

B Ö L Ü M

6

SQL ile Veri İşleme ve Sorgulama



6.1 Örnek Veri Tabanı: Fakülte

Bu bölümdeki sorgular, 5.6'da oluşturulan **Fakülte veri tabanı** üzerinde çalıştırılacaktır. Tabloların içeriği aşağıdadır.

Ogrenci						
OgrId PK	OgrNo	Ad	Soyad	İli	BolumId FK	Sinif
1	4051	Ali	Ay	Van	1	1
2	3042	Veli	Demir	Muş	1	2
3	2033	Elif	Yıldız	Rize	2	3
4	2024	Can	Güneş	İzmir	5	3
5	4015	Ayşe	Çelik	Kilis	2	1
6	1026	Ali	Demir	Muş	1	4
7	2067	Elif	Can	Niğde	2	3
8	1088	Murat	Temiz	Ordu	1	4

Dersler						
DersId PK	DersKodu	DersAdi	Kredi	Teori	Uyg	BolumId FK
1	ABM105	Algoritma ve Programlama	5	3	2	1
2	ABM106	Programlama Dilleri	5	3	2	1
3	ABM313	İşletim Sistemleri	3	3	0	1
4	ABM409	Yazılım Etiği	2	2	0	1
5	INT205	Dinamik	3	3	0	4
6	YMT211	Ayrık Yapılar	2	2	0	5
7	YMT217	Programlama Dilleri	4	3	2	5
8	YMT321	Ağ Sistemleri	3	3	0	5

Bolum		
BolumId PK	BolumKod	BolumAdi
1	ABM	Adli Bilişim Mühendisliği
2	EEM	Elektrik Elektronik Mühendisliği
3	ESM	Enerji Sistemleri Mühendisliği
4	INT	İnşaat Mühendisliği
5	YMT	Yazılım Mühendisliği

Dersi_Alanlar	
DersId PK, FK	OgrId PK, FK
1	1
1	2
3	6
3	1
4	8
6	4
7	4

Ogrenci ve Dersler tablolarındaki **BolumId**, Bolum tablosuna bağlanan yabancı anahtardır. **Dersi_Alanlar** tablosu hangi öğrencinin hangi dersi aldığını tutar: örneğin (1, 2) satırı, 1 numaralı dersi (ABM105) 2 numaralı öğrencinin (Veli Demir) aldığını gösterir.

6.2 Kayıt Ekleme: INSERT

Tabloya yeni kayıt eklemek için `INSERT INTO` komutu kullanılır. Önce tablo adı ve alan listesi, sonra `VALUES` ile yeni kaydın değerleri yazılır. **Metin ve tarih değerleri tek tırnak içinde** yazılır; sayılar tırnaksız yazılır.

GENEL YAZIM

```
INSERT INTO tablo_adi (alan1, alan2, ...) VALUES (deger1, deger2, ...);
```

```
-- Bolum tablosuna yeni bir bölüm ekler
INSERT INTO Bolum (BolumKod, BolumAdi) VALUES ('SUB', 'Su Ürünleri');

-- Tek komutla birden çok kayıt
INSERT INTO Ogrenci (OgrNo, Ad, Soyad, Ili, BolumId, Sinif) VALUES
    ('4051', 'Ali', 'Ay', 'Van', 1, 1),
    ('3042', 'Veli', 'Demir', 'Muş', 1, 2);

-- 8 numaralı dersi 8 numaralı öğrencinin aldığını kaydeder
INSERT INTO Dersi_Alanlar (DersId, OgrId) VALUES (8, 8);
```

- ◆ Alan listesindeki sütun sayısı ile VALUES içindeki değer sayısı **eşit ve aynı sırada** olmalıdır.
- ◆ IDENTITY tanımlı sütuna (BolumId, OgrId gibi) değer yazılmaz; sistem kendisi verir.
- ◆ Alan listesi hiç yazılmazsa tablonun **bütün** sütunları için sırasıyla değer verilmelidir.
- ◆ Yabancı anahtar nedeniyle önce ana tablo (Bolum), sonra ona bağlı tablolar (Ogrenci, Dersler), en son köprü tablo (Dersi_Alanlar) doldurulur.

6.3 Kayıtları Listeleme: SELECT

Bir tablodan kayıt çekmek için `SELECT` komutu kullanılır. Yıldız (*) tablonun bütün sütunlarını getirir; yalnızca belli sütunlar isteniyorsa adları virgülle yazılır.

```
SELECT * FROM Ogrenci; -- tüm sütunlar, tüm kayıtlar
SELECT OgrNo, Ad, Soyad FROM Ogrenci; -- yalnızca üç sütun
```

Sıralama: ORDER BY

Sonuçlar bir alana göre sıralanabilir. **ASC** artan (varsayılan), **DESC** azalan sıra demektir. Birden fazla alana göre de sıralama yapılabilir.



ORDER BY: kayıtları küçükten büyüğe ya da büyükten küçüğe dizer

```
SELECT Ad, Soyad, Sinif FROM Ogrenci
ORDER BY Sinif DESC, Soyad ASC;
```

SONUÇ

Ad	Soyad	Sinif
Ali	Demir	4
Murat	Temiz	4
Elif	Can	3
Can	Güneş	3
Elif	Yıldız	3
Veli	Demir	2
Ali	Ay	1
Ayşe	Çelik	1

Kayıtlar önce sınıfa göre büyükten küçüğe, aynı sınıftakiler ise soyada göre alfabetik sıralanmıştır.

6.4 Koşul Belirtme: WHERE, AND, OR

Yalnızca belli bir koşula uyan kayıtları getirmek için **WHERE** kullanılır. Karşılaştırmada **=**, **<>** (eşit değil), **<**, **>**, **<=**, **>=** işleçleri kullanılır.



WHERE bir elek gibidir: koşula uyanlar geçer, uymayanlar elenir

```
SELECT OgrNo, Ad, Soyad FROM Ogresci WHERE Sinif < 3;
```

SONUÇ

OgrNo	Ad	Soyad
4051	Ali	Ay
3042	Veli	Demir
4015	Ayşe	Çelik

Birden fazla koşul **AND** (ve) ile bağlanırsa **hepsinin**, **OR** (veya) ile bağlanırsa **en az birinin** sağlanması gerekir.

```
SELECT * FROM Ogresci WHERE BolumId = 1 AND Sinif = 4;
```

SONUÇ

OgrId	OgrNo	Ad	Soyad	İli	BolumId	Sinif
6	1026	Ali	Demir	Muş	1	4
8	1088	Murat	Temiz	Ordu	1	4

```
SELECT Ad, Soyad, İli FROM Ogresci WHERE İli = 'Muş' OR İli = 'Van';
```

SONUÇ

Ad	Soyad	İli
Ali	Ay	Van
Veli	Demir	Muş
Ali	Demir	Muş

AND ve OR birlikte kullanılırken **parantez** sonucu değiştirir: (a OR b) AND c ile a OR (b AND c) aynı şey değildir.

```
SELECT Ad, Soyad FROM Ogrenci
WHERE (BolumId = 2 OR BolumId = 5) AND Sinif = 3;
```

SONUÇ

Ad	Soyad
Elif	Yıldız
Can	Güneş
Elif	Can

6.5 LIKE, BETWEEN, IN ve DISTINCT**LIKE: Kalıba Göre Arama**

“Adı A ile başlayan”, “soyadı N ile biten” gibi aramalar için **LIKE** kullanılır. SQL Server’da % işareti **herhangi sayıda karakterin**, _ işareti **tek bir karakterin** yerini tutar.

Kalıp	Anlamı
LIKE 'D%'	D ile başlayanlar
LIKE '%n'	n ile bitenler
LIKE '%em%'	İçinde “em” geçenler
LIKE 'A_i'	A ile başlayıp i ile biten üç harfliler (Ali)

```
SELECT Ad, Soyad FROM Ogrenci WHERE Soyad LIKE 'D%';
```

SONUÇ

Ad	Soyad
Veli	Demir
Ali	Demir

NOT

Joker karakter veri tabanı yönetim sistemine göre değişir. SQL Server, MySQL ve PostgreSQL'de % kullanılır; Microsoft Access'te aynı iş için * kullanılır (LIKE 'D*'). Ders slaytlarındaki yıldızlı örnekler Access yazımıdır.

BETWEEN ve IN

BETWEEN a AND b belli bir aralıktaki kayıtları seçer (uç değerler dâhildir). IN (...) ise bir listedeki değerlerden herhangi birine eşit olanları seçer.

```
SELECT DersKodu, DersAdi, Kredi FROM Dersler WHERE Kredi BETWEEN 3 AND 4;
-- aynı sorgu: WHERE Kredi >= 3 AND Kredi <= 4
```

SONUÇ

DersKodu	DersAdi	Kredi
ABM313	İşletim Sistemleri	3
INT205	Dinamik	3
YMT217	Programlama Dilleri	4
YMT321	Ağ Sistemleri	3

```
SELECT Ad, Soyad, Ili FROM Ogrenci WHERE Ili IN ('Muş', 'Ordu');
```

SONUÇ

Ad	Soyad	Ili
Veli	Demir	Muş
Ali	Demir	Muş
Murat	Temiz	Ordu

DISTINCT: Tekrarları Bir Kez Getirme

Tekrarlayan değerleri yalnızca bir kez listelemek için DISTINCT kullanılır.

```
SELECT DISTINCT Ili FROM Ogrenci;
```

Sekiz öğrenci vardır, ancak Muş iki kez geçtiği için sonuçta **yedi** satır döner: Van, Muş, Rize, İzmir, Kilis, Niğde, Ordu.

6.6 Güncelleme ve Silme: UPDATE, DELETE

UPDATE

Tablodaki bir veya birden fazla sütunun değerini değiştirmek için **UPDATE** kullanılır.

GENEL YAZIM

```
UPDATE tablo_adi SET kolon1 = deger1, kolon2 = deger2 WHERE kosul;
```

```
-- 4051 numaralı öğrenciyi 2. sınıfa geçirir
UPDATE Ogranci SET Sinif = 2 WHERE OgrNo = '4051';

-- Ders kodu ABM ile başlayan tüm derslerin kredisini 5 yapar
UPDATE Dersler SET Kredi = 5 WHERE DersKodu LIKE 'ABM%';
```

DELETE

Tablodaki kayıt ya da kayıtları silmek için **DELETE** kullanılır.

```
DELETE FROM Ogranci WHERE Ad = 'Ayşe'; -- adı Ayşe olan öğrenci(ler)
silinir
```

DİKKAT

UPDATE ve DELETE komutlarında **WHERE yazılmazsa işlem tablodaki bütün satırlara uygulanır**: bütün kayıtlar güncellenir ya da silinir. Bu komutları çalıştırmadan önce aynı WHERE koşuluyla bir SELECT çalıştırıp hangi satırların etkileneceğine bakmak iyi bir alışkanlıktır.

Yabancı Anahtar Silmeyi Engeller

```
DELETE FROM Dersler WHERE BolumId = 5;
```

Bu komut YMT bölümünün üç dersini (YMT211, YMT217, YMT321) silmek ister. Ancak Dersi_Alanlar tablosunda 6 ve 7 numaralı derslere ait kayıtlar bulunduğu için veri tabanı **yabancı anahtar hatası** verir ve silme yapılmaz. Önce bağlı kayıtlar silinmeli, sonra dersler silinmelidir:

```
DELETE FROM Dersi_Alanlar WHERE DersId IN (6, 7, 8);
DELETE FROM Dersler WHERE BolumId = 5;
```

Bu davranış, 1. bölümde anlatılan **veri tutarlılığının** (referential integrity) uygulamadaki karşılığıdır.

6.7 Hazır Fonksiyonlar: COUNT, SUM, AVG, MIN, MAX

Fonksiyon	Görevi	Örnek	Sonuç
COUNT(*)	Kayıt sayısını verir.	SELECT COUNT(*) FROM Ogrenci;	8
SUM(alan)	Değerleri toplar.	SELECT SUM(Kredi) FROM Dersler;	27
AVG(alan)	Ortalamayı verir.	SELECT AVG(Kredi) FROM Dersler;	3
MAX(alan)	En büyük değeri verir.	SELECT MAX(Kredi) FROM Dersler;	5
MIN(alan)	En küçük değeri verir.	SELECT MIN(Kredi) FROM Dersler;	2

Sonuç sütununa **AS** ile bir ad (takma ad) verilebilir; koşul da eklenebilir:

```
SELECT COUNT(*) AS OgrenciSayisi FROM Ogrenci WHERE BolumId = 1; -- 4
```

TAM SAYI BÖLMESİNE DİKKAT

Kredilerin toplamı $5 + 5 + 3 + 2 + 3 + 2 + 4 + 3 = 27$, ders sayısı 8'dir; gerçek ortalama $27 / 8 = 3,375$ eder. Ancak Kredi sütunu **INT** olduğu için SQL Server tam sayı bölmesi yapar ve **AVG(Kredi)** sonucu **3** döner. Ondalıklı sonuç için **AVG(Kredi * 1.0)** yazılır.

6.8 Tabloları Birleştirme: JOIN

İki farklı tablodaki kayıtları eşleştirmek için JOIN kullanılır. Normalizasyonla ayrılan veriler raporlanırken JOIN ile yeniden bir araya getirilir. Tablolar genellikle birinin birincil anahtarı ile diğerinin yabancı anahtarı üzerinden bağlanır.



JOIN: iki ayrı parça, ortak noktalarından birleştirilir

GENEL YAZIM

```
SELECT tablo1.alan1, tablo2.alan2
FROM tablo1
INNER JOIN tablo2 ON tablo1.ortak_alan = tablo2.ortak_alan;
```

INNER JOIN

Yalnızca iki tabloda da **eşleşen** kayıtları getirir.

```
SELECT Ogrenci.Ad, Ogrenci.Soyad, Bolum.BolumKod
FROM Ogrenci
INNER JOIN Bolum ON Ogrenci.BolumId = Bolum.BolumId;
```

SONUÇ

Ad	Soyad	BolumKod
Ali	Ay	ABM
Veli	Demir	ABM
Elif	Yıldız	EEM
Can	Güneş	YMT
Ayşe	Çelik	EEM
Ali	Demir	ABM
Elif	Can	EEM
Murat	Temiz	ABM

Ogrenci.BolumId

Ogrenci tablosunun BolumId sütunu

Bolum.BolumId

Bolum tablosunun BolumId sütunu

ON Öğrenci.BolumId = Bolum.BolumId

Öğrenci ve Bolum tablolarında BolumId değerleri eşit olan satırlar eşleştirilir.

Her öğrencinin BolumId değeri Bolum tablosunda aranır ve o bölümün kodu getirilir. Sekiz öğrenci olduğu için sekiz satır döner. ESM ve INT bölümlerinde öğrenci olmadığı için bu bölümler sonuçta **görünmez**.

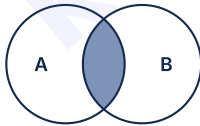
Üç Tabloyu Birleştirme

Hangi öğrencinin hangi dersi aldığı, köprü tablo üzerinden iki JOIN ile bulunur. Tablo adlarını kısaltmak için takma ad verilebilir (Öğrenci O gibi).

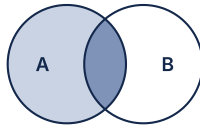
```
SELECT O.Ad, O.Soyad, D.DersAdi
FROM Dersi_Alanlar DA
INNER JOIN Öğrenci O ON DA.OgrId = O.OgrId
INNER JOIN Dersler D ON DA.DersId = D.DersId;
```

SONUÇ

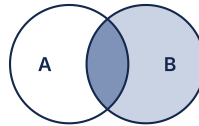
Ad	Soyad	DersAdi
Ali	Ay	Algoritma ve Programlama
Veli	Demir	Algoritma ve Programlama
Ali	Demir	İşletim Sistemleri
Ali	Ay	İşletim Sistemleri
Murat	Temiz	Yazılım Etiği
Can	Güneş	Ayrık Yapılar
Can	Güneş	Programlama Dilleri

LEFT, RIGHT ve FULL (OUTER) JOIN

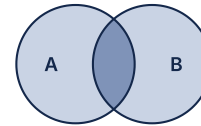
INNER JOIN
yalnızca eşleşenler



LEFT JOIN
soldaki tablonun tamamı



RIGHT JOIN
sağdaki tablonun tamamı



FULL JOIN
iki tablonun tamamı

JOIN türleri: A soldaki, B sağdaki tablodur

LEFT (OUTER) JOIN

Soldaki tablonun **bütün** kayıtlarını getirir; sağdaki tabloda eşleşeni olmayanların sağ taraftan gelen sütunları **NULL** olur.

RIGHT (OUTER) JOIN

Sağdaki tablonun bütün kayıtlarını getirir; solda eşleşeni olmayanlar **NULL** ile tamamlanır.

FULL (OUTER) JOIN

İki tablonun da bütün kayıtlarını getirir; eşleşmeyen taraflar NULL olur.

```
SELECT B.BolumKod, O.Ad
FROM Bolum B
LEFT JOIN Ogrenci O ON B.BolumId = O.BolumId;
```

SONUÇ

BolumKod	Ad
ABM	Ali
ABM	Veli
ABM	Ali
ABM	Murat
EEM	Elif
EEM	Ayşe
EEM	Elif
ESM	NULL
INT	NULL
YMT	Can

Soldaki tablo Bolum olduğu için beş bölümün hepsi listelenir. Öğrencisi olmayan ESM ve INT bölümleri de sonuçta yer alır; bu satırlarda Ad sütunu NULL'dır. Aynı sorgu INNER JOIN ile yazılsaydı bu iki satır gelmezdi.

6.9**Gruplama: GROUP BY ve HAVING**

GROUP BY, tablodaki satırları belirtilen sütuna göre ortak özelliklerine göre paketler (gruplar). Her grup için COUNT, SUM, AVG gibi fonksiyonlarla tek bir özet değer hesaplanır.

MADENİ PARA BENZETMESİ

Elinizde karışık bir madeni para yığını var. Bunları masanın üzerinde 1 TL'likler, 50 kuruşluklar ve 25 kuruşluklar olarak ayrı ayrı desteliyorsunuz. Yaptığınız iş bir GROUP BY'dır. Her destenin kaç para ettiğini saymak ise COUNT ya da SUM'dır.



Solda karışık paralar, sağda türlerine göre gruplanmış desteler (GROUP BY)

```
SELECT BolumId, COUNT(*) AS OgrenciSayisi
FROM Ogrenci
GROUP BY BolumId;
```

SONUÇ

BolumId	OgrenciSayisi
1	4
2	3
5	1

SQL motoru önce bütün öğrencileri bölüm numaralarına göre gruplara ayırır (1'ler bir yere, 2'ler bir yere), sonra her gruptaki öğrencileri sayıp özet tabloyu çıkarır.

HAVING: Gruplara Şart Koşma

HAVING, **WHERE** ile aynı işi yapar; tek farkı uygulandığı yerdir: **WHERE** henüz gruplanmamış ham satırları, **HAVING** ise **GROUP BY**'dan sonra oluşan özet grupları filtreler.

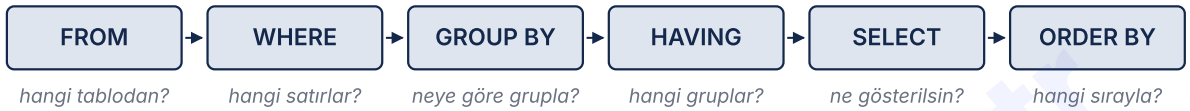
```
SELECT BolumId, COUNT(*) AS OgrenciSayisi
FROM Ogrenci
GROUP BY BolumId
HAVING COUNT(*) > 1;
```

SONUÇ

BolumId	OgrenciSayisi
1	4
2	3

KRİTİK KURAL

Burada `WHERE COUNT(*) > 1` yazılamaz; SQL hata verir. Çünkü WHERE çalıştığında ortada henüz gruplanmış veya sayılmış bir veri yoktur. Sayım bittikten sonra filtreleme yapılacağı için HAVING kullanılır.

SQL'in Çalışma Sırası

Bir SELECT sorgusunun bölümleri bu sırayla çalışır

WHERE tek tek kişileri eler: “Adı Elif olanlar ayağa kalsın.” **HAVING grubu toptan eler:** “Yalnızca mevcudu 5’ten fazla olan gruplar kalsın.”

```

-- Kredisi en az 3 olan dersler içinde, toplam kredisi 5'ten büyük olan
bölümler
SELECT BolumId, SUM(Kredi) AS ToplamKredi
FROM Dersler
WHERE Kredi >= 3 -- önce satırlar elenir
GROUP BY BolumId -- sonra gruplanır
HAVING SUM(Kredi) > 5 -- sonra gruplar elenir
ORDER BY ToplamKredi DESC; -- en son sıralanır
  
```

SONUÇ

BolumId	ToplamKredi
1	13
5	7

6.10 İç İçe SELECT (Alt Sorgu)

Bir sorgunun sonucu, başka bir sorgunun koşulunda kullanılabilir. İçteki sorgu önce çalışır; sonucu dıştaki sorguya verilir.

```

-- En az bir ders alan öğrenciler
SELECT Ad, Soyad FROM Ogranci
WHERE OgrId IN (SELECT OgrId FROM Dersi_Alanlar);
  
```

SONUÇ

Ad	Soyad
Ali	Ay
Veli	Demir
Can	Güneş
Ali	Demir
Murat	Temiz

```
-- Kredisi ortalamanın üzerinde olan dersler
SELECT DersKodu, Kredi FROM Dersler
WHERE Kredi > (SELECT AVG(Kredi) FROM Dersler);
```

SONUÇ

DersKodu	Kredi
ABM105	5
ABM106	5
YMT217	4

6.11 Görüntü (VIEW): Sanal Tablo**TANIM**

VIEW (görüntü, sanal tablo); veri tabanında fiziksel olarak yer kaplamayan, arka planda bir SQL sorgusunun kaydedilmiş hâlidir. VIEW “içi veriyle dolu bir kutu” değil, “veriye bakan akıllı bir pencere”dir.



VIEW bir pencere gibidir: büyük verinin yalnızca gösterilmek istenen bölümünü gösterir

GENEL YAZIM

```
CREATE VIEW view_adi AS
SELECT kolon1, kolon2 FROM tablo_adi WHERE kosul;
```

```
CREATE VIEW ABM_Ogrencileri AS
SELECT OgrNo, Ad, Soyad, Sinif FROM Ogrenci WHERE BolumId = 1;
```

Bu komut çalıştırıldığında ABM_Ogrencileri adında yeni bir fiziksel tablo açılmaz; yalnızca sorgu veri tabanına kaydedilir. Sonradan bir tablo gibi çağrılır:

```
SELECT * FROM ABM_Ogrencileri;
```

Var olan bir görüntünün sorgusunu değiştirmek için **ALTER VIEW**, silmek için **DROP VIEW** kullanılır:

```
ALTER VIEW ABM_Ogrencileri AS
SELECT OgrNo, Ad, Soyad FROM Ogrenci WHERE BolumId = 1 AND Sinif = 4;
```

Neden VIEW Kullanılır?**Veri güvenliği**

Kullanıcıya tablonun tamamı yerine yalnızca görmesi gereken sütunlar ve satırlar gösterilir.

Sorgu basitliği

Dört beş tabloyu JOIN ile bağlayan uzun bir sorgu bir kez VIEW yapılır; sonra tek satırla (**SELECT * FROM view_adi**) çağrılır. Kod tekrarını önler.

Yalıtım

Arka plandaki tablonun adı veya yapısı değişse bile, VIEW üzerinden bağlanan raporlama ekranları ve uygulamalar bu değişiklikten etkilenmez.

Özellik	Normal tablo	Sanal tablo (VIEW)
Fiziksel alan kaplar mı?	Evet, diskte yer tutar.	Hayır, yalnızca sorgu tanımı saklanır.
İçinde veri barındırır mı?	Evet, satırlar fizikseldir.	Hayır, veriyi ana tablodan anlık çeker.
Ana tablo değişirse?	Etkilenmez.	VIEW'un gösterdiği veri de anında güncellenir.