VERİTABANI YÖNETİM SİSTEMLERİ

Dr. Öğr. Üyesi
Ender Şahinaslan

BÖLÜM -8-

VERİLERİ GRUPLAYARAK ANALİZ ETMEK

GENEL BAKIŞ…

Grup fonksiyonlarının tanımlanması,

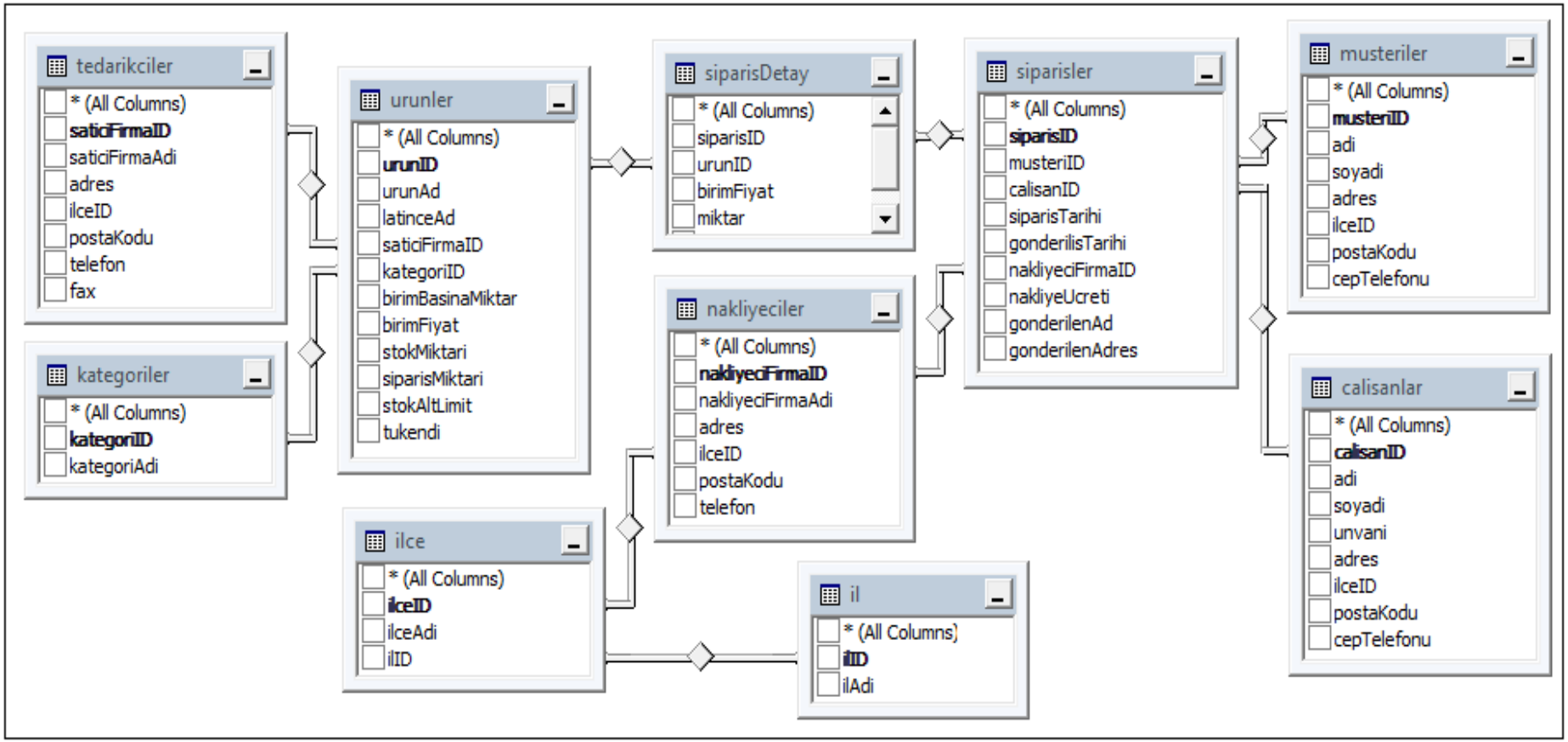
Gruplama işlemlerini,

Gruplama işlemlerinde dikkat edilecek noktalar,

Birden fazla sütuna göre gruplama yapmak,

Grup koşullarının tanımlanması.

KULLANILACAK VERİTABANI YAPISI



8.1. DISTINCT KOMUTU

SQL içerisinde bir tabloda birden fazla kayıt olabilmektedir.

Eğer aynı satırların listelenmesi gerekirse bu satırların bir defa yazılması gerekmektedir. Böyle bir durumda **DISTINCT** komutu kullanılmak zorundadır.

Siparişleri hangi Nakliye firmalarının taşıdığını ekrana listeleyen SQL ifadesi şöyle olacaktır:

	Nakliyeci Firma
1	Aras Kargo
2	Aras Kargo
3	Aras Kargo
4	Aras Kargo
5	Aras Kargo
6	Aras Kargo
7	Aras Kargo
8	Aras Kargo
9	Aras Kargo
10	Aras Kargo
11	Aras Kargo
12	Aras Kargo
13	Yurtiçi Kargo
14	Yurtiçi Kargo
15	Yurtiçi Kargo

```
SELECT nakliyeciFirmaAdi FROM siparisler, nakliyeciler
```

8.1. DISTINCT KOMUTU (DEVAM...)

Dikkat edilecek olursa nakliyeci firma adları tekrar etmiştir. Bunu engellemek için şöyle bir SQL ifadesi yazılması gerekmektedir.

```
SELECT DISTINCT nakliyeciFirmaAdi
    FROM siparisler, nakliyeciler
```

	Nakliyeci Firma
1	Aras Kargo
2	DHL
3	MNG Kargo
4	Sürat Kargo
5	UPS
6	Yurtiçi Kargo

8.2. JOIN KOMUTU

Şu ana kadar anlatmış olduğumuz sorgular sadece tek tablo kullanılarak gerçekleştirildi. Bazı durumlarda tek bir tablo ile çalışmak yeterli olmayacaktır.

Böyle bir durumda **birleştirme/eşleştirme** adını verdiğimiz bir yapı kullanılmak zorundadır. Bu işlemin gerçekleştirilmesi için **JOIN komutu** kullanılmaktadır.

8.2. JOIN KOMUTU (DEVAM...)

4 temel JOIN türü vardır:

- **INNER** JOIN
- **LEFT** JOIN
- **RIGHT** JOIN
- **CROSS** JOIN

8.2. JOIN KOMUTU (DEVAM...)

JOIN işlemini anlatmak için "**kategoriler**" ve "**urunler**" tablolarını göz önüne alalım. Bu tablolarda yer alan kayıtlar aşağıda görüldüğü şekildedir.

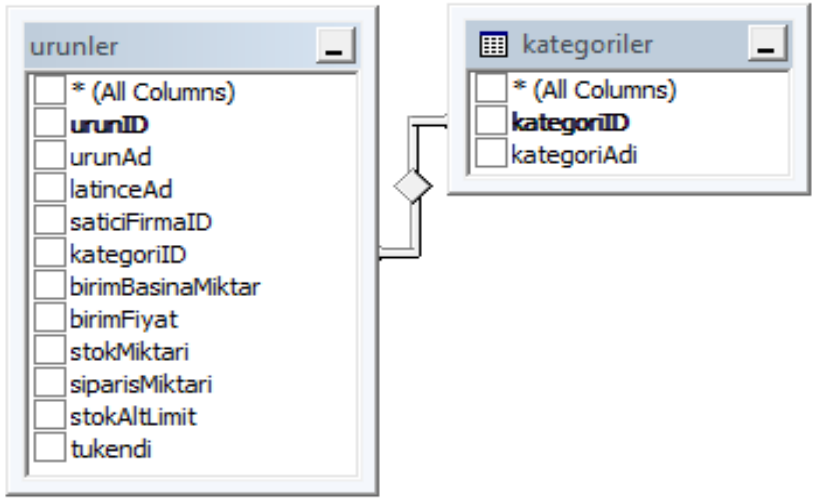
kategoriID	kategoriAdi
1	Buket Çiçekler
2	Aranjmanlar
3	Saksı Çiçekler
4	Yapay Çiçekler

urunID	urunAd	latinceAd	saticiFirmaID	kategoriID	birimBasinaMiktar	birimFiyat	stokMiktari	siparisMiktari	stokAltLimit	tukendi
1	Kırmızı Gül	Roses	1	1	10	57,0000	20	10	5	False
2	Kırmızı Karanfil	Dianthus	2	1	15	39,0000	25	5	10	False
3	Lilyum	Antirrhinum	1	1	3	75,0000	15	10	20	False
4	Barış Çiçeği	Spathiphyllum	3	3	1	55,0000	5	5	5	False
5	Orkide	Orchidaceae	3	3	2	125,0000	35	3	12	False
6	Antoryum	Anthurium	2	3	1	70,0000	30	1	7	False
7	Benjamin	Şeflera	3	3	1	65,0000	15	4	10	False
8	Yucca	Yucca Gloriosa	1	3	1	65,0000	10	5	5	False
9	Şebboy	Cheiranıhus	1	1	15	35,0000	40	3	20	False
10	Lila Orkide	Orchidaceae	4	4	1	55,0000	50	5	15	False
11	Beyaz Tutku	Anthurium	4	4	1	75,0000	65	4	10	True
12	Beyaz Orkide	Orchidaceae	4	4	3	119,0000	5	2	3	False

8.2. JOIN KOMUTU (DEVAM...)

- Dikkat edilirse, "kategoriler" tablosunda ilgili ürünün bir kategori numarası ve ait olduğu kategori adı bulunmaktadır.

- Bir ürüne ilişkin detaylar bu tabloda verilmemiş olup, gerekli diğer detaylar "urunler" tablosunda verilmiştir. Bu iki tablo birbirlerine **'kategoriID'** ile bağlıdır.



8.2. JOIN KOMUTU (DEVAM...)

- Şimdi bu iki tabloyu kullanarak iki tablo arasındaki bağlantıyı da gösteren SQL ifadesini şöyle yazabiliriz.

```
SELECT K.kategoriID, K.kategoriAdi, U.urunAd
FROM kategoriler K, urunler U
WHERE K.kategoriID = U.kategoriID
```

8.2. JOIN KOMUTU (DEVAM...)

```
SELECT K.kategoriID, K.kategoriAdi, U.urunAd
FROM kategoriler K, urunler U
WHERE K.kategoriID = U.kategoriID
```

- Bu SQL cümlesinde tablolara bir takma ad verilmiştir.
- Buradaki "**WHERE K.kategoriID = U.kategoriID**" cümlesi ile bir birleştirme işlemi gerçekleştirilmiştir. Yani "kategoriler" tablosundaki 'kategoriID' ile "urunler" tablosundaki 'kategoriID' eşit ise kayıtlar listelenecektir.

8.2. JOIN KOMUTU (DEVAM...)

```
SELECT K.kategoriID, K.kategoriAdi, U.urunAd
FROM kategoriler K, urunler U
WHERE K.kategoriID = U.kategoriID
```

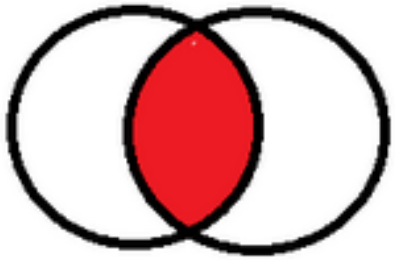
- Bu SQL ifade çalıştırıldığında yandaki liste elde edilecektir.

kategoriID	kategoriAdi	urunAd
1	Buket Çiçekler	Kırmızı Gül
1	Buket Çiçekler	Kırmızı Karanfil
1	Buket Çiçekler	Lilyum
3	Saksı Çiçekler	Barış Çiçeği
3	Saksı Çiçekler	Orkide
3	Saksı Çiçekler	Antoryum
3	Saksı Çiçekler	Benjamin
3	Saksı Çiçekler	Yucca
1	Buket Çiçekler	Şebboy
4	Yapay Çiçekler	Lila Orkide
4	Yapay Çiçekler	Beyaz Tutku
4	Yapay Çiçekler	Beyaz Orkide

8.2. JOIN KOMUTU (DEVAM...)

- İki tabloyu birleştirme işlemini INNER JOIN komutunu kullanarak gerçekleştirebiliriz.
- INNER JOIN en çok kullanılan JOIN türüdür ve her iki tablodaki ortak kayıtları döndürür.

  - **INNER JOIN** başka bir ifade ile iki tablonun **kesişimini** döndürür.



inner join

8.2. JOIN KOMUTU (DEVAM...)

- Az önceki işlemi **INNER JOIN** komutu ile gerçekleştirmek istersek **SQL** cümlemiz aşağıdaki gibi olacaktır.

```
SELECT urunAd, kategoriAdi
FROM kategoriler
      INNER JOIN urunler
         ON kategoriler.kategoriID = urunler.kategoriID
```

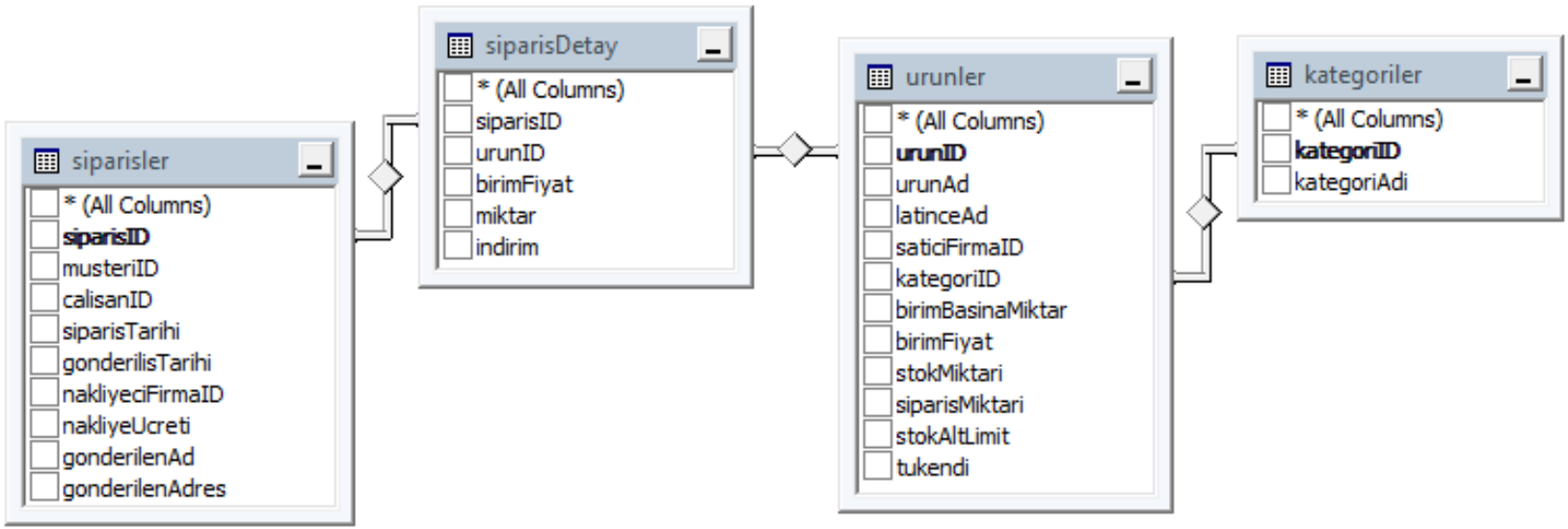
- Burada INNER JOIN ifadesi **her iki tabloda da ortak olan değerlere** bakarak eşit/aynı olanları ekrana listeleyecektir.

8.2. JOIN KOMUTU (DEVAM...)

- INNER JOIN işlemi her türlü FROM yan cümlesi ile kullanılabilir. En sık kullanılan birleştirme türüdür.
- Her iki tabloda eşleşen değerler var ise birleştirme işlemi gerçekleştirilir ve kayıtlar ekrana listelenir.

8.2. JOIN KOMUTU (DEVAM...)

- INNER JOIN işlemini daha iyi anlatabilmek için daha karmaşık bir örnek yapalım. Aşağıdaki ilişkiyi göz önüne alalım.



8.2. JOIN KOMUTU (DEVAM...)

- Dikkat edilirse, birden fazla tablo birbirlerine farklı anahtarlar kullanılarak bağlanmış durumdadırlar. **INNER JOIN komutu** bu tablolarda ortak olan değerler kullanılarak birleştirme işlemini çok kolay bir şekilde yerine getirmektedir.
- Bu dört tablonun INNER JOIN komutu kullanılarak yazılan SQL cümlesi şöyle olacaktır:

```
SELECT *
FROM (kategoriler INNER JOIN urunler
                  ON kategoriler.kategoriID = urunler.kategoriID)
      INNER JOIN
      (siparisler INNER JOIN siparisDetay
                  ON siparisler.siparisID = siparisDetay.siparisID)
      ON urunler.urunID = siparisDetay.urunID
```

8.2. JOIN KOMUTU (DEVAM...)

- Bu SQL ifadesinin çalıştırıldıktan sonraki görüntüsü aşağıdaki gibidir.

kategoriID	kategoriAdi	urunID	urunAd	latinceAd	saticiFirmaID	kategoriID	birimBasinaMiktar	birimFiyat	stokMiktari	siparisMiktari	stokAltLimit
1	Buket Çiçekler	1	Kırmızı Gül	Roses	1	1	10	57,00	20	10	5
3	Saksı Çiçekler	5	Orkide	Orchidaceae	3	3	2	125,00	35	3	12
3	Saksı Çiçekler	4	Barış Çiçeği	Spathiphyllum	3	3	1	55,00	5	5	5
3	Saksı Çiçekler	7	Benjamin	Şeflera	3	3	1	65,00	15	4	10
3	Saksı Çiçekler	6	Antoryum	Anthurium	2	3	1	70,00	30	1	7
1	Buket Çiçekler	3	Lilyum	Antirrhinum	1	1	3	75,00	15	10	20
1	Buket Çiçekler	1	Kırmızı Gül	Roses	1	1	10	57,00	20	10	5

8.2. JOIN KOMUTU (DEVAM...)

- Aşağıdaki söz dizimi kullanılarak **INNER JOIN ifadesi içerisinde** **ON yan cümleleri** de yazılabilir.

SELECT alanlar

FROM tablo1 **INNER JOIN** tablo2

ON tablo1.alan1 **karşılaştırma** tablo2.alan1 **AND**

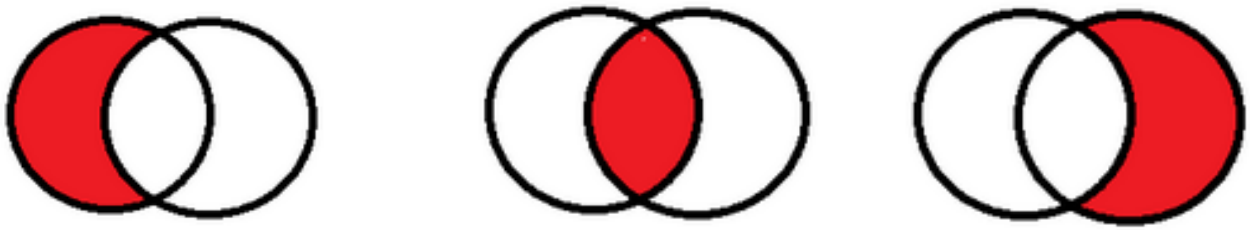
ON tablo1.alan2 **karşılaştırma** tablo2.alan2 **OR**

ON tablo1.alan3 **karşılaştırma** tablo2.alan3

8.2. JOIN KOMUTU (DEVAM...)

- Eğer bir tablodaki tüm kayıtlar ile diğer tablodaki birleştirme koşulunu sağlayan kayıtları döndürmek istenilirse **LEFT JOIN** veya **RIGHT JOIN** kullanılır.
- Örneğin: tüm kategorileri (hiçbir ürüne sahip olmayanlar da dahil) ve bir kategori ile eşleşmiş ürünleri sorgulamak istersek:

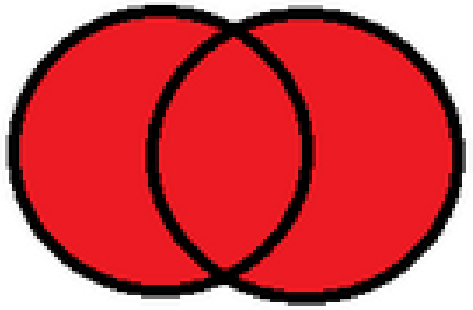
```
SELECT *
FROM (kategoriler LEFT JOIN urunler
                  ON kategoriler.kategoriID = urunler.kategoriID)
```



left outer join | inner join | right outer join

8.2. JOIN KOMUTU (DEVAM...)

- **CROSS JOIN** ile bağladığımız tüm tablolar herhangi bir kısıtlamaya uğramadan **tüm eşleşmeleri** listeler.



cross join

```
SELECT *
FROM kategoriler CROSS JOIN urunler
```

8.3. GRUPLANDIRMA FONKSİYONLARI

- Bazı durumlarda belli bir koşula sahip bilgileri sorgulamak için SQL ifadeleri kullanılabilir. Bu gibi durumlarda gruplama fonksiyonlarından yararlanılmaktadır.
- Örneğin bir ürünü, ürün kategorilerine göre gruplayıp ardından bu gruplar içerisinde sorgu yapmak isteyebiliriz. Bu işlemi gerçekleştirmek için **GROUP BY** fonksiyonu kullanılmaktadır.

8.3. GRUPLANDIRMA FONKSİYONLARI (DEVAM...)

- **GROUP BY fonksiyonu** belli bir alana göre gruplama işlemi yapmak için kullanılmaktadır. Genel kullanımı şu şekildedir.

```
SELECT Alanlar
    FROM Tablo_Adı
        WHERE Şart_İfadeleri
            GROUP BY Sütun/Sütunlar
```

8.3. GRUPLANDIRMA FONKSİYONLARI (DEVAM...)

Bu fonksiyonun kullanımına örnek verelim.

- "siparisler" tablosunu göz önüne alarak her bir nakliye firmasının kaç kere nakliye yaptığını bulmak için **SQL ifade**miz şöyle olacaktır:

```
SELECT nakliyeciFirmaID, COUNT (*) AS 'Nakliye Sayısı'
FROM siparisler
GROUP BY nakliyeciFirmaID
```

8.3. GRUPLANDIRMA FONKSİYONLARI (DEVAM...)

- Bu SQL deyimi çalıştırıldığında aşağıdaki sonuç elde edilecektir.

	nakliyeciFirmaID	Nakliye Sayısı
1	1	1
2	2	5
3	3	2
4	4	3
5	5	5
6	6	1

8.4. HAVING DEYİMİ

HAVING deyimi sorguda bir koşul belirtilmesi gerekirse kullanılmaktadır. Daha önceki slayttan hatırlayacağınız gibi gruplandırma işlemini gerçekleştirmek için **GROUP BY** fonksiyonunu kullanmıştık.

GROUP BY fonksiyonunda bir koşula ihtiyaç var ise **HAVING** deyimi kullanılmaktadır.

8.4. HAVING DEYİMİ (DEVAM...)

HAVING deyiminin genel kullanım biçimi aşağıdaki gibidir.

```
SELECT Alan_Adları
    FROM Tablo/Tablolar
        [WHERE Şart/Şartlar]
            [GROUP BY Sütunlar]
                [HAVING Grup_Kısıtlaması]
```

HAVING deyimi, GROUP BY fonksiyonu olmadan kullanılamaz.

8.4. HAVING DEYİMİ (DEVAM...)

HAVING deyiminin ardından **SUM, COUNT, MAX, MIN** ve **AVG** gibi fonksiyonlar gelmelidir.

- **HAVING deyimi** sadece gruplanmış veriler üzerindeki işlemlerde geçerlidir.

WHERE komutu ise, tablonun tek satırı üzerinde işlem yapıldığında kullanılabilmektedir.

- Bazı durumlarda bir SQL sorgusunda WHERE ve HAVING deyimleri birlikte kullanılabilmektedir.

8.4. HAVING DEYİMİ (DEVAM...)

HAVING deyiminin kullanımına bir örnek verelim.

Sipariş detayları tablosunda 1 den fazla sipariş edilen ürünlerin, ürün numaraları ve sayısını gösteren SQL ifadesi şöyle olacaktır:

```
SELECT urunID, COUNT (*) AS 'Sipariş Sayısı'
FROM siparisDetay
GROUP BY urunID
HAVING COUNT(*)>1
```

8.4. HAVING DEYİMİ (DEVAM...)

Bu sorgu, öncelikle siparisDetay tablosunu urunID'lerine göre gruplandırır. Daha sonra gruplamış olduğu bu urunID'lerini COUNT(*) fonksiyonunu kullanarak sayar. Eğer 1'den büyük ise 'urunID' ve verilen sipariş sayısını kullanıcıya gösterir.

Sorgu çalıştırıldığında aşağıdaki sonuç elde edilir.

	urunID	Sipariş Sayısı
1	1	4
2	3	2
3	4	2
4	5	4
5	7	3

8.4. HAVING DEYİMİ (DEVAM...)

Başka bir örnek verelim. Bu örneğimizde birden fazla tablo kullanılacaktır.

Toplam sipariş tutarı 100 TL' nin üzerinde olan müşterilerin vermiş oldukları toplam sipariş sayısını, müşteri numaralarını ve toplam sipariş tutarlarını görüntüleyen SQL ifadesi şöyle olacaktır:

```
SELECT siparisler .musteriID, COUNT (*) AS 'Sipariş Sayısı',
            SUM (siparisDetay.birimFiyat*siparisDetay.miktar)
            AS 'Toplam Sipariş Tutarı'
FROM siparisler  INNER JOIN siparisDetay
        ON siparisler.siparisID = siparisDetay.siparisID
GROUP BY siparisler.musteriID
HAVING SUM (siparisDetay.birimFiyat*siparisDetay.miktar) > 100
```

8.4. HAVING DEYİMİ (DEVAM...)

SQL sorgusu çalıştırıldığında aşağıdaki liste görüntülenecektir.

musteriID	Toplam Sipariş Sayısı	Toplam Sipariş Miktarı
1	1	228
3	1	150
4	1	110
5	1	125
7	1	225

8.4. HAVING DEYİMİ (DEVAM...)

Ortalama sipariş tutarı 50 TL nin üzerinde olan tüm müşterilerin vermiş oldukları toplam sipariş sayısı, toplam sipariş tutarı ve ortalama sipariş tutarını gösteren SQL ifadesi şöyle olacaktır.

```
SELECT siparisler .musteriID, COUNT(*) AS 'Sipariş Sayısı',
        SUM (siparisDetay.birimFiyat*siparisDetay.miktar)
        AS 'Toplam Sipariş Tutarı',
        AVG (siparisDetay.birimFiyat*siparisDetay.miktar)
        AS 'Ortalama Sipariş Tutarı'
FROM siparisler INNER JOIN siparisDetay ON
        siparisler.siparisID = siparisDetay.siparisID
GROUP BY siparisler.musteriID
HAVING AVG (siparisDetay.birimFiyat*siparisDetay.miktar) > 50
```

NOT: Bu sorgular yazılırken satici veritabanında Siparişler ve Sipariş Detayları tablolarını kullanınız.

8.4. HAVING DEYİMİ (DEVAM...)

Bu sorguda "siparisler" ve "siparişDetay" tabloları birleştirilerek, müsterilD'lerine göre gruplanmıştır. Daha sonra HAVING kısmında AVG ile ortalama sipariş tutarları hesaplanarak 50 TL den büyük ise listelenmesi sağlanmıştır. Sorgu çalıştırıldığında aşağıdaki sonuç elde edilecektir.

musterilD	Toplam Sipariş Sayısı	Toplam Sipariş Miktarı	Ortalama Sipariş Miktarı
1	3	549	183
3	2	462	231
4	1	110	110
5	1	125	125
6	2	320	160
7	2	300	150
8	2	332	166

8.4. HAVING DEYİMİ (DEVAM...)

HAVING ve diğer gruplama fonksiyonlarının kullanımına dikkat etmek gerekir. Grup koşulları WHERE ifadesi ile birlikte kullanılamazlar.

Örneğin bir bölüm bazında yapılacak olan bir işlemde bölümdeki personelin ücret ortalaması 1000 TL den büyük olanları listelemek istersek aşağıdaki kullanım **yanlış** olacaktır.

```
SELECT bolumNo, AVG(ucret)
    FROM personel
        WHERE AVG(ucret) > 1000
            GROUP BY bolumNo
```

8.4. HAVING DEYİMİ (DEVAM...)

Bu SQL cümlesi istenildiği gibi çalışmayacaktır. Çünkü **grup koşulları WHERE ifadesi ile birlikte çalışmazlar**. Bunun doğru kullanımı şöyle olacaktır.

```
SELECT bolumNo, AVG(ucret)
    FROM personel
        GROUP BY bolumNo
            HAVING AVG(ucret) > 1000
```

8.5. TABLOYA KAYIT EKLEME

Bir **tabloya kayıt eklemek için** **INSERT INTO deyimi** kullanılmaktadır. Bazı durumlarda SQL ifadeleri kullanarak bir tabloya kayıt eklemek durumunda kalabilirsiniz. Bu gibi durumlarda bu deyimi kullanmak yerinde olacaktır. Bu deyimin genel kullanım biçimi şöyledir.

INSERT INTO tablo_adı (sütunlar)

VALUES (değerler)

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

Burada, **tablo_adı**; verilerin kaydedileceği tablonun adıdır.

Sütunlar, tablo içinde verilerin kaydedileceği alanlardır.

Değerler ise tabloya kaydedilecek olan verileri göstermektedir.

Bu SQL deyimi her çalıştırıldığında **tabloya yeni bir kayıt** eklenir.

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

Örnek için yine bahçe işleri veritabanında **"kategoriler"** tablosuna çeşitli kayıtlar ekleyelim. Bunun için SQL ifademiz şöyle olacaktır:

```
INSERT INTO kategoriler (kategoriID, kategoriAdi)
    VALUES (5, 'Bahçe Çiçekleri')
```

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

```
INSERT INTO kategoriler (kategorilD, kategoriAdi)
        VALUES (5, 'Bahçe Çiçekleri')
```

Sorgusunda eğer tablodaki bütün alanlara kayıt yapılacaksa sütun adlarını kullanmadan da kayıt ekleme işlemi yapılabilmektedir.

```
INSERT INTO kategoriler
        VALUES (5, 'Bahçe Çiçekleri')
```

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

INSERT INTO deyimi ile çeşitli fonksiyonları da kullanmak mümkündür. Bir tarih bilgisi eklenmek istenildiğinde bunu doğrudan elle değil de bir fonksiyon aracılığı ile gerçekleştirebilirsiniz.

Örneğin, "siparisler" tablosunu göz önüne alalım. Bir sipariş verildiğinde bu siparişin verilme tarihini sistemden alarak tabloya kaydedelim.

```
INSERT INTO siparisler (siparisID, musterilD, calisanID,
                        siparisTarihi)
VALUES (16, 7, 4, GETDATE())
```

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

Dikkat edilecek olursa sipariş tarihi bilgisi eklenmek istenildiğinde bu **tarih bilgisi GETDATE () fonksiyonu** kullanılarak sistemden otomatik olarak alınmaktadır.

Alınan bu sistem bilgisi, siparisTarihi niteliğinin giriş değeri olmaktadır. Sorgu çalıştırıldığında ilgili kaydın eklendiği görülür.

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

INSERT INTO deyimi ile bir tabloya başka bir tablodan belirlenen bir satır bilgi eklenebilmektedir. Yani INSERT INTO deyimi ile birlikte **alt sorgu**larda kullanılabilmektedir.

Alt sorgulama işlemleri için SELECT ifadesi kullanılmaktadır. Böyle bir alt sorgulama kullanılacağı zaman **VALUES sözcüğü kullanılmaz.**

8.5. TABLOYA KAYIT EKLEME (DEVAM...)

Soru: Müşteriler tablosunda ilçesi 'Çankaya' olan müşterilerin musterilD, adi ve soyadi bilgilerini «secilenMüsteri» tablosuna kaydeden SQL ifadesi nedir?

Soruya dikkat edilirse soru iki bölümden oluşmaktadır. **İlçesi Çankaya olanlar** bulunacak ve ikinci olarak **«secilenMusteri» tablosuna kaydedilecek.** Bu işlemler için SQL ifademiz şöyle olacaktır:

```
INSERT INTO secilenMusteri (musterilD, adi, soyadi)
    SELECT musterilD, adi, soyadi
        FROM musteriler
            WHERE ilce ='Çankaya'
```

VALUES

8.6. KAYIT GÜNCELLEME

Bir tabloya eklenen bilgiler gerektiğinde değiştirilebilmelidir. Bu amaçla **UPDATE komutu** kullanılmaktadır. UPDATE komutunun genel kullanıl biçimi şöyledir.

```
UPDATE tablo_adı
    SET sütun1=değer1, sütun2=değer2, ...
        [WHERE koşul]
```

Bu tanımda kullanılan WHERE deyimi değişikliğin hangi kayıt üzerinde yapılacağının bulunmasına yardımcı olmaktadır. **Bu tanım yapılmadığında** söz konusu **tablonun tüm satırlarında değişiklik**ler meydana gelecektir.

8.6. KAYIT GÜNCELLEME (DEVAM...)

Bahçe işleri veritabanında müşteriler tablosunda isim bilgisi "Ali Arslan" olan kişinin Ankara'dan İstanbul Üsküdar'a iş yerini taşıdığını düşünerek gerekli değişiklikleri yapmak için aşağıdaki SQL sorgusunu yazmak gerekir.

```
UPDATE musteriler
    SET il='İstanbul', ilce = 'Üsküdar'
        WHERE (adi='Ali' AND soyadi='Arslan')
```

** **Dikkat**! Birden fazla Ali Arslan olabilir. Olması durumunda hepsinin il içe bilgisi güncellenir.

8.6. KAYIT GÜNCELLEME (DEVAM...)

UPDATE komutu ile matematiksel işlemler de gerçekleştirilebilmektedir. Tüm personelin ücretine %20 zam yapılacağını düşünerek tablo üzerinde değişiklikleri gerçekleştiren SQL ifadesi aşağıdaki gibi olacaktır.

UPDATE personel SET ucret = ucret*1.20

Bu komut çalıştırıldığında bütün kayıtlar bu değişiklikten etkilenecektir. Çünkü herhangi bir koşula bağlı bir işlem gerçekleştirilmemiştir.

8.6. KAYIT GÜNCELLEME (DEVAM...)

UPDATE işlemi tıpkı INSERT INTO deyiminde olduğu gibi **alt sorgularla da çalışabilmektedir**. Yani bilgi güncelleştirme işlemi bir tablodan okunan bilgilerin güncellenerek başka bir tabloya eklenmesi ile de gerçekleşebilmektedir.

Yine burada alt sorgular yazılırken SELECT deyiminden faydalanılmaktadır.

8.6. KAYIT GÜNCELLEME (DEVAM...)

UPDATE komutunda alt sorgulara şöyle bir örnek verebiliriz. BolumID'si 200 olan personel ile aynı göreve sahip tüm personelin bolumID'sini 100 olarak değiştirmek istersek şöyle bir SQL ifadesi yazabiliriz.

```
UPDATE personel SET bolumID = 100
    WHERE gorevi = (SELECT gorevi
                    FROM personel
                    WHERE bolumID=200)
```

8.6. KAYIT GÜNCELLEME (DEVAM...)

```
UPDATE personel SET bolumID = 100
    WHERE gorevi = (SELECT gorevi
                    FROM personel
                    WHERE bolumID=200)
```

Bu sorguda öncelikle alt sorguda bolumID'si 200 olan personelin görevinin ne olduğu tespit edilmektedir. Daha sonra tespit edilen bu görevle aynı görevi icra eden personel kayıtlarında bolumID'leri 100 olarak değiştirilmektedir.

8.7. KAYIT SİLME

Bir tabloda bulunan bir yada daha fazla satır silinebilmektedir. Silme işlemi için DELETE deyimi kullanılmaktadır. Bu deyimin genel yapısı şu şekildedir.

DELETE FROM tablo_adı

WHERE koşul

8.7. KAYIT SİLME (DEVAM...)

Bir tabloda bütün kayıtlar silinebileceği gibi belli bir şartı sağlayan herhangi bir kayıtta silinebilmektedir. Örneğin daha önce oluşturduğumuz «secilenmüsteri» tablosunun tüm kayıtlarını silmek için şöyle bir SQL ifadesi yazılacaktır.

DELETE FROM secilenMüsteri

Herhangi bir koşul olmadığı için bütün kayıtlar silinecektir.

8.7. KAYIT SİLME (DEVAM...)

Şimdi de belli bir şarta sahip kayıtları silmek için bir örnek verelim. Müşteriler tablosunda "Ali Arslan" isimli müşteri ile artık çalışmadığımızı düşünerek bu müşteriyi tamamen silelim. SQL ifademiz şöyle olacaktır.

```
DELETE FROM musteriler

    WHERE (adi='Ali' AND soyadi='Arslan')
```

8.7. KAYIT SİLME (DEVAM...)

Aynı şekilde bir tabloda istenilen şartı sağlayan birden fazla kayıtta silinebilmektedir. Örneğin Müşteriler tablosunda Ankara 'da oturan bütün müşterileri silmek istersek SQL ifademiz aşağıdaki gibi olacaktır.

```
DELETE FROM musteriler
    WHERE il ='Ankara'
```

8.7. KAYIT SİLME (DEVAM...)

DELETE komutunda da tıpkı INSERT INTO ve UPDATE komutlarında olduğu gibi alt sorgular kullanılarak da kayıtlar silinebilmektedir. Bu noktada yine SELECT ifadesinden yararlanılmaktadır.

Örneğin; personel tablosunda İMALAT bölümüne ait bütün kayıtları silmek istiyoruz. Bu işlemi gerçekleştiren SQL ifadesi şöyle olacaktır:

```
DELETE FROM personel
    WHERE bolumID = (SELECT bolumID
                     FROM bolum
                     WHERE bolumAdi = 'İMALAT')
```

TEŞEKKÜRLER…

SORULARINIZ ?