



Veri Tabanı Yönetim Sistemleri

Bölüm 4 – SQL – Veri İşleme (Data Manipulation)

Dr. Öğr. Üyesi Elvan DUMAN

Bölüm 4

SQL – Veri İşleme (Data Manipulation)

Bölüm 4 - Amaçlar



- ▶ **Veri Seçme:** SELECT ile verileri almak
- ▶ **Koşullu Filtreleme:** WHERE ile belirli kriterlere göre veri seçimi
- ▶ **Sonuçları Sıralama:** ORDER BY ile sıralama işlemleri
- ▶ **Kümeleme Fonksiyonları:** COUNT, SUM, AVG, MIN, MAX gibi fonksiyonları kullanma
- ▶ **Gruplama ve Filtreleme:** GROUP BY ile verileri gruplamak, HAVING ile grupları filtrelemek

SQL'in Genel Özellikleri



► Kolay Öğrenilebilir ve Kullanıcı Dostudur.

- SQL prosedürel değildir, neyin yapılması gerektiğini belirtmek yeterlidir.
- Okunabilir bir sözdizimi vardır (SELECT * FROM tablo_adi;).

► Platform Bağımsızdır ve Taşınabilirdir

- SQL, farklı veritabanı yönetim sistemlerinde (VTYS) çalışabilir (Oracle, MySQL, PostgreSQL, SQL Server).
- Veri aktarımı ve sorgulama açısından esneklik sağlar.

► ISO ve ANSI Tarafından Standart Hale Getirilmiştir

- SQL-92, SQL:1999, SQL:2003, SQL:2011 gibi güncellenen standartlarla sürekli gelişmektedir.

► Farklı Kullanıcı Grupları Tarafından Kullanılabilir

- Veri Tabanı Yöneticileri (DBA), yazılım geliştiriciler ve son kullanıcılar için uygundur.

► İş Zekası ve Büyük Veri ile Entegre Çalışır

- SQL, büyük veri platformlarında (BigQuery, Snowflake, Apache Hive) yaygın olarak kullanılır.

SQL'in Temel Amaçları



- ▶ **Veri Tabanı Yönetimi:** Verilerin saklanması, erişilmesini ve yönetilmesini sağlar.
- ▶ **Veri Manipülasyonu:** INSERT, UPDATE, DELETE komutlarıyla verilerin eklenmesini, değiştirilmesini ve silinmesini sağlar.
- ▶ **Veri Sorgulama:** SELECT komutu ile veritabanından istenen bilgilerin çekilmesini mümkün kılar.
- ▶ **İlişkisel Veri Modeli ile Çalışma:** JOIN işlemleri ile farklı tablolar arasındaki ilişkileri yönetir.
- ▶ **Veri Bütünlüğü ve Güvenliği:** CONSTRAINTS, GRANT, REVOKE gibi mekanizmalar ile veri güvenliğini ve bütünlüğünü korur.
- ▶ **Performans ve Optimizasyon:** INDEX, VIEW, STORED PROCEDURES gibi araçlarla veritabanı performansını artırır.
- ▶ **Standartlaşmış Bir Dil Olması:** SQL, ANSI ve ISO standartlarına uygun olarak geliştirilmiş ve farklı VTYS'lerde kullanılabilen taşınabilir bir dildir.

SQL'in Doğuşu ve Gelişimi



► 1970: İlişkisel Modelin Tanıtılması

- IBM'den Edgar F. Codd, ilişkisel veritabanı modelini tanıttı.

► 1974: SEQUEL'in Geliştirilmesi

- IBM araştırmacıları Donald D. Chamberlin ve Raymond F. Boyce, SEQUEL (Structured English Query Language) adlı dili geliştirdi.

► 1979: İlk Ticari Uygulama

- Relational Software Inc. (daha sonra Oracle Corporation), VAX bilgisayarları için ilk ticari SQL sürümü olan Oracle V2'yi piyasaya sürdü.

SQL'in Standartlaşması ve Evrimi



► 1986: ANSI Standardı

- Amerikan Ulusal Standartlar Enstitüsü (ANSI), SQL'i standart veritabanı dili olarak kabul etti.

► 1987: ISO Onayı

- Uluslararası Standardizasyon Örgütü (ISO), SQL'i uluslararası standart olarak benimsedi.

► 1992: SQL-92 (SQL2)

- SQL diline önemli eklemeler ve iyileştirmeler yapıldı.

► 1999: SQL:1999 (SQL3)

- Nesne yönelimli özellikler, XML desteği ve diğer modern veritabanı ihtiyaçlarına yanıt veren özellikler eklendi.

► 2023: SQL:2023

- En son güncelleme ile grafik verileri işleme ve JSON desteği gibi modern veri ihtiyaçlarına yönelik özellikler eklendi.

SQL'in Önemi



► Kurumsal ve Stratejik Kullanım

- SQL, IBM'in Sistem Uygulama Mimarisi gibi büyük ölçekli sistemlerin temel bir bileşeni haline gelmiştir.
- Büyük kuruluşlar (Örn: X/OPEN) için stratejik bir seçimdir.
- Amerikan hükümeti tarafından veritabanı satışlarında Federal Bilgi İşleme Standardı (FIPS) olarak kabul edilmiştir.

► Standartların Gelişimini Etkilemesi

- SQL, diğer uluslararası veri standartlarının gelişimine katkıda bulunmuştur.
- ISO Bilgi Kaynakları Rehberi Sistemi (IRDS) ve Uzak Veri Erişimi (RDA) gibi standartlar, SQL'in veri yönetimi felsefesini temel almıştır.

SQL Komutlarının Yazılması



► SQL Komutlarının Yapısı

- SQL deyimleri ayrılmış sözcükler ve kullanıcı tanımlı sözcüklerden oluşur.
- **Ayrılmış sözcükler** (Reserved Keywords), SQL'in sabit bir parçasıdır ve tam olarak yazılmalıdır. (SELECT, FROM, WHERE vb.)
- **Kullanıcı tanımlı sözcükler** (Identifiers), tablo, sütun veya veri tabanı nesnelerinin adlarını ifade eder.

► SQL Sözdizimi Kuralları

- SQL ifadeleri büyük/küçük harf duyarlı değildir, ancak literal karakter verileri (örneğin string ifadeler) duyarlıdır.
- Girinti ve satır düzeni, okunabilirliği artırır. Uzun SQL ifadeleri, hizalamaya dikkat edilerek bölünebilir.

► NF Gösterimi (Backus-Naur Formu) ve SQL

- Büyük harfler ayrılmış sözcükleri, küçük harfler kullanıcı tanımlı sözcükleri temsil eder.
- | operatörü alternatifler arasındaki seçimi gösterir.
- {} küme parantezi, zorunlu elemanları, [] köşeli parantez ise isteğe bağlı elemanları gösterir.

Sabitler (Literals)



► Sabit (Literal) Nedir?

- Sabitler, SQL ifadelerinde doğrudan kullanılan veri değerleridir.
- Değişkenlerden farklı olarak, sabitlerin değeri SQL sorgusu içinde sabittir ve değiştirilemez.

► Sabit Türleri

- **Metin (String) Sabitleri:** Tek tırnak içinde belirtilir: 'Merhaba Dünya', 'SQL Öğreniyorum'
- **Sayısal Sabitler:** Ondalık veya tam sayı olabilir: 42, 3.14, -15
- **Tarih/Zaman Sabitleri:** Standart formatlarda belirtilir: '2024-03-04', '14:30:00', '2024-03-04 14:30:00'

► Mantıksal Sabitler:

- TRUE ve FALSE gibi Boolean değerlerdir.

Örnek Kullanım: `SELECT * FROM Musteriler WHERE Ad = 'Ahmet' AND Yas > 25;`

SELECT Deyimi



SQL'de veri tabanından veri çekmek için kullanılan temel komuttur.

- **Kullanımı:** `SELECT` sütun1, sütun2 `FROM` tablo_adi;
- ▶ Tüm sütunları seçmek için `*` kullanılır:
 - **Kullanımı:** `SELECT` `*` `FROM` tablo_adi;

Filtreleme ve Sıralama

- ▶ `WHERE` koşulu ile belirli kriterlere göre veri seçilebilir:
 - `SELECT` `*` `FROM` Musteriler `WHERE` `yas` > 30;
- ▶ `ORDER BY` ile sıralama yapılabilir:
 - `SELECT` `*` `FROM` Musteriler `ORDER BY` `ad` `ASC`;

SELECT Deyimi



```
► SELECT [ALL | DISTINCT] { * | sütun1, sütun2, ... }  
FROM tablo_adi  
    [WHERE koşul]  
    [GROUP BY sütun_adı]  
    [HAVING koşul]  
    [ORDER BY sütun_adı [ASC | DESC]];
```

- ALL | DISTINCT: Varsayılan olarak ALL kullanılır. DISTINCT tekrar eden satırları kaldırır.
- GROUP BY: Verileri belirli bir sütuna göre gruplar.
- HAVING: GROUP BY ile filtreleme yapmak için kullanılır.
- **Dikkat Edilmesi Gerekenler**
 - SELECT ve FROM zorunludur, diğer ifadeler isteğe bağlıdır.
 - Komut sırası değiştirilmez, yanlış sıralama hata üretir.

SELECT Deyimi – Örnek Durum (tüm satırlar, tüm sütunlar)



- ▶ Tüm personel bilgilerini getirmek için aşağıdaki sorgu kullanılır:
- ▶ **SELECT * FROM Staff;**

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

SELECT Deyimi – Örnek Durum (Belirli Sütunlar, Tüm Satırlar)



- ▶ Belirli sütunları seçmek için sütun adları belirtilir.
- ▶ **SELECT fName, lName, position, salary**
FROM Staff;

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

- ▶ Neden Belirli Sütunları Seçmeliyiz?
 - Gereksiz veri yükünü önleyerek sorguların performansını artırır.
 - Kullanıcının ihtiyacı olan bilgileri daha net ve düzenli şekilde almasını sağlar.

DISTINCT Kullanımı



► DISTINCT Nedir?

- DISTINCT ifadesi, tekrar eden (duplicate) verileri kaldırarak yalnızca benzersiz (unique) değerleri döndürür.
- Veri kümesinde aynı değer birden fazla kez bulunuyorsa, DISTINCT bunları tek bir değer olarak gösterir.

► Örnek Kullanım: İncelenen tüm emlakların nosu

► **SELECT propertyNo**
FROM Viewing;

► **SELECT DISTINCT propertyNo**
FROM Viewing;

►

propertyNo
PA14
PG4
PG4
PA14
PG36

propertyNo
PA14
PG4
PG36

Hesaplanan Alanlar



- ▶ **SELECT fName, lName, salary/12 AS MonthlySalary**
FROM Staff;
- ▶ Çalışanların yıllık maaşlarını 12'ye bölerek aylık maaşlarını hesaplar ve MonthlySalary adıyla yeni bir sütun oluşturur.

- ▶ **SELECT fName, lName, salary, salary * 1.10 AS SalaryAfterRaise**
FROM Staff;
- ▶ Çalışan maaşlarına %10 zam ekleyerek yeni maaşları hesaplar.

Karşılaştırmalı Arama Şartı



- ▶ WHERE koşulu içinde karşılaştırma operatörleri kullanılarak belirli kriterlere göre veri filtrelenebilir.
- ▶ Kullanılan temel karşılaştırma operatörleri:
 - = (Eşittir)
 - != veya <> (Eşit değildir)
 - > (Büyüktür), < (Küçüktür)
 - >= (Büyük eşittir), <= (Küçük eşittir)
- ▶ **Örnek:** Maaşı 15.000'den fazla olan çalışanlar
- ▶ **SELECT** staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 15000;

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Karşılaştırmalı Arama Şartı



► Birleşik Karşılaştırmalar Nedir?

- Mantıksal operatörler (AND, OR, NOT) kullanılarak birden fazla koşul aynı anda uygulanabilir.
- Koşullar arasında ilişkiler kurarak daha spesifik sorgular yazmamızı sağlar.

► Temel Mantıksal Operatörler:

- **AND:** Tüm koşulların sağlanması gerekir.
- **OR:** En az bir koşulun sağlanması yeterlidir.
- **NOT:** Belirtilen koşulu sağlamayan verileri döndürür.

► Örnek: Şehri 'London' veya 'Glasgow' Olan Şubeleri Getir

► **SELECT * FROM Branch**
WHERE city = 'London' OR city = 'Glasgow';

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Aralıklı Arama Şartı



- ▶ Belirli bir değer aralığında bulunan verileri seçmek için kullanılır.
- ▶ BETWEEN ... AND ... ifadesi ile bir aralık tanımlanır.
- ▶ Örneğin: Belirli bir maaş aralığındaki çalışanları listelemek için kullanılabilir.
- ▶ **SELECT** staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

Aralıklı Arama Şartı



- ▶ BETWEEN, SQL'e anlamlı güç katmıyor. Ayrıca, BETWEEN yerine $\geq$ ve $\leq$ kullanılarak da aynı işlem yapılabilir.
- ▶ **SELECT * FROM Staff**
WHERE salary $\geq$ 10000 AND salary $\leq$ 25000;
- ▶ NOT BETWEEN ise belirtilen aralık dışındaki değerleri seçer.
- ▶ **SELECT staffNo, fName, lName, position, salary**
FROM Staff
WHERE salary NOT BETWEEN 20000 AND 30000;

Küme Üyeliği



- ▶ **IN** operatörü belirli bir değer listesinin içinde olup olmadığını kontrol etmek için kullanılırken,
- ▶ **NOT IN** ise belirli bir listeye dahil olmayan verileri seçmek için kullanılır.
- ▶ **Örnek:** Yalnızca 'Manager' ve 'Supervisor' pozisyonlarına sahip çalışanları getirir.
- ▶ **SELECT** staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Küme Üyeliği



► -- IN ile kullanım

```
SELECT * FROM Staff WHERE position IN ('Manager', 'Supervisor', 'Assistant');
```

-- OR ile kullanım

```
SELECT * FROM Staff WHERE position = 'Manager' OR position = 'Supervisor' OR position = 'Assistant';
```

► Önemli Hatırlatmalar

- Küçük veri kümelerinde IN ve OR benzer performans gösterir.
- Büyük veri kümelerinde IN, indeksleme ile daha verimli çalışabilir.
- Alt sorgular ile kullanıldığında, IN bazen **EXISTS operatörüne göre** daha yavaş olabilir.

Örüntü Eşleştirme (Pattern Matching)



- ▶ SQL'de LIKE operatörü, belirli bir desenle eşleşen satırları seçmek için kullanılır.
- ▶ LIKE Operatörü Kullanımı
 - % : Sıfır veya daha fazla karakteri temsil eder.
 - _ : Tek bir karakteri temsil eder.

Örnek Kullanımlar

- ▶ İsmi 'T' harfiyle başlayanları getirir.
 - `SELECT * FROM Owner WHERE fName LIKE 'T%';`
- ▶ Soyadı 'y' harfiyle bitenleri getirir.
 - `SELECT * FROM Owner WHERE lName LIKE '%y';`
- ▶ İsmnin içinde 'ar' geçenleri getirir
 - `SELECT * FROM Owner WHERE fName LIKE '%ar%';`
- ▶ Üçüncü karakteri r olan isimleri getir.
 - `SELECT * FROM Owner WHERE fName LIKE '__r%';`

Örüntü Eşleştirme (Pattern Matching)



► Adresinde 'Glasgow' olan tüm emlak sahiplerini bulun.

► **SELECT** ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address **LIKE** '%Glasgow%';

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Örüntü Eşleştirme (Pattern Matching)



- ▶ SQL'de NULL, bilinmeyen veya tanımsız bir değeri temsil eder ve doğrudan karşılaştırma operatörleriyle (=, !=, <, >, vb.) kullanılamaz.
 - `SELECT * FROM TableName WHERE ColumnName IS NULL;`
 - `SELECT * FROM TableName WHERE ColumnName IS NOT NULL;`
- ▶ Mülkiyet PG4'de bir yorum yapılmamış tüm gösterimlerin ayrıntılarını listeleyin.

▶ `SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo = 'PG4' AND
comment IS NULL;`

clientNo	viewDate
CR56	26-May-04

- ▶ Ters versiyonu (**IS NOT NULL**), null olmayan değerler için kullanılır.

Tek Sütun Sıralama (ORDER BY)



- ▶ Tek bir sütunu artan ya da azalan şekilde sıralamak için ORDER BY kullanılır.
- ▶ Artan sıralama (varsayılan olarak ASC kullanılır):
 - `SELECT * FROM Staff ORDER BY salary;`
 - `SELECT * FROM Staff ORDER BY salary ASC;`
- ▶ Azalan sıralama (DESC kullanılır):
 - `SELECT * FROM Staff ORDER BY salary DESC;`
- ▶ **Önemli Hatırlatmalar**
 - Metinsel değerlerde alfabetik sıralama yapılır.
 - NULL değerleri sıralarken kullanılan veritabanı yönetim sistemine bağlı olarak farklılık gösterebilir.
 - `SELECT * FROM Staff ORDER BY salary NULLS LAST;` (Null sonda, eğer başta istersek **LAST** yerine **FIRST**)

Tek Sütun Sıralama (ORDER BY)



- Tüm personel için maaşları azalan şekilde sıralanmış olarak listeleyin.
- **SELECT** staffNo, fName, lName, salary
FROM Staff
ORDER BY salary DESC;

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

Çoklu Sütun Sıralama (ORDER BY)



- ▶ SQL'de birden fazla sütuna göre sıralama yapmak için ORDER BY ifadesine birden fazla sütun eklenebilir.
- ▶ Önce birinci sütuna, sonra ikinci sütuna göre sıralama:
 - `SELECT * FROM Staff ORDER BY position, salary;`
- ▶ Farklı sıralama türleriyle sıralama:
 - `SELECT * FROM Staff ORDER BY position ASC, salary DESC;`
 - **position** sütunu **artan (A-Z)** olarak sıralanır.
 - Aynı **position** değerine sahip satırlar varsa **salary** değerine göre **azalan (büyükten küçüğe)** sıralanır.

Çoklu Sütun Sıralama (ORDER BY)



```
▶ SELECT propertyNo, type, rooms, rent  
FROM PropertyForRent  
ORDER BY type, rent DESC;
```

- ▶ Öncelikle type sütununa göre sıralama yapılır (örneğin "Apartment", "House" gibi).
- ▶ Aynı type değerine sahip satırlar rent sütununa göre azalan sıralanır (en yüksek kira bedelinden en düşüğe doğru).

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

SELECT Deyimi – Küme Fonksiyonları



- ▶ Küme fonksiyonları, tablonun belirli bir sütunu üzerinde işlem yaparak tek bir değer döndürür.
- ▶ **COUNT, MIN ve MAX** hem sayısal hem de sayısal olmayan sütunlarda kullanılabilir.
- ▶ **SUM ve AVG** yalnızca sayısal sütunlarda çalışır.
- ▶ **COUNT(*) haricinde**, tüm küme fonksiyonları **NULL değerleri dikkate almaz** ve yalnızca NULL olmayan değerler üzerinden işlem yapar.
- ▶ Örnek Sorgu: kiralık mülklerin toplam sayısını, ortalama kirasını, en düşük ve en yüksek kirasını hesaplar.

- SELECT

```
COUNT(propertyNo) AS total_properties,  
AVG(rent) AS average_rent,  
MIN(rent) AS lowest_rent,  
MAX(rent) AS highest_rent  
FROM PropertyForRent;
```

SELECT Deyimi – Küme Fonksiyonları



- ▶ COUNT(*) Bir tablodaki tüm satırları sayar, NULL veya yinelenen değerleri göz önüne almaz.
 - `SELECT COUNT(*) AS total_records FROM PropertyForRent;`
- ▶ Sütun adından önce **DISTINCT** eklenirse, tekrar eden (duplicate) değerler kaldırılır.
 - `SELECT COUNT(DISTINCT type) AS unique_property_types FROM PropertyForRent;`
- ▶ Yalnızca benzersiz kira değerleri ile ortalama hesaplayın.
 - `SELECT
AVG(rent) AS avg_rent,
AVG(DISTINCT rent) AS avg_unique_rent
FROM PropertyForRent;`

Küme Fonksiyonları ve Kullanım Kuralları



- ▶ Küme fonksiyonları nerede kullanılabilir?
 - Sadece SELECT listesinde ve HAVING cümleciğinde kullanılabilir.
- ▶ **GROUP BY** olmadan küme fonksiyonları
 - Eğer SELECT listesinde bir küme fonksiyonu varsa ve GROUP BY kullanılmıyorsa, SELECT listesinde küme fonksiyonu dışında başka sütunlar kullanılamaz!
- ▶ **Yanlış Kullanım**
 - `SELECT staffNo, COUNT(salary)`
`FROM Staff;`
 - **Hata:** staffNo bireysel değerler içerirken, COUNT(salary) küme fonksiyonu tüm satırları sayar. GROUP BY eksik olduğu için hata oluşur.
- ▶ **Doğru Kullanım:** GROUP BY eklenerek her personel için maaşların sayısı hesaplanır.
 - `SELECT staffNo, COUNT(salary)`
`FROM Staff`
`GROUP BY staffNo;`

COUNT(*) Kullanımı



- Kiralamak için ayda 350' den fazla maliyeti olan kaç emlak var?
- **SELECT** *COUNT*(*) **AS** myCount
FROM PropertyForRent
WHERE rent > 350;

myCount
5

COUNT ve SUM 'ın Kullanımı



- Yöneticilerin sayısını ve Maaşlarının toplamını bulun.
- **SELECT** *COUNT*(**staffNo**) **AS** myCount,
SUM(**salary**) **AS** mySum
FROM Staff
WHERE **position** = 'Manager';

myCount	mySum
2	54000.00

MIN, MAX, AVG'nin Kullanımı



- ▶ Minimum, maksimum ve ortalama personel maaşını bulun.
- ▶ **SELECT** *MIN*(salary) **AS** myMin,
 MAX(salary) **AS** myMax,
 AVG(salary) **AS** myAvg
FROM Staff;

▶

myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT Deyimi - Gruplama



► GROUP BY Nedir?

- GROUP BY ifadesi, benzer değerlere sahip satırları gruplamak için kullanılır.
- Genellikle küme fonksiyonları ile birlikte çalışır (COUNT, SUM, AVG, MIN, MAX).

► Örnek Sorgu

- **PropertyForRent** tablosundaki her şehir için toplam kiralanabilir mülk sayısını getirin.
- `SELECT city, COUNT(*) AS totalProperties
FROM PropertyForRent
GROUP BY city;`

	city	totalProperties
1	Aberdeen	1
2	Glasgow	4
3	London	1

SELECT Deyimi - Gruplama



- ▶ GROUP BY, verileri belirli bir sütuna göre gruplamak için kullanılır.
- ▶ HAVING, gruplama sonrası filtreleme yapmak için kullanılır. (WHERE'in aksine, gruplara uygulanır.)
- ▶ Eğer WHERE, GROUP BY ile kullanılırsa, WHERE ilk uygulanır, daha sonra gruplar kalan satırlardan oluşturulur.
- ▶ **Örnek** : Her Şubedeki personel sayısını ve toplam maaşlarını bulun.

```
SELECT branchNo,  
       COUNT(staffNo) AS myCount,  
       SUM(salary) AS mySum  
FROM Staff  
GROUP BY branchNo  
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

SELECT Deyimi - Gruplama



- ▶ HAVING cümlecği, nihai sonucu tabloda görünen grupları kısıtlamak için, GROUP BY ile kullanılmak üzere tasarlanmıştır.
- ▶ WHERE'e benzer, fakat WHERE tek tek satırları filtrelerken, HAVING grupları filtreler.
- ▶ HAVING cümlecğindeki sütun adlarının da GROUP BY listesinde görünmesi gerekir veya bir küme fonksiyonu içinde yer almalıdır.
- ▶ **Örnek** : Personel sayısı 1'den fazla her şube için, her şubedeki personel sayısını ve toplam maaşlarını bulun.

```
SELECT branchNo,  
       COUNT(staffNo) AS myCount,  
       SUM(salary) AS mySum  
FROM Staff  
GROUP BY branchNo  
HAVING COUNT(staffNo) > 1  
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Gelecek Bölümde

Alt sorgular (sub queries) konusunu görerek sorgulama yeteneklerimizi geliştireceğiz.

Bölüm sonu- Teşekkürler.

Dr. Öğr. Üyesi Elvan DUMAN – eduman@mehmetakif.edu.tr