
  - **Nesne-Yönelimli (Object-oriented):** Veriler tablo yerine nesne (object) olarak tutuluyor. Online kitapçı örneğindeki gibi, Yazarlar, Kitaplar birer sınıf (class) olabilir.
  - **Graf (Graph) Veritabanları:** Veriler **düğümler (nodes)** olarak tutuluyor. Müşteri, Sipariş birer düğüm. Aralarındaki ilişkiler de **kenarlar (edges)** ile gösteriliyor. Social network analizi için faydalı olabilir bu, araştırmak gerek.
  - **Döküman (Document) Veritabanları:** Veriler **JSON** objeleri olarak saklanıyor. Koleksiyonlar (collection) içinde dökümanlar var. NoSQL tarafı işte.

- Veritabanları bir organizasyonun kendi sunucusunda (on-premise) olabilir ya da daha popüler ve düşük maliyetli bir seçenek olan **Bulut (Cloud)** üzerinde barındırılabilir.

Veriler Nasıl İlişkilendirilir?

- Veritabanındaki veriler izole bir şekilde duramaz, birbiriyle **ilişki (relationship)** içinde olmalı ki anlamlı bilgiye dönüşebilsin.
- Örnek: Online mağazada Müşteri tablosu ve Sipariş tablosu ayrı. Bir siparişin hangi müşteriye ait olduğunu nasıl anlarız?
- Cevap: **Müşteri ID'si (Customer ID)** ile. Bu ID hem Müşteri tablosunda var, hem de Sipariş tablosunda.

Primary Key (Birincil Anahtar) ve Foreign Key (Yabancı Anahtar)

- **Primary Key (PK):** Bir tablodaki her bir kaydı (record) **benzersiz** şekilde tanımlayan sütun. Mesela Müşteri tablosundaki Customer ID. Aynı isimde iki müşteri olsa bile, ID'leri farklı. Bu alandaki değerler asla null olamaz ve tekrarlanamaz. Çok önemli.
- **Foreign Key (FK):** Bir tablodaki, başka bir tablonun **primary key'ine** referans veren sütun. Sipariş tablosundaki Customer ID buna örnek. Buradaki Customer ID, Müşteri tablosunun PK'sine bağlanır.
- **İlişki bu şekilde kurulur:** Sipariş tablosundaki FK (Customer ID), Müşteri tablosundaki PK (Customer ID) ile eşleşir. Böylece "Sarah'ın siparişleri nedir?" gibi sorulara cevap bulabiliriz.

Veriyi Görselleştirme: Grafikler (Charts)

- Veriyi daha anlamlı hale getirmek ve insanların dikkatini çekmek için grafikler kullanılır. "Bir resim bin kelimeye bedeldir" misali.
- **Çubuk Grafik (Bar Chart):** Kategorik verileri karşılaştırmak için. Çubukların yüksekliği değeri temsil eder. Örneğin, bir kitapçının yıllara göre satışları. En yüksek çubuk en iyi yılı gösterir.
- **Baloncuk Grafik (Bubble Chart):** Baloncukların boyutu değerleri karşılaştırır. Büyük baloncuk büyük değer. Örneğin, ülkelerin nüfusları. Çin ve Hindistan'ın baloncuğu kocaman olur.
- **Çizgi Grafik (Line Chart):** Zaman içindeki **trendleri** (eğilimleri) göstermek için harika. Veri noktaları (markers) çizgilerle birleştirilir. Örneğin, altın fiyatının bir aylık değişimi. Yukarı aşağı hareketler net görülür. Geleceği tahmin etmede kullanılır.
- **Pasta Grafik (Pie Chart):** Bir bütünün parçalarını yüzdesel olarak göstermek için. Toplam %100'dür. Örneğin, bir sınıfta öğrencilerin sevdiği sporlar. Futbol yarısını kaplıyorsa %50'dir. Çok dilim olunca okuması zorlaşıyor ama.

Hangi grafiği seçeceğim; hedef kitleme, vermek istediğim mesaja ve sahip olduğum veri türüne bağlı. Biraz deneme yanılma gerekebilir.

SQL (Structured Query Language) - Veritabanı Dili

- Veritabanıyla konuşmak, onunla etkileşime geçmek için kullandığımız standart dil. "Sequel" diye okunuyor genelde.

- **CRUD Operasyonları** en temel işlemler:
  - **C**reate (Oluştur)
  - **R**ead (Oku)
  - **U**pdate (Güncelle)
  - **D**elete (Sil)
- SQL, DBMS (Database Management System) sayesinde yorumlanır ve çalıştırılır. DBMS, SQL komutlarını veritabanının anlayacağı hale getirir.

SQL Alt Dilleri (Subsets)

SQL işlevlerine göre gruplara ayrılmış. Bunu kafamda netleştirmek için küçük bir tablo yaptım:

Alt Dil	Ne İşe Yarar?	Örnek Komutlar
DDL	Yapıyı tanımlar	`CREATE`, `ALTER`, `DROP`
DML	Veriyi değiştirir	`INSERT`, `UPDATE`, `DELETE`
DQL	Veriyi okur	`SELECT`
DCL	Erişimi yönetir	`GRANT`, `REVOKE`

1. **DDL - Data Definition Language (Veri Tanımlama Dili):** Veritabanı ve tablo gibi yapıları oluşturur, değiştirir, siler.
   - `CREATE`: Database veya tablo oluşturur.
   - `ALTER`: Var olan bir tablonun yapısını değiştirir (sütun ekleme gibi).
   - `DROP`: Database veya tabloyu siler.
2. **DML - Data Manipulation Language (Veri İşleme Dili):** CRUD'un çoğu burada. Veriyle oynarız.
   - `INSERT`: Tabloya yeni veri kaydı ekler.
   - `UPDATE`: Var olan bir veriyi değiştirir.
   - `DELETE`: Veriyi siler.
3. **DQL - Data Query Language (Veri Sorgulama Dili):** Veriyi okumak, sorgulamak için.
   - `SELECT`: Veriyi seçip getirir. En çok kullanacağım komut galiba.
4. **DCL - Data Control Language (Veri Kontrol Dili):** Kullanıcı izinlerini yönetmek için.
   - `GRANT`: İzin verir.
   - `REVOKE`: İzni geri alır.

Temel SQL Syntax Örnekleri (Kolej Veritabanı Örneği)

- **Database Oluşturma (DDL):**

```
CREATE DATABASE college;
```

- **Tablo Oluşturma (DDL):**

```
CREATE TABLE student (...);
```

- **Veri Ekleme (DML - INSERT):**

```
INSERT INTO student (id, first_name, last_name, date_of_birth) VALUES (1,
'John', 'Murphy', '1995-05-15');
```

- **Veri Güncelleme (DML - UPDATE):**

```
UPDATE student SET date_of_birth = '1996-06-16' WHERE id = 2;
```

WHERE olmazsa tüm tabloyu mahvederim, dikkat!

- **Veri Silme (DML - DELETE):**

```
DELETE FROM student WHERE id = 3;
```

Yine `WHERE` şart! Yoksa tüm kayıtlar gider.

- **Veri Okuma/Sorgulama (DQL - SELECT):**

```
SELECT first_name, last_name FROM student WHERE id = 1;
```

Bu bana John Murphy'yi getirir.

SQL Pratik Çalışmam

Öğrendiklerimi pekiştirmek için https://sqliteonline.com üzerinde basit bir `books` tablosu oluşturup sorgular yazdım. İşte kendi pratiğim sonucunda ortaya çıkanlar:

Oluşturduğum `books` Tablosunun İçeriği:

id	name	price	author
4	Berserk	50	Miura
0	Dune	20	Frank
1	Twilight	10	Stephenie
2	Yerdeniz	35	Ursula
3	Şeytan yıldızı	10	jo

Çalıştırdığım Sorgu:

```
SELECT * FROM books ORDER BY name ASC;
```

Bu sorgu, kitapları isme göre alfabetik (A'dan Z'ye) sıralayıp getiriyor. `ORDER BY` kullanımını pratik etmiş oldum. `ASC` (ascending) yazmasam da default olarak zaten öyle sıralıyor, ama yazmak daha iyi alışkanlık. `Berserk`'in en başta çıkması lazım yani. Evet, `Berserk`, `Dune`... diye gidiyor. Oldu!

Tablolar (Tables) & Veri Tipleri Derin Bakış

- Tablo, ilişkisel veritabanının temel nesnesi. Veri burada saklanır.
- **Row (Satır) = Record (Kayıt) = Tuple**
- **Column (Sütun) = Field (Alan) = Attribute (Nitelik)**
- Her sütunun bir **data type (veri tipi)** vardır. Bu, o sütunda ne tür veri saklanacağını belirler. Android'de Room vs. kullanırken Kotlin tipleriyle SQL tiplerini eşleştirmek çok önemli. Not almak için bir tablo yaptım:

Kotlin Tipi	SQL Tipi (SQLite)	Açıklama
`Int`	INTEGER	
`Long`	INTEGER	
`Float`	REAL	
`Double`	REAL	
`String`	TEXT	
`Boolean`	INTEGER (0 veya 1)	0 false, 1 true
`ByteArray`	BLOB	Resim, dosya gibi binary veriler

- **Schema (Şema):** Tablonun ismi, nitelikleri ve veri tiplerinden oluşan yapı. Yani tablonun **blueprint'i**.

Integrity Constraints (Bütünlük Kısıtlamaları)

Tabloların uyması gereken kurallar:

1. **Key Constraint (Anahtar Kısıtı):** Her tablonun bir **Primary Key**'i olmalı ve bu benzersiz (unique) ve boş olmayan (not null) olmalı.
2. **Domain Constraint (Domain Kısıtı):** Her sütuna, tanımlandığı veri tipinde değerler girilmeli. Örneğin, "isim" sütununa telefon numarası yazılamaz. Yukarıdaki veri tipi eşleştirmesi bu yüzden kritik.
3. **Referential Integrity Constraint (Referans Bütünlüğü Kısıtı):** Bir **Foreign Key**, bağlı olduğu tabloda (parent table) mutlaka var olan bir **Primary Key** değerine referans vermeli. Sipariş tablosundaki Customer ID, Müşteri tablosunda gerçekten olan bir ID olmalı.

Bu kısıtlamalar veri tutarlılığı için hayati önemde. Unutmamak lazım.

Son Düşünceler: SQL syntax'ını iyice oturtmam gerekiyor, özellikle WHERE clause olmadan UPDATE ve DELETE yapmamalıyım. Pratik yapmak çok faydalı oldu. Ayrıca farklı veritabanı türlerini (NoSQL vs.

SQL) ve Kotlin-SQL veri tipi eşleşmelerini ne zaman kullanacağımı daha iyi anlamalıyım. Bu notları, özellikle kendi yaptığım pratik çıktıları ve tabloları, ilerideki projelerde sık sık referans alacağım.