31.10.2025 Ders Notları

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

Tarih: 31.10.2025
İçerdiği Konular: Veritabanı Yapısı, Tablolar, Veri Tipleri, SQL Constraints, CREATE TABLE, ALTER TABLE, SELECT, UPDATE, DELETE

Veritabanı Yapısına Genel Bakış

- **Veritabanı yapısı** (database structure) basitçe, verinin veritabanında nasıl düzenlendiği demek.
- İlişkili veriler **tablolarda** (tables/entities) gruplanıyor. Her tablo, Excel'deki gibi **satırlar** (rows/tuples) ve **sütunlardan** (columns/fields) oluşuyor.
- Temel bileşenler:
  - **Tablolar (Entities):** Verinin asıl saklandığı yer.
  - **Nitelikler (Attributes):** Tabloyu/tüzel varlığı tanımlayan özellikler. "Bu tabloda ne var?" sorusunun cevabı.
  - **Alanlar (Fields):** Nitelikleri yakalayan sütunlar.
  - **Kayıt (Record):** Bir tablodaki bir entity'ye ait tüm verilerin bulunduğu satır.
  - **Birincil Anahtar (Primary Key):** Bir entity için benzersiz değer. Her kaydı tek başına tanımlar.

Mantıksal Veritabanı Yapısı & ERD

- Mantıksal yapı, **Varlık İlişki Diyagramı (ERD - Entity Relationship Diagram)** ile gösterilir. Bu, veritabanının fiziksel olarak tablolara nasıl dönüştürüleceğinin görsel bir taslağı gibi.
- Entity'ler (varlıklar) arasında üç tür ilişki kurulabilir:
  1. **Bire-bir (One-to-one)**
  2. **Bire-çok (One-to-many)** -> En yaygın olan bu herhalde.
  3. **Çoka-çok (Many-to-many)**
- Buna ilişkilerin **kardinalitesi** (cardinality) deniyor. ERD'de bu ilişkileri oklarla falan gösteriyorlar.

Fiziksel Veritabanı Yapısı

- Mantıksal yapı fiziksel hale getirilirken, entity'ler tablo olarak oluşturulur.
- İlişkiler ise **Yabancı Anahtar (Foreign Key)** denen bir alanla kurulur. Yabancı anahtar, bir tablodaki, başka bir tablonun (genelde birincil anahtarının) referans aldığı alandır.
- Örneğin, `student` tablosunda `Stud_id` birincil anahtar olsun. `department` tablosunda da `Stud_id` alanı varsa ve bu, `student` tablosundakini referans alıyorsa, bu bir yabancı anahtardır. İki tablo bu şekilde ilişkilendirilmiş olur.

Veri Tipleri (Data Types) - Numeric & String

- Her sütunun bir **veri tipi** olmalı. Bu, o sütuna ne tür veri girilebileceğini belirler. Doğruluk ve güvenilirlik için çok önemli.
- **Numeric Data Types (Sayısal Veri Tipleri):**
  - **Integer (Tamsayı):** Tam sayılar için. `INT`, `TINYINT`, `BIGINT` gibi türleri var. `TINYINT` çok küçük sayılar (max 255), `INT` ise milyarlarca değer alabilir.
  - **Decimal (Ondalıklı):** Kesirli sayılar için. Örneğin fiyat bilgisi (`80.90`). Tam sayı girilirse sonuna `.0` eklenir.
- **String Data Types (Metin Veri Tipleri):**
  - Alfabetik, nümerik karakterler ve özel sembollerin hepsini saklayabilir.
  - **`CHAR(n)`:** Sabit uzunluklu metin. `CHAR(50)` dersen, girdiğin metin 3 karakter de olsa 50 karakterlik yer kaplar. Uzunluğu önceden net belli alanlar için (örneğin ülke kodu 'TR', 'US') ideal.
  - **`VARCHAR(n)`:** Değişken uzunluklu metin. `VARCHAR(50)` dersen, "Ali" 3 karakterlik yer kaplar, "Aleksandra" 10 karakterlik. Ama maksimum 50 karaktere kadar izin verir. Hafıza verimliliği için daha iyi. İsim, adres gibi değişken uzunluklu şeyler için bunu kullanacağım.
  - Diğerleri: `TINYTEXT` (kısa paragraflar), `TEXT` (makaleler), `MEDIUMTEXT` (kitap metinleri!), `LONGTEXT` (devasa metinler). Bunlara şimdilik çok takılmayayım.

Kısıtlamalar (Constraints)

- **Constraints**, veritabanına girebilecek veri türünü sınırlar. Verinin doğru ve güvenilir olmasını sağlar. Kural ihlali olursa (geçersiz veri eklemeye çalışmak gibi) veritabanı işlemi iptal eder.
- **`NOT NULL`:** Bir alanın kesinlikle **boş bırakılamayacağını** belirtir. Örneğin `customer_id` ve `customer_name` mutlaka dolu olmalı. Boş bırakılırsa kayıt oluşturulmaz.
- **`DEFAULT`:** Bir alan için değer girilmezse otomatik olarak atanacak **varsayılan değeri** belirtir.
  - Örnek: Bir futbol takımı oyuncu tablosunda `city` sütunu için varsayılan değeri `Barcelona` yapabilirim. Böylece Barcelona'lı bir oyuncu eklendiğinde şehir bilgisini tekrar tekrar yazmama gerek kalmaz, otomatik `Barcelona` dolar.

`NOT NULL` ve `DEFAULT` SQL Örneği:

```sql
CREATE TABLE player (
    name VARCHAR(50) NOT NULL,
    city VARCHAR(50) DEFAULT 'Barcelona'
);
```

Burada `name` boş geçilemez, `city` boş geçilirse otomatik 'Barcelona' olur.

Tablo Oluşturma (CREATE TABLE)

- Tablo oluşturmak için **DDL** komutu olan `CREATE TABLE` kullanılır.
- **Önemli Noktalar:**
  - Tablo ve sütun isimleri **anlamlı** olmalı.
  - Veri tipleri veritabanı sistemine (MySQL, Oracle, SQL Server) göre değişiklik gösterebiliyor. Kullandığım sisteme dikkat etmeliyim.
  - `VARCHAR` mümkün olduğunca kullanmalıyım, çünkü sadece kullanılan karakter kadar yer kaplar, sabit uzunluklu `CHAR`'a göre daha verimli.
- **Temel Syntax:**

```
CREATE TABLE table_name (
    column1 datatype(length),
    column2 datatype(length),
    ...
);
```

- **Örnek - `customers` Tablosu:**

```
CREATE TABLE customers (
    CustomerId INT,
    FirstName VARCHAR(40),
    LastName VARCHAR(20),
    Email VARCHAR(60)
);
```

  Burada uzunlukları, o alana girebilecek maksimum karakter tahminime göre belirliyorum. `FirstName` için 40, `LastName` için 20 mantıklı görünüyor.

Tabloyu Değiştirme (ALTER TABLE)

- Hiçbir tablo sabit değil, ihtiyaçlara göre değiştirilir. Bunun için `ALTER TABLE` komutu kullanılır.
- **Sütun Eklemek (ADD):**

```
ALTER TABLE students
ADD (age INT, country VARCHAR(50));
```

  `students` tablosuna `age` (tamsayı) ve `country` (max 50 karakter string) sütunları ekler.
- **Sütun Silmek (DROP COLUMN):**

```
ALTER TABLE students
DROP COLUMN nationality;
```

  `nationality` sütununu tablodan tamamen siler. Dikkatli olmalıyım, veriler gider!
- **Sütun Yapısını Değiştirmek (MODIFY):**

```sql
ALTER TABLE students
MODIFY country VARCHAR(100);
```

`country` sütununun maksimum karakter uzunluğunu 50'den 100'e çıkarır.

Veri Sorgulama (SELECT) & Veri Manipülasyonu (UPDATE, DELETE)

- **Veri Sorgulama (SELECT - DQL):**
  - Temel kullanım: `SELECT column_name FROM table_name;`
  - Tüm sütunlar için: `SELECT * FROM table_name;` (Asteriks `*` kısayol)
  - Birden fazla sütun: `SELECT name, age FROM players;`
  - Basit bir `SELECT` çok hızlı ve hafif ama tablo büyüdükçe `WHERE` ile filtreleme yapmak şart olacak.
- **Veri Güncelleme (UPDATE - DML):**
  - **TEK KAYIT:** `UPDATE student_table SET home_address = 'Yeni Adres', contact_number = '555-1234' WHERE id = 3;`
  - **ÇOKLU KAYIT:** `UPDATE student_table SET college_address = 'Harper Building' WHERE department = 'Engineering';` -> Mühendislik bölümündeki *tüm* öğrencilerin adresini günceller.
  - **`WHERE` ŞARTI HAYATİ ÖNEMDE!** `WHERE` yazmazsam tablodaki **TÜM KAYITLAR** güncellenir ve bu bir felaket olur!
- **Veri Silme (DELETE - DML):**
  - **TEK KAYIT:** `DELETE FROM student_table WHERE last_name = 'Miller';`
  - **ÇOKLU KAYIT:** `DELETE FROM student_table WHERE department = 'Engineering';` -> Mühendislik bölümündeki tüm öğrencileri siler.
  - **TÜM KAYITLAR (TABLOYU BOŞALTMA):** `DELETE FROM student_table;` -> `WHERE` olmadığı için tüm kayıtlar gider, tablo bomboş kalır. ÇOK TEHLİKELİ!
  - **`WHERE` Yine Çok Önemli!** Silme işlemlerinde iki kere düşünüp bir kere yazmalıyım.

Modül Sonu Özeti

- **Veritabanları & SQL** modülünün sonuna geldik.
- **Veritabanı Temelleri:** Verinin nasıl düzenlendiğini, ilişkilerin neden önemli olduğunu öğrendim.
- **SQL:** Veritabanıyla konuşmanın yolu. CRUD operasyonları (Create, Read, Update, Delete) ve DDL, DML, DQL gibi alt dilleri gördüm.
- **Tablolar:** Tabloların yapısı, veri tipleri, kısıtlamalar (constraints) üzerine çalıştım. `CREATE TABLE` ve `ALTER TABLE` komutlarını kullanmayı öğrendim.
- **Veri İşleme:** `SELECT`, `UPDATE`, `DELETE` komutlarını kullanarak veri sorgulamayı, güncellemeyi ve silmeyi pratik ettim. En kritik nokta `WHERE` cümleciğini doğru kullanmak!