T-SQL (MS SQL SERVER)

1. GİRİŞ

1.1 Veritabanı Nedir?

Veritabanı, verilerin belirli bir düzen içerisinde saklandığı, yönetildiği ve gerektiğinde hızlı bir şekilde erişilebildiği sistemlerdir. Günümüzde büyük miktarda verinin güvenli ve tutarlı bir biçimde saklanabilmesi için veritabanı sistemleri vazgeçilmez bir yapı haline gelmiştir.

Veritabanları, yalnızca veriyi depolamakla kalmaz; aynı zamanda bu veriler üzerinde işlem yapılmasına, güncellenmesine, silinmesine ve analiz edilmesine olanak tanır. Bu yönüyle veritabanları, modern yazılım sistemlerinin temel bileşenlerinden biridir.

İlişkisel veritabanı yönetim sistemlerinde (RDBMS), veriler tablolar halinde tutulur. Her tablo satır (kayıt) ve sütunlardan (alanlardan) oluşur. Bu yapı sayesinde veriler hem düzenli hem de anlamlı bir biçimde organize edilir.

Veritabanlarının temel amaçları aşağıdaki şekilde özetlenebilir:

- Verilerin kalıcı olarak saklanmasını sağlamak
- Verileri düzenli ve yapılandırılmış bir biçimde tutmak
- Veriye hızlı ve etkin erişim imkanı sunmak
- Veri güvenliğini sağlamak
- Veri bütünlüğünü korumak

Örneğin bir okul yönetim sistemi düşünüldüğünde; öğrenciler, öğretmenler, dersler ve notlar gibi veriler ayrı tablolar halinde tutulur. Bu tablolar arasında kurulan ilişkiler sayesinde karmaşık veri yapıları anlamlı ve yönetilebilir hale gelir.

Sonuç olarak veritabanı, büyük ve karmaşık veri kümelerinin sistematik bir şekilde yönetilmesini sağlayan temel bir teknolojidir.

1.2 İlişkisel Veritabanı Mantığı

İlişkisel veritabanı modeli, verilerin tablolar aracılığıyla saklandığı ve bu tablolar arasında ilişkiler kurularak veri bütünlüğünün sağlandığı bir yaklaşımdır. Bu model, matematiksel olarak “ilişkiler teorisine” dayanır ve günümüzde en yaygın kullanılan veritabanı modelidir.

İlişkisel veritabanlarında her tablo belirli bir varlığı temsil eder. Örneğin “Ogrenciler” tablosu öğrenci bilgilerini, “Dersler” tablosu ders bilgilerini içerir. Bu tablolar arasında kurulan ilişkiler sayesinde veri tekrarının önüne geçilir ve veri tutarlılığı sağlanır.

İlişkisel modelin temel özellikleri şunlardır:

- Veriler tablolar halinde saklanır
- Her tablo birincil anahtar (Primary Key) içerir
- Tablolar arasında yabancı anahtar (Foreign Key) ile ilişki kurulur
- Veri tekrarını azaltmak için normalizasyon uygulanır

Örneğin bir öğrencinin aldığı dersleri tutmak için, öğrenci bilgileri ile ders bilgileri ayrı tablolarda tutulur. Bu iki tablo arasında kurulan ilişki sayesinde, bir öğrencinin hangi dersleri aldığı kolayca belirlenebilir.

Bu yapı sayesinde veritabanı: - Daha az yer kaplar - Daha düzenli olur - Daha güvenilir hale gelir

İlişkisel veritabanı mantığı, T-SQL ile yapılacak tüm işlemlerin temelini oluşturur.

1.3 SQL ve T-SQL Arasındaki Fark

SQL (Structured Query Language), ilişkisel veritabanları ile iletişim kurmak için kullanılan standart bir sorgu dilidir. Veritabanı oluşturma, veri ekleme, silme, güncelleme ve sorgulama gibi işlemler SQL komutları ile gerçekleştirilir.

T-SQL (Transact-SQL) ise Microsoft SQL Server tarafından geliştirilmiş, SQL dilinin genişletilmiş bir sürümüdür. Standart SQL'e ek olarak programlama yetenekleri sunar.

T-SQL'in SQL'den farkları şu şekilde özetlenebilir:

- Değişken tanımlama ve kullanma imkanı sağlar
- IF-ELSE gibi kontrol yapıları içerir
- WHILE gibi döngü yapıları desteklenir
- TRY-CATCH ile hata yönetimi yapılabilir
- Stored Procedure ve Trigger gibi gelişmiş yapılar içerir

Bu özellikler sayesinde T-SQL, yalnızca sorgu dili olmanın ötesine geçerek bir programlama dili gibi kullanılabilir.

1.4 SQL Server Mimarisi (Genel Bakış)

Microsoft SQL Server, istemci (client) ve sunucu (server) mantığı ile çalışan bir veritabanı yönetim sistemidir. Kullanıcılar tarafından gönderilen sorgular, SQL Server tarafından işlenir ve sonuçlar geri döndürülür.

SQL Server mimarisi temel olarak iki ana bileşenden oluşur:

1. Query Processor (Sorgu İşleyici)

Query Processor, kullanıcıdan gelen T-SQL sorgularını analiz eder ve en verimli şekilde nasıl çalıştırılacağını belirler. Bu süreçte sorgu optimize edilir ve bir yürütme planı (execution plan) oluşturulur.

2. Storage Engine (Depolama Motoru)

Storage Engine, verilerin fiziksel olarak disk üzerinde saklanmasından ve gerektiğinde okunmasından sorumludur. Query Processor tarafından oluşturulan plan doğrultusunda veriye erişim sağlar.

Bu iki yapı birlikte çalışarak, sorguların hızlı ve verimli bir şekilde işlenmesini sağlar.

---

2. TEMEL KAVRAMLAR

2.1 Tablo (Table)

Tablo, veritabanında verilerin saklandığı temel yapıdır. Her tablo belirli bir varlığı temsil eder ve satır (row) ile sütunlardan (column) oluşur. Örneğin bir “Ogrenciler” tablosu, öğrencilere ait bilgileri içerir.

Tabloların doğru tasarlanması, veritabanının performansı ve veri bütünlüğü açısından kritik öneme sahiptir.

2.2 Satır (Row) ve Sütun (Column)

- **Satır (Row):** Tablodaki her bir kayıtı temsil eder. Örneğin bir öğrenciye ait tüm bilgiler bir satırda tutulur.
- **Sütun (Column):** Verinin türünü ve özelliğini belirler. Örneğin “Ad”, “Soyad”, “Yas” gibi alanlar sütundur.

Her sütun belirli bir veri tipine sahiptir ve bu veri tipleri, o sütuna hangi türde veri girileceğini belirler.

2.3 Primary Key (Birincil Anahtar)

Primary Key, bir tablodaki her kaydı benzersiz olarak tanımlayan alandır. Aynı tabloda iki satırın Primary Key değeri aynı olamaz ve boş (NULL) bırakılamaz.

Primary Key özellikleri: - Benzersizdir (Unique) - NULL değer alamaz - Tablo başına bir tane bulunur

Örnek kullanım:

```
CREATE TABLE Ogrenciler (
    OgrenciID INT PRIMARY KEY,
    Ad NVARCHAR(50)
);
```

2.4 Foreign Key (Yabancı Anahtar)

Foreign Key, bir tablodaki alanın başka bir tablodaki Primary Key ile ilişki kurmasını sağlar. Bu yapı sayesinde tablolar arasında bağlantı kurulur.

Örnek:

```
CREATE TABLE Dersler (
    DersID INT PRIMARY KEY,
    DersAdi NVARCHAR(50)
);

CREATE TABLE OgrenciDers (
    OgrenciID INT,
    DersID INT,
    FOREIGN KEY (DersID) REFERENCES Dersler(DersID)
);
```

Bu yapı, veri bütünlüğünü korur ve yanlış veri girişini engeller.

2.5 Constraint (Kısıtlar)

Constraint yapıları, veritabanına girilen verilerin belirli kurallara uygun olmasını sağlar. Bu sayede veri doğruluğu ve tutarlılığı korunur.

Başlıca constraint türleri:

- **NOT NULL:** Alan boş bırakılamaz
- **UNIQUE:** Aynı değer tekrar edemez
- **CHECK:** Belirli bir koşulu sağlamalıdır
- **DEFAULT:** Varsayılan değer atar

Örnek:

```
CREATE TABLE Ogrenciler (
    Yas INT CHECK (Yas > 0),
    Ad NVARCHAR(50) NOT NULL
);
```

2.6 Index (İndeks – Temel Mantık)

Index, veritabanında verilere daha hızlı erişim sağlamak için kullanılan yapılardır. Kitapların sonundaki indeks mantığına benzer.

Index kullanımı sayesinde: - Sorgular hızlanır - Büyük veri setlerinde performans artar

Ancak gereksiz index kullanımı: - Yazma işlemlerini yavaşlatır - Ek depolama alanı kullanır

Bu nedenle index kullanımı dikkatli planlanmalıdır.

Hazırsan sıradaki bölüm olan **3. Veri Tipleri** kısmına geçebiliriz.

---

3. VERİ TİPLERİ (DATA TYPES)

Veri tipleri, bir sütunda tutulacak verinin türünü belirler. Doğru veri tipi seçimi; performans, depolama verimliliği ve veri doğruluğu açısından büyük önem taşır.

SQL Server'da veri tipleri genel olarak sayısal, metinsel, tarih ve diğer veri tipleri olarak sınıflandırılır.

3.1 Sayısal Veri Tipleri

Sayısal veri tipleri, matematiksel işlemler yapılacak veriler için kullanılır.

- **INT:** Tam sayı değerleri için kullanılır (-2 milyar ile +2 milyar arası)
- **BIGINT:** Çok büyük tam sayılar için kullanılır
- **SMALLINT:** Daha küçük tam sayılar için kullanılır
- **TINYINT:** 0 ile 255 arasındaki değerler için kullanılır

Kesirli sayılar için:

- **DECIMAL(p,s):** Hassas sayılar için kullanılır (p: toplam basamak, s: ondalık kısmı)
- **NUMERIC:** DECIMAL ile aynıdır
- **FLOAT:** Yaklaşık değerler için kullanılır (bilimsel hesaplamalar)

Örnek:

```
Maas DECIMAL(10,2)
```

3.2 Metinsel Veri Tipleri

Metin tabanlı veriler için kullanılır.

- **CHAR(n):** Sabit uzunluklu metin
- **VARCHAR(n):** Değişken uzunluklu metin
- **NCHAR / NVARCHAR:** Unicode destekli metin (Türkçe karakterler için önerilir)

Örnek:

```
Ad NVARCHAR(50)
```

NOT: NVARCHAR kullanımı, çok dilli uygulamalarda veri kaybını önler.

3.3 Tarih ve Zaman Veri Tipleri

Tarih ve saat bilgileri için kullanılır.

- **DATE:** Sadece tarih
- **TIME:** Sadece saat
- **DATETIME:** Tarih ve saat
- **DATETIME2:** Daha hassas tarih ve saat
- **SMALLDATETIME:** Daha az hassas ama daha az yer kaplar

Örnek:

```
KayitTarihi DATETIME2
```

3.4 Diğer Veri Tipleri

- **BIT:** 0 veya 1 (true/false)
- **UNIQUEIDENTIFIER:** Benzersiz ID (GUID)
- **XML:** XML veri saklamak için
- **VARBINARY:** Binary veri (resim, dosya vb.)

3.5 Veri Tipi Seçiminde Dikkat Edilmesi Gerekenler

Doğru veri tipi seçimi şu açılardan kritiktir:

1. Performans

Gereğinden büyük veri tipi kullanmak performansı düşürür.

2. Depolama

Örneğin: - INT yerine BIGINT kullanmak gereksiz yer kaplar

3. Veri Doğruluğu

Yanlış veri tipi: - Veri kaybına - Hatalı hesaplamalara neden olabilir

4. Unicode Kullanımı

Türkçe karakterler için:

```
NVARCHAR tercih edilmelidir
```

Sonuç olarak veri tipleri, veritabanı tasarımının temel taşlarından biridir ve doğru seçilmesi sistemin genel başarısını doğrudan etkiler.

4. TABLO OLUŞTURMA VE YAPI TASARIMI (DDL)

DDL (Data Definition Language), veritabanı nesnelerinin oluşturulması, değiştirilmesi ve silinmesi için kullanılan komutları içerir. Bu bölümde tablo oluşturma ve yapı yönetimi detaylı olarak ele alınacaktır.

4.1 CREATE TABLE (Tablo Oluşturma)

CREATE TABLE komutu, veritabanında yeni bir tablo oluşturmak için kullanılır. Tablo oluşturulurken sütun isimleri, veri tipleri ve kısıtlamalar (constraint) belirlenir.

Örnek:

```
CREATE TABLE Ogrenciler (
    OgrenciID INT PRIMARY KEY,
    Ad NVARCHAR(50) NOT NULL,
    Soyad NVARCHAR(50) NOT NULL,
    Yas INT,
    KayitTarihi DATETIME2 DEFAULT GETDATE()
);
```

Bu örnekte: - OgrenciID benzersizdir - Ad ve Soyad boş bırakılamaz - KayitTarihi otomatik atanır

---

4.2 Constraint Türleri (Detaylı)

Constraint yapıları veri bütünlüğünü sağlamak için kullanılır.

PRIMARY KEY

Bir kaydı benzersiz tanımlar.

FOREIGN KEY

Tablolar arasında ilişki kurar.

UNIQUE

Bir sütundaki değerlerin tekrar etmesini engeller.

NOT NULL

Boş değer girilmesini engeller.

CHECK

Belirli bir koşul koyar.

```
Yas INT CHECK (Yas >= 0)
```

DEFAULT

Varsayılan değer atar.

```
Durum BIT DEFAULT 1
```

4.3 ALTER TABLE (Tablo Değiştirme)

Var olan tablolar üzerinde değişiklik yapmak için kullanılır.

Sütun ekleme

```
ALTER TABLE Ogrenciler
ADD Email NVARCHAR(100);
```

Sütun silme

```
ALTER TABLE Ogrenciler
DROP COLUMN Email;
```

Veri tipi değiştirme

```
ALTER TABLE Ogrenciler
ALTER COLUMN Ad NVARCHAR(100);
```

4.4 DROP ve TRUNCATE Arasındaki Fark

DROP TABLE

- Tabloyu tamamen siler
- Yapı + veri gider

```
DROP TABLE Ogrenciler;
```

TRUNCATE TABLE

- Sadece verileri siler
- Tablo yapısı korunur
- Daha hızlıdır

```
TRUNCATE TABLE Ogrenciler;
```

Farklar: - TRUNCATE daha performanslıdır - DROP geri dönüşü olmayan işlemdir

---

4.5 Tablo Tasarımında Dikkat Edilmesi Gerekenler

İyi bir tablo tasarımı, veritabanı performansını doğrudan etkiler.

Dikkat edilmesi gerekenler:

- Doğru veri tipi seçimi yapılmalıdır
- Gereksiz sütunlardan kaçınılmalıdır
- Primary Key mutlaka tanımlanmalıdır
- İlişkiler doğru kurulmalıdır
- Veri tekrarından kaçınılmalıdır

Kötü tasarım: - Performans düşüşü - Veri tutarsızlığı - Bakım zorluğu

---

Bu bölüm, veritabanı tasarımının temelini oluşturur ve sonraki konuların anlaşılması için kritik öneme sahiptir.

5. İLİŞKİSEL VERİTABANI TASARIMI

İlişkisel veritabanı tasarımı, verilerin tablolar arasında anlamlı ilişkiler kurularak organize edilmesini ifade eder. Bu yapı, veri tekrarını azaltır ve veri bütünlüğünü sağlar.

İlişkisel modelin temel amacı, verilerin mantıksal olarak bölünmesi ve bu bölümler arasında doğru bağlantıların kurulmasıdır.

5.1 İlişki Türleri

Veritabanlarında tablolar arasında üç temel ilişki türü vardır:

1. One-to-One (Bire Bir İlişki)

Bir tablodaki bir kaydın, diğer tabloda yalnızca bir karşılığı vardır.

Örnek: - Kişi ↔ Kimlik Bilgileri

Kullanım amacı: Hassas veya ayrı tutulması gereken verileri bölmek.

---

2. One-to-Many (Bire Çok İlişki)

Bir tablodaki bir kaydın, diğer tabloda birden fazla karşılığı olabilir.

Örnek: - Öğretmen → Dersler - Bir öğretmen birden fazla ders verebilir.

Bu en yaygın kullanılan ilişki türüdür.

---

3. Many-to-Many (Çoka Çok İlişki)

Bir tablodaki bir kayıt, diğer tablodaki birden fazla kayıtla ilişkili olabilir.

Örnek: - Öğrenci ↔ Ders - Bir öğrenci birden fazla ders alabilir, bir ders de birden fazla öğrenciye ait olabilir.

Bu ilişki genellikle ara tablo (junction table) ile çözülür.

5.2 Normalizasyon

Normalizasyon, veritabanındaki veri tekrarını azaltmak ve veri bütünlüğünü sağlamak için kullanılan bir tasarım tekniğidir.

1NF (Birinci Normal Form)

- Her hücre tek değer içermelidir
- Tekrarlayan sütunlar olmamalıdır

Yanlış: | Ogrenci | Dersler | |———|———| | Ali | Matematik, Fizik |

Doğru: | Ogrenci | Ders | |———|——| | Ali | Matematik | | Ali | Fizik |

2NF (İkinci Normal Form)

- 1NF sağlanmalıdır
- Tablodaki her alan, tam anahtara bağlı olmalıdır

Amaç: Kısmi bağımlılığı ortadan kaldırmak

3NF (Üçüncü Normal Form)

- 2NF sağlanmalıdır
- Alanlar arasında geçişli bağımlılık olmamalıdır

Örnek: - Ogrenci → Bolum → BolumTelefon Bu yapı bölünmelidir.

5.3 Foreign Key ve Veri Bütünlüğü

Foreign Key, tablolar arasında ilişki kurarak veri bütünlüğünü sağlar.

Örnek:

```
CREATE TABLE Ogrenciler (
    OgrenciID INT PRIMARY KEY,
    Ad NVARCHAR(50)
);

CREATE TABLE DersKayit (
    KayitID INT PRIMARY KEY,
    OgrenciID INT,
    FOREIGN KEY (OgrenciID) REFERENCES Ogrenciler(OgrenciID)
);
```

Bu yapı sayesinde: - Olmayan öğrenciye ders atanamaz - Veri tutarlılığı korunur

5.4 Cascade İşlemleri

Cascade işlemleri, ilişkili tablolar arasında otomatik güncelleme ve silme işlemlerini sağlar.

ON DELETE CASCADE

Ana tablodaki kayıt silinirse, bağlı kayıtlar da silinir.

ON UPDATE CASCADE

Ana tablodaki anahtar güncellenirse, bağlı tablolar da güncellenir.

Örnek:

```
FOREIGN KEY (OgrenciID)
REFERENCES Ogrenciler(OgrenciID)
ON DELETE CASCADE
ON UPDATE CASCADE
```

5.5 Tasarım Mantığı (Önemli Prensipler)

- Veri tekrarından kaçınılmalıdır
- Tablolar mantıksal parçalara ayrılmalıdır
- Her tablo tek bir konuyu temsil etmelidir
- İlişkiler Foreign Key ile kurulmalıdır

Bu yapı, profesyonel veritabanı tasarımının temelini oluşturur ve büyük ölçekli sistemlerde veri tutarlılığını garanti eder.

6. DML KOMUTLARI (DATA MANIPULATION LANGUAGE)

DML (Data Manipulation Language), veritabanı içerisindeki verilerin eklenmesi, güncellenmesi, silinmesi ve sorgulanması için kullanılan komutları kapsar. Bu komutlar doğrudan tablo içindeki veriler üzerinde işlem yapar.

6.1 INSERT (Veri Ekleme)

INSERT komutu, tabloya yeni kayıt eklemek için kullanılır.

Tek kayıt ekleme:

```
INSERT INTO Ogrenciler (OgrenciID, Ad, Soyad, Yas)
VALUES (1, 'Ahmet', 'Yılmaz', 15);
```

Çoklu kayıt ekleme:

```
INSERT INTO Ogrenciler (OgrenciID, Ad, Soyad, Yas)
VALUES
(2, 'Ayşe', 'Kara', 16),
(3, 'Mehmet', 'Demir', 14);
```

INSERT işlemlerinde sütunların doğru sırada ve uyumlu veri tipinde olması gerekir.

6.2 SELECT (Veri Sorgulama)

SELECT komutu, veritabanından veri çekmek için kullanılır ve en çok kullanılan SQL komutudur.

Tüm verileri çekme:

```
SELECT * FROM Ogrenciler;
```

Belirli sütunları çekme:

```
SELECT Ad, Soyad FROM Ogrenciler;
```

WHERE ile koşullu sorgu:

```
SELECT * FROM Ogrenciler
WHERE Yas > 15;
```

ORDER BY (Sıralama):

```
SELECT * FROM Ogrenciler
ORDER BY Yas DESC;
```

DISTINCT (Tekrarsız veri):

```
SELECT DISTINCT Yas FROM Ogrenciler;
```

TOP (İlk N kayıt):

```
SELECT TOP 5 * FROM Ogrenciler;
```

6.3 UPDATE (Veri Güncelleme)

UPDATE komutu, mevcut kayıtları güncellemek için kullanılır.

Örnek:

```
UPDATE Ogrenciler
SET Yas = 17
WHERE OgrenciID = 1;
```

DİKKAT: WHERE kullanılmazsa tüm tablo güncellenir.

6.4 DELETE (Veri Silme)

DELETE komutu, tablodan kayıt silmek için kullanılır.

Örnek:

```
DELETE FROM Ogrenciler
WHERE OgrenciID = 1;
```

DİKKAT: WHERE kullanılmazsa tüm veriler silinir.

6.5 WHERE KOŞULLARI

WHERE ifadesi filtreleme işlemi için kullanılır.

Karşılaştırma operatörleri:

- = (eşittir)
- != (eşit değil)
- (büyüktür)
- < (küçüktür)
- = (büyük eşit)
- <= (küçük eşit)

Mantıksal operatörler:

```
AND
OR
NOT
```

Örnek:

```
SELECT * FROM Ogrenciler
WHERE Yas > 14 AND Yas < 18;
```

6.6 GROUP BY ve HAVING

GROUP BY

Verileri gruplamak için kullanılır.

```
SELECT Yas, COUNT(*) AS OgrenciSayisi
FROM Ogrenciler
GROUP BY Yas;
```

HAVING

Gruplar üzerinde koşul uygulamak için kullanılır.

```sql
SELECT Yas, COUNT(*) AS OgrenciSayisi
FROM Ogrenciler
GROUP BY Yas
HAVING COUNT(*) > 1;
```