# İLİŞKİSEL VERİ TABANLARI (SQL - RELATIONAL - RDBMS)

İlişkisel veri tabanları, verileri tablo biçiminde düzenleyen ve bu tablolar arasında mantıksal ilişkiler kurarak bilgileri yöneten sistemlerdir. SQL (Structured Query Language), bu sistemlerde veri sorgulama ve yönetme dili olarak kullanılır. RDBMS (Relational Database Management System), bu modelin yazılımsal karşılığıdır.

## 1. İlişkisel Veri Tabanlarının Temel Özellikleri

- **Veri tablolar içinde saklanır.** Her tablo, belirli bir konuya ait satır ve sütunlardan oluşur.
- **Satırlar (kayıtlar),** her bir varlığa ait bilgileri içerir.
- **Sütunlar (alanlar),** verinin türünü ve niteliğini belirler.
- **Her satır genellikle benzersiz bir anahtarla (primary key) tanımlanır.**
- **Tablolar arasında ilişki kurmak mümkündür.** Bu ilişkiler sayesinde birden fazla tabloyu birbirine bağlayarak anlamlı sonuçlar çıkarılabilir.

## 2. Tablolar ve İlişkiler

Veri tabanında her tablo, bir nesne ya da varlık kümesini temsil eder. Örneğin bir personel veri tabanında aşağıdaki tablolar yer alabilir:

| Tablo Adı | Tanımı |
|---|---|
| EMPLOYEES | Çalışanların bilgilerini tutar |
| DEPARTMENTS | Departman bilgilerini içerir |
| JOB_HISTORY | Çalışanların önceki görevlerini kaydeder |
| LOCATIONS | Şirket lokasyonları |
| COUNTRIES | Ülkeler bilgisi |
| REGIONS | Kıtalar ya da bölgesel bilgiler |

### Örnek:

- `EMPLOYEES` tablosunda her çalışan için `employee_id` adlı **birincil anahtar (primary key)** bulunur.
- `DEPARTMENTS` tablosunda ise her departman için `department_id` adlı anahtar vardır.
- `EMPLOYEES` tablosundaki `department_id` sütunu, `DEPARTMENTS` tablosuyla **yabancı anahtar (foreign key)** ilişkisi kurar.
- Böylece bir çalışanın hangi departmanda görev yaptığını tablo ilişkisiyle öğrenebiliriz.

## 3. Anahtarlar (Keys)

İlişkisel veri tabanlarında **anahtarlar (keys)**, veriler arasındaki bağıntıları tanımlamak için kullanılır:

- **Primary Key (Birincil Anahtar):** Her satırı eşsiz şekilde tanımlar. Boş bırakılamaz, tekrar edemez.

**Örnek:** `employee_id`, `department_id`

- **Foreign Key (Yabancı Anahtar):** Bir tablodaki sütunun başka bir tablodaki birincil anahtara işaret etmesini sağlar.

**Örnek:** `EMPLOYEES.department_id → DEPARTMENTS.department_id`

## 4. İlişkilerin Türleri

Tablolar arasında üç temel ilişki türü bulunur:

| İlişki Türü | Açıklama | Örnek |
|---|---|---|
| **1:1** | Bir kaydın yalnızca bir karşılığı vardır | `Kişi – TC Kimlik Bilgisi` |
| **1:N (Bire Çok)** | Bir kaydın birçok karşılığı olabilir | `DEPARTMENTS – EMPLOYEES` (bir departmanda birçok çalışan olabilir) |
| **N:M (Çoktan Çoka)** | Ara tabloyla kurulan ilişki | `Öğrenciler – Dersler` (bir öğrenci birçok dersi alabilir, bir ders birçok öğrenciye ait olabilir) |

## 5. Normalizasyon

İlişkisel veri tabanlarında **normalizasyon**, verilerin tekrarını azaltmak ve veri tutarlılığını sağlamak için kullanılan bir tekniktir. Veriler küçük, ilişkilendirilmiş tablolara bölünür.

**Örnek:**

- Çalışan bilgilerini EMPLOYEES tablosuna, departman bilgilerini DEPARTMENTS tablosuna koyarak veri tekrarını önleriz.

## 6. Örnek SQL İfadeleri

**Tablo oluşturma:**

```
CREATE TABLE EMPLOYEES (
  employee_id INT PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  department_id INT,
  FOREIGN KEY (department_id) REFERENCES DEPARTMENTS(department_id)
);
```

**Tabloları ilişkilendirme:**

```
SELECT E.first_name, E.last_name, D.department_name
FROM EMPLOYEES E
JOIN DEPARTMENTS D ON E.department_id = D.department_id;
```

## SONUÇ

İlişkisel veri tabanları, yapısal ve güvenilir veri yönetimi sunan güçlü sistemlerdir. Tablolar, sütunlar ve anahtarlar sayesinde veriler anlamlı şekilde bağlanabilir ve ilişkiler kurulabilir. SQL dili ile bu yapılar üzerinde sorgular ve işlemler gerçekleştirmek mümkündür. Bu sistem, veri bütünlüğünü sağlamak, tekrarları önlemek ve karmaşık veri kümelerini yönetilebilir hâle getirmek için temel bir modeldir.

# Veri Tabanı Kısıtlamaları (Constraints)

Veri tabanı sistemlerinde *kısıtlamalar (constraints)*, veri bütünlüğünü sağlamak amacıyla kullanılan kurallardır. Bu kurallar, tabloya girilecek verilerin doğruluğunu ve tutarlılığını garanti altına alır. Kısıtlamalar sayesinde hatalı veri girişi engellenir, veri tabanı daha güvenilir hâle gelir. Kısıtlamalar, tablo oluşturulurken ya da sonradan ALTER komutlarıyla eklenebilir.

## 1. PRIMARY KEY (Birincil Anahtar)

- Her tablonun benzersiz şekilde tanımlanmasını sağlayan anahtar sütundur.
- Aynı değeri iki kere alamaz (tekrarsızdır).
- NULL değeri içeremez.
- Her tabloda yalnızca **bir adet** birincil anahtar tanımlanabilir.

**Örnek:**

```
CREATE TABLE BOLUMLER (
  bolum_id INT PRIMARY KEY,
  bolum_adi VARCHAR(50)
);
```

Burada `bolum_id` her bölüm için benzersiz bir kimlik sağlar.

## 2. FOREIGN KEY (Yabancı Anahtar)

- Bir tablodaki sütunun, başka bir tablonun birincil anahtarına referans vermesini sağlar.
- İki tablo arasında ilişkisel bağlantı kurar.
- Veri bütünlüğünü korur.

**Örnek:**

```
CREATE TABLE OGRENCILER (
  ogrenci_id INT PRIMARY KEY,
```

```
  ad VARCHAR(50),
  bolum_id INT,
  FOREIGN KEY (bolum_id) REFERENCES BOLUMLER(bolum_id)
);
```

`OGRENCILER` tablosundaki `bolum_id`, sadece `BOLUMLER` tablosunda tanımlı olan `bolum_id` değerlerini alabilir.

### 3. NOT NULL

- Bir sütuna **boş değer (NULL)** girilmesini engeller.
- Veri girişinin zorunlu olduğu alanlarda kullanılır.

**Örnek:**

```
CREATE TABLE KULLANICILAR (
  kullanici_id INT PRIMARY KEY,
  kullanici_adi VARCHAR(50) NOT NULL
);
```

`kullanici_adi` alanı boş bırakılamaz.

## 4. UNIQUE

- Sütundaki her değerin **benzersiz** olmasını sağlar.
- NULL değer alabilir (bir kez).
- Aynı tabloda birden fazla UNIQUE sütun olabilir.

**Örnek:**

```
CREATE TABLE EMAIL_KAYIT (
  kayit_id INT PRIMARY KEY,
  email_adresi VARCHAR(100) UNIQUE
);
```

Her e-posta adresi tabloda yalnızca bir kez yer alabilir.

## 5. CHECK

- Belirli bir koşula göre veri girişini sınırlar.
- Sayısal aralık, uzunluk veya özel şartlar belirlenebilir.

**Örnek:**

```
CREATE TABLE URUNLER (
  urun_id INT PRIMARY KEY,
  fiyat DECIMAL(10,2) CHECK (fiyat > 0)
);
```

Ürün fiyatı sıfırdan büyük olmalıdır, aksi hâlde kayıt yapılmaz.

## 6. DEFAULT

- Bir sütuna **varsayılan bir değer** atar.
- Veri girilmediğinde otomatik olarak bu değer kullanılır.

**Örnek:**

```
CREATE TABLE SIPARISLER (
  siparis_id INT PRIMARY KEY,
  durum VARCHAR(20) DEFAULT 'Beklemede'
);
```

Kullanıcı sipariş durumu belirtmezse otomatik olarak "Beklemede" olarak kaydedilir.

## 7. AUTO_INCREMENT / IDENTITY

- Yeni veri girildiğinde **otomatik artan** benzersiz sayılar oluşturur.
- Genellikle birincil anahtar sütunlarında kullanılır.

**MySQL:**

```
CREATE TABLE OGRETMENLER (
  ogretmen_id INT AUTO_INCREMENT PRIMARY KEY,
  ad_soyad VARCHAR(50)
);
```

**SQL Server:**

```
CREATE TABLE OGRETMENLER (
  ogretmen_id INT IDENTITY(1,1) PRIMARY KEY,
  ad_soyad VARCHAR(50)
);
```

Her yeni öğretmen kaydı için `ogretmen_id` otomatik olarak artar.

## Sonuç

Veri tabanı kısıtlamaları; güvenilir, hatasız ve düzenli veri yönetimi için temel unsurlardır. Doğru şekilde kullanıldığında, hem veri bütünlüğü sağlanır hem de tablolar arası ilişkiler sorunsuz biçimde yönetilir.

**Bu çalışma BTK Akademi eğitim içerikleri referans alınarak, Arda Karadağ tarafından hazırlanmıştır.**