VERİ TABANI YÖNETİM SİSTEMLERİ TERİMLER

Mustafa Acar - 01.01.2019

- Veri tabanı nedir?

Veri tabanı, herhangi bir konuda birbirleri ile ilişkili verilerin sistematik olarak oluşturduğu yapılardır.

- Veri tabanı yönetim sistemi nedir?

Veri tabanında tabloları oluşturmayı sağlayan, bu tablolara veri girişi yapan, tablolardaki mevcut olan veride güncelleştirme- değişiklik yapmak ve verileri silmek gibi işlemlerin yapılmasına imkân veren ve veri tabanının yönetimini sağlayan yazılım sistemlerine denir.

Temel kontrol yapılarını kapsayan ileri SQL dili MS-SQL SERVER VTYS'de ne olarak adlandırılır?

- T-SQL

Oracle ve IBM DB2'de temel kontrol yapılarını kapsayan ileri SQL dili hangisidir?

- PL/SQL

- Primary ve Foreign Key nedir?

Primary Key (Birincil Anahtar): Veri tekrarını önleyen anahtar

Foreign Key (Yabancı Anahtar): Veri tutarsızlığına engel olan anahtar.

- GPU veri tabanları - Graphics Processing Unit

GPU veri tabanları, Makine Öğrenme gibi veri bilimi ile ilgili işlemler için Grafik İşleme Ünitesi kartlarının gelişmesi ortaya çıkmıştır. GPU'lar, sorgu paralel hale getirildiğinde en iyi sonucu verir. Bunun nedeni, GPU'ların binlerce çekirdek içermesidir. Her bir çekirdek, verilerin küçük bir alt kümesi üzerinde çalıştırılırsa, bir GPU genellikle CPU tabanlı sorgu performansını aşan hızlarda hesap yapabilir.

- En çok kullanılan veri tabanı yönetim model yaklaşımları;

Hiyerarşik, Ağ, İlişkisel ve Nesneye dayalı(Object Oriented) modellerdir.

Günümüzde en fazla kullanılan model, ilişkisel veri tabanı modelidir.

- dbo: DataBase Owner

- CHAR VE VARCHAR

CHAR sabit uzunluktadır, her veri girişinde aynı miktarda depolama alanı kullanır.

VARCHAR ise değişken uzunluktadır, sadece verinin uzunluğu kadar bir alan kullanır.

- Bire-Çok (1:M) İlişki

Bir yönetici birçok çalışanı yönetmektedir, ancak her bir çalışanın sadece bir yöneticisi vardır, bu nedenle bire çok (1: m) ilişki vardır.

VERİ TABANI YÖNETİM SİSTEMLERİ KOMUTLAR

01.01.2019

- `CREATE DATABASE` komutu veri tabanı oluşturur.

ÖRN: `CREATE DATABASE MuhendislikFakultesi`

- `USE` komutu çalışacağımız veri tabanını seçer.

ÖRN: `USE MuhendislikFakultesi`

- `CREATE TABLE` komutu veri tabanına yeni bir tablo ekler ve alanları oluşturur.

ÖRN:

```
CREATE TABLE Fakulte
(
FakulteID INT NOT NULL PRIMARY KEY,
FakulteAdi VARCHAR(50) NOT NULL
)
```

BİLGİ: `NOT NULL` bu alana kayıt yapılırken boş geçilemeyeceği anlamına geliyor.

- `ALTER TABLE ADD` mevcut bir tabloya yeni bir alan ekler.

ÖRN:

```
ALTER TABLE Fakulte
ADD BolumSayisi INT NOT NULL
```

- `ALTER TABLE ALTER COLUMN` mevcut bir alan üzerinde güncelleme yapar.

ÖRN:

```
ALTER TABLE TabloAdi
ALTER COLUMN AlanAdi veri_tipi NOT NULL
```

- `INSERT INTO` komutu ile bir tabloya veri eklenir.

ÖRN:

```
INSERT INTO Fakulte (FakulteID, FakulteAdi)
VALUES (1, 'Bilgisayar ve Bilişim Bilimleri')
```

	FakulteID	FakulteAdi
	1	Bilgisayar ve Bilişim Bilimleri
▶*	*NULL*	*NULL*

`INSERT INTO` komutu ile bir tabloya aynı anda birden fazla veri ekleyebiliriz.

```
INSERT INTO Fakulte (FakulteID, FakulteAdi)
VALUES (2, 'Elektrik-Elektronik'), (3, 'Makine ve Metalurji'), (4, 'Endüstri')
```

	FakulteID	FakulteAdi
	1	Bilgisayar ve Bilişim Bilimleri
	2	Elektrik-Elektronik
	3	Makine ve Metalurji
	4	Endüstri
▶*	*NULL*	*NULL*

- UPDATE komutu ile mevcut bir kaydı güncelleriz.

ÖRN:

```
UPDATE Fakulte
SET FakulteAdi = 'Elektrik-Elektronik Mühendisliği'
WHERE FakulteID = 2
```

	FakulteID	FakulteAdi
	1	Bilgisayar ve Bilişim Bilimleri
	2	Elektrik-Elektronik Mühendisliği
	3	Makine ve Metalurji
	4	Endüstri
▶*	*NULL*	*NULL*

- DELETE komutu ile mevcut bir kaydı sileriz.

ÖRN:

```
DELETE
FROM Fakulte
WHERE FakulteID = 4
```

	FakulteID	FakulteAdi
	1	Bilgisayar ve Bilişim Bilimleri
	2	Elektrik-Elektronik Mühendisliği
	3	Makine ve Metalurji
▶*	*NULL*	*NULL*

- DROP TABLE mevcut bir tabloyu sileriz.

ÖRN: `DROP TABLE Fakulte`

- SELECT

SELECT ifadesi veri tabanından verileri okumak için kullanılır. Veri tabanındaki tablo üzerindeki bütün alanlardan veri çekilebileceği gibi belirteceğimiz bir kaç alandan da veri çekebiliriz. En azından bir alan belirtmek gerekmektedir.

```
SELECT *
FROM Fakulte
```

! Eğer bütün alanlardan veri çekeceksek, Select ifadesinden sonra * işareti konulur. Böylece bütün alanlardan veri çekileceği belirtilmiş olunur.

```
SELECT FakulteID, FakulteAdi
FROM Fakulte
```

- SELECT DISTINCT

SELECT DISTINCT ifadesi tablodaki belirtilen alanda bulunan kayıtlardan birer örnek alır. Yani tekrar eden kayıtlardan bir tane alır ve bunun yanına da tekrar etmeyen kayıtları koyarak bir veri kümesi oluşturur. Kısacası veri tekrarını engeller.

```
SELECT DISTINCT FakulteAdi
FROM Fakulte
```

- WHERE

WHERE ifadesi tablodaki alanlarda okuma, güncelleme, silme gibi işlemleri yaparken belli kriterlere sahip kayıtlar üzerinde işlem yapmamızı sağlar. WHERE ifadesi belirtilmezse uygulanan komut bütün kayıtlar üzerinde geçerli olur. Mesela bir kaydı silmek istediğimiz zaman WHERE ifadesini kullanmazsak tablodaki bütün kayıtları silecektir.

```
SELECT *
FROM Fakulte
WHERE FakulteID = 1
```

```
SELECT *
FROM Fakulte
WHERE FakulteAdi = 'Bilgisayar ve Bilişim Bilimleri'
```

Operatör	Açıklama
=	Eşit
<>	Eşit Değil. Not: Bazı versiyonlarda "!=" kullanılabilir.
>	Büyüktür
<	Küçüktür.
>=	Büyük Eşit
<=	Küçük Eşit
BETWEEN	Arasında
LIKE	Örüntü arama
IN	Bir sütun için birden çok olası değerleri belirtmek için

- AND – OR

AND ve OR ifadeleri birden fazla alanda işlem yapılacaksa kullanılan operatörlerdir. AND operatörü birinci durumla beraber ikinci durumunda olduğu zaman kullanılır. OR operatörü ise birinci durum veya ikinci durumun gerçekleşmesi durumunda kullanılır.

```
SELECT *
FROM Fakulte
WHERE FakulteAdi = 'Bilgisayar ve Bilişim Bilimleri'
AND Yerleske = 'Serdivan'
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	5	Bilgisayar ve Bilişim Bilimleri	Serdivan

```
SELECT *
FROM Fakulte
WHERE FakulteAdi = 'Bilgisayar ve Bilişim Bilimleri'
OR Yerleske = 'Adapazarı'
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	2	Elektrik-Elektronik	Adapazarı
3	5	Bilgisayar ve Bilişim Bilimleri	Serdivan

- ORDER BY

ORDER BY ifadesi kayıtları belirtilen alanda büyükten küçüğe veya küçükten büyüğe göre sıralar. ASC (ascending) parametresi ile küçükten büyüğe, DESC (descending) parametresi ile büyükten küçüğe göre sıralar. Burada sadece sayısal alanlar değil metinsel alanlarda alfabetik olarak sıralanabilir.

ASC küçükten büyüğe (Artan sırada)
DESC büyükten küçüğe (Azalan sırada)

```
SELECT *
FROM Fakulte
ORDER BY FakulteID ASC
```

! ORDER BY FakulteID yazdığımızda otomatik olarak ASC tipinde sıralar.
! ORDER BY NEWID() yazdığımızda her defasında random olarak sıralar.

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	2	Elektrik-Elektronik	Adapazarı
3	3	Makine ve Metalurji	Kemalpaşa
4	4	Endüstri	Camili
5	5	Bilgisayar ve Bilişim Bilimleri	Serdivan

```
SELECT *
FROM Fakulte
ORDER BY FakulteID DESC
```

	FakulteID	FakulteAdi	Yerleske
1	5	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	4	Endüstri	Camili
3	3	Makine ve Metalurji	Kemalpaşa
4	2	Elektrik-Elektronik	Adapazarı
5	1	Bilgisayar ve Bilişim Bilimleri	Serdivan

```
SELECT *
FROM Fakulte
ORDER BY FakulteAdi DESC, FakulteID DESC
```

! Bu kod, tablodaki FakulteAdi alanına göre kayıtları büyükten küçüğe (DESC) sıralar ve eğer aynı içeriğe sahip kayıt tespit ederse FakulteID alanını dikkate alır. FakulteID alanı DESC olarak sıralanır. İlgili tabloda iki tane Bilgisayar ve Bilişim Bilimleri Fakültesi vardır. Bu durumda Bilgisayar ve Bilişim Bilimleri Fakültesi ID alanları kendi içerisinde DESC olarak sıralanacaktır.

	FakulteID	FakulteAdi	Yerleske
1	3	Makine ve Metalurji	Kemalpaşa
2	4	Endüstri	Camili
3	2	Elektrik-Elektronik	Adapazarı
4	5	Bilgisayar ve Bilişim Bilimleri	Serdivan
5	1	Bilgisayar ve Bilişim Bilimleri	Serdivan

- SELECT TOP – PERCENT

SELECT ifadesi veri tabanından verileri okumak için kullanılır. Select ifadesi tek başına kullanıldığında veri tabanındaki tablo üzerinde bulunan bütün verileri seçer. Ancak binlerce hatta on binlerce kaydın olduğu bir tablodan bütün verileri çekmek veri tabanını zorlayacak ve uygulamanın kitlenmesine neden olacaktır. Bunun yerine SELECT TOP ile belirtilen kadar kayıt seçilir. Select Top ifadesi ile kayıt adedi veya yüzdesi belirtildikten sonra alan adları mutlaka yazılmalıdır. Yazılan alan adlarındaki kayıtlar ekrana gelir. * işareti konulursa bütün alanlar seçilir.

```
SELECT TOP 3 *
FROM Fakulte
```

Kayıtlardan ilk 3 tanesi ve * ile tüm alanlar seçilmiştir.

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	2	Elektrik-Elektronik	Adapazan
3	3	Makine ve Metalurji	Kemalpaşa

```
SELECT TOP 60 PERCENT *
FROM Fakulte
```

PERCENT ifadesi ile belirttiğimiz sayı yüzde olarak tanımlanır. Örneğimizde tüm kayıtların %60'lık kısmı gösterilecek.

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	2	Elektrik-Elektronik	Adapazan
3	3	Makine ve Metalurji	Kemalpaşa

- LIKE

Belirtilen bir değeri aramak için kullanılır.

Örnek olarak Fakulte tablomuzda FakulteAdi alanında baş harfi B ile başlayanları listeleyelim.

```
SELECT *
FROM Fakulte
WHERE FakulteAdi LIKE 'B%'
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	5	Bilgisayar ve Bilişim Bilimleri	Serdivan

Örnek olarak Fakulte tablomuzda FakulteAdi alanında son harfi i ile bitenleri listeleyelim.

```
SELECT *
FROM Fakulte
WHERE FakulteAdi LIKE '%i'
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	3	Makine ve Metalurji	Kemalpaşa
3	4	Endüstri	Camili
4	5	Bilgisayar ve Bilişim Bilimleri	Serdivan
5	7	Teknoloji	Arifiye

FakulteAdi alanındaki kayıtların içerisinde Bilişim kelimesi geçenleri listeler.

```
SELECT *
FROM Fakulte
WHERE FakulteAdi LIKE '%Bilişim%'
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	5	Bilgisayar ve Bilişim Bilimleri	Serdivan

FakulteAdi alanındaki kayıtların içerisinde Bilişim kelimesi geçmeyenleri listeler.

```
SELECT *
FROM Fakulte
WHERE FakulteAdi NOT LIKE '%Bilişim%'
```

	FakulteID	FakulteAdi	Yerleske
1	2	Elektrik-Elektronik	Adapazarı
2	3	Makine ve Metalurji	Kemalpaşa
3	4	Endüstri	Camili
4	6	İnşaat	Arifiye
5	7	Teknoloji	Arifiye

İşaret	Açıklaması
%	Birden fazla harf ya da rakamın yerini tutar
_	Bir tek harf veya rakamın yerini tutar
[harfler]	Herhangi bir harf yerine gelebilecek harfleri belirtir
[^harfler] ya da [!harfler]	Herhangi bir harf yerine gelemeyecek harfleri belirtir
[harf-harf]	Belirtilen aralıktaki harfleri belirtir

Birinci harfi A, dördüncü harfi f, yedinci harfi e olanları listeler.

```
SELECT *
FROM Fakulte
WHERE Yerleske LIKE 'A__f__e'
```

	FakulteID	FakulteAdi	Yerleske
1	6	İnşaat	Arifiye
2	7	Teknoloji	Arifiye

İlk harfi s veya c olanları listeler.

```
SELECT *
FROM Fakulte
WHERE Yerleske LIKE '[sc]%'
```

Son harfi e veya i olanları listeler.

```
SELECT *
FROM Fakulte
WHERE Yerleske LIKE '%[ei]'
```

Baş harfi s veya c olmayanları listeler. ^ işareti yerine ! işareti de kullanılabilir.

```
SELECT *
FROM Fakulte
WHERE Yerleske LIKE '[^sc]%'
```

İlk harfi A ile c harfleri arasında ki herhangi bir harf ile başlayan kayıtları listeler.

```
SELECT *
FROM Fakulte
WHERE Yerleske LIKE '[a-c]%'
```

- IN

IN operatörü belirtilen tek bir alanda birden fazla değeri aramak için kullanılır.

```
SELECT *
FROM Fakulte
WHERE FakulteID IN (1, 3, 5)
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	3	Makine ve Metalurji	Kemalpaşa
3	5	Bilgisayar ve Bilişim Bilimleri	Serdivan

```
SELECT *
FROM Fakulte
WHERE Yerleske IN ('Arifiye', 'Serdivan')
```

	FakulteID	FakulteAdi	Yerleske
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	5	Bilgisayar ve Bilişim Bilimleri	Serdivan
3	6	İnşaat	Arifiye
4	7	Teknoloji	Arifiye

- BETWEEN

Between operatörü ile bir alanda belirtilen aralıktaki değerleri aramak için kullanılır.

```
SELECT *
FROM Fakulte
WHERE FakulteID BETWEEN 1 AND 3
```

```
SELECT *
FROM Fakulte
WHERE FakulteAdi BETWEEN 'A' AND 'E'
```

- ALIASES (AS)

AS ifadesi ile uzun ve kullanımı zor olan tablo veya alan adlarına geçici olarak kısa isimler vererek bunları kodlamalarımızda kullanabiliriz. Böylece mevcut tablo yapımız bozmadan anlık olarak belirlediğimiz isimleri kullanabiliriz. Verilecek olan geçici ad eğer boşluk içeriyorsa köşeli parantez içinde yazılır.

```
SELECT FakulteID AS [Fakülte Kodu], FakulteAdi AS FakülteAdı, Yerleske AS Yerleşke
FROM Fakulte
```

	Fakülte Kodu	FakülteAdı	Yerleşke
1	1	Bilgisayar ve Bilişim Bilimleri	Serdivan
2	2	Elektrik-Elektronik	Adapazarı
3	3	Makine ve Metalurji	Kemalpaşa
4	4	Endüstri	Camili
5	5	Bilgisayar ve Bilişim Bilimleri	Serdivan
6	6	İnşaat	Arifiye
7	7	Teknoloji	Arifiye

! Her bir alanda tek tek AS yazmamıza gerek yoktur. İlk alanda kullanmamız yeterlidir.

```
SELECT FakulteID AS [Fakülte Kodu], FakulteAdi FakülteAdı, Yerleske Yerleşke
FROM Fakulte
```

- IS NULL ve IS NOT NULL

Tablomuzdaki veri girişleri esnasında bazı alanlara veriler girilmemiş olabilir. Bu durumda veri tabanından çekilen raporlarda bazı hatalar oluşabileceği gibi ortaya çıkan sonuçlar da yanıltıcı olabilir. Oluşturacağımız sorgu ile tablomuzdaki boş kayıtları göz ardı ederek daha sağlıklı sonuçlar elde edebiliriz. Bunun için WHERE ifadesinden sonra IS NULL operatörü ile boş kayıtlar seçilebileceği gibi IS NOT NULL operatörü ile de boş olmayan kayıtlar seçilebilir.

Not: NULL değer ile 0 değeri birbirinin aynısı değildir. İlgili alana sıfır değeri girilmiş olsa bile o alan NULL yani boş değildir.

```
SELECT *
FROM Fakulte
WHERE FakulteAdi IS NULL          //FakulteAdi alanındaki boş kayıtları seçer.
```

```
SELECT *
FROM Fakulte
WHERE FakulteAdi IS NOT NULL      //FakulteAdi alanındaki boş olmayan kayıtları seçer.
```

- GROUP BY

Bir fonksiyonu kullanırken bazı durumlarda GROUP BY fonksiyonu ile belli alanlara göre gruplamak gerekebilir.

```
SELECT FakulteAdi, COUNT (FakulteID) AS [Fakulte Sayısı]
FROM Fakulte
GROUP BY FakulteAdi
```

```
SELECT Color AS Renk, COUNT (*)  [Kayıt Sayısı], SUM (ListPrice) [Toplam Fiyat], AVG
(ListPrice) Ortalama
FROM Production.Product
GROUP BY Color
```

Renklere göre kümeler, her renge ait kayıt sayısı, toplam fiyat ve fiyat ortalamasını hesaplar.

	Renk	Kayıt Sayısı	Toplam Fiyat	Ortalama
1	NULL	248	4182,32	16,8641
2	Black	93	67436,26	725,121
3	Blue	26	24015,66	923,6792
4	Grey	1	125,00	125,00
5	Multi	8	478,92	59,865
6	Red	38	53274,10	1401,95
7	Silver	43	36563,13	850,3053
8	Silver/Black	7	448,13	64,0185
9	White	4	36,98	9,245
10	Yellow	36	34527,29	959,0913

- COUNT

COUNT() fonksiyonu belirtilen alandaki veya tablodaki toplam kayıt sayısını verir. Burada dikkat edilmesi gereken alan üzerindeki kayıt sayıları alınırken boş verilerin dikkate alınmamasıdır.

```
SELECT COUNT (*)  [Kayıt Sayısı]
FROM Fakulte
```

```
SELECT COUNT (DISTINCT FakulteAdi)  [Kayıt Sayısı]
FROM Fakulte
```

- AVG

AVG() fonksiyonu ile belirtilen alandaki değerlerin ortalaması elde edilir.

```
SELECT AVG(alan_adi) FROM tablo_adi
```

- MAX

MAX() fonksiyonu belirtilen alandaki en büyük değeri verir.

```
SELECT MAX(alan_adi) FROM tablo_adi
```

- MİN

MAX() fonksiyonu belirtilen alandaki en küçük değeri verir.

```
SELECT MIN(alan_adi) FROM tablo_adi
```

- SUM

SUM() fonksiyonu ile belirtilen alandaki değerlerin toplamı elde edilir.

```
SELECT SUM(alan_adi) FROM tablo_adi
```

- Upper()

Upper() fonksiyonu ile metin tipindeki alanlardaki verileri büyük karaktere çevirmeye yarar.

```
SELECT UPPER(alan_adi) FROM tablo_adi
```

- Lower()

Lower() fonksiyonu ile metin tipindeki alanlardaki verileri ufak karaktere çevirmeye yarar.

```
SELECT LOWER(alan_adi) FROM tablo_adi
```

- LEN()

LEN() fonksiyonu metin tipindeki alanlarda verilerin boşluklar dâhil karakter sayısını öğrenmemize yarar.

```
SELECT LEN (FakulteAdi) AS Uzunluk FROM Fakulte
```

```
SELECT *
FROM Fakulte
WHERE LEN (FakulteAdi) > 9
```

- ROUND

ROUND() fonksiyonu belirtilen ondalık sayı için sayısal bir alanda yuvarlamak için kullanılır.

```
SELECT ROUND (Fiyat,0) FROM Urunler
```

! (Fiyat, 1) olsaydı virgülden sonraki rakamların ilki dikkate alınmayacaktır. İkinci rakamlar ise 5'ten büyük olanlar bir üst rakama, ufak olanlar ise bir alt rakama yuvarlandı.

- NOW

NOW() fonksiyonu o anki geçerli olan sistem tarihi ve saatini verir.

- REPLACE

REPLACE kelimesinin Türkçe' de "yerine koymak, yerini almak, yenisi ile değiştirmek" anlamlarına gelmektedir. Yani tablomuzda bulunan bir metinsel alandaki verinin bir parçasını veya tamamını ekrana farklı biçimde yazdırabiliriz.

```
SELECT FakulteAdi, REPLACE(FakulteID, 1, 'M1') FROM Fakulte
```

	FakulteAdi	(No column name)
1	Bilgisayar ve Bilişim Bilimleri	M1

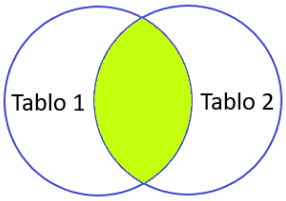
- INNER JOIN

İki adet tablomuzdaki kayıtları belli bir kritere göre birleştirmek için INNER JOIN komutu kullanılır.

```
SELECT alan_ad(lari)
FROM tablo1 INNER JOIN tablo2
ON tablo1.alan_adi=tablo2.alan_adi
```

INNER JOIN yerine sadece JOIN kullanılabilir.

Burada görüleceği üzere FROM ifadesi ile birinci tablomuzu ve ardından JOIN (veya INNER JOIN) ile ikinci tablomuzu belirtmiş oluyoruz. ON ile hangi alanların eşitleneceği gösterilmektedir. Birinci tabloda olan bir kayıt ikinci tabloda olmayabilir veya ikinci tabloda olan ir kayıt birinci tabloda olmayabilir. Bu durumda sadece belirtilen alanlarda eşit olan kayıtlar seçilir.



Birinci tablomuz Musteriler olsun

id	Adi_soyadi
1	Salih ESKİOĞLU
2	Ayhan ÇETİNKAYA
3	Serkan ÖZGÜREL
4	İlhan ÖZLÜ

İkinci tablomuz olan Satışlar ise aşağıdaki gibi olsun.

id	Satilan_mal	Satis_fiyati
1	Buzdolabi	1200
1	LCD TV	1800
2	LCD TV	1750
1	Çamaşır Makinesi	950

```
SELECT *
FROM Musteriler JOIN Satislar
ON Musteriler.id = Satislar.id
```

Çıktısı:

id	Adi_soyadi	Satilan_mal	Satis_fiyati
1	Salih ESKİOĞLU	Buzdolabı	1200
1	Salih ESKİOĞLU	LCD TV	1800
2	Ayhan ÇETİNKAYA	LCD TV	1750
1	Salih ESKİOĞLU	Çamaşır Makinesi	950

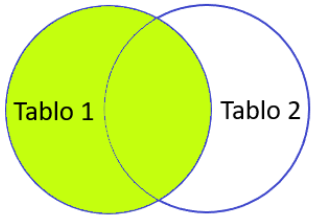
- LEFT JOİN

LEFT JOIN ile iki adet tablomuzdaki kayıtları belli bir kritere göre birleştirebiliriz. Burada asıl olan birinci tablodaki kayıtlardır. İkinci tablodan sadece birinci tabloda olan kayıtlar alınır. İkinci tabloda olup birinci tabloda olmayan alanların değeri boş (NULL) olarak gelecektir.

```
SELECT alan_ad(lari)
FROM tablo1 LEFT JOIN tablo2 ON tablo1.alan_adi=tablo2.alan_adi
```

LEFT JOIN aynı zamanda LEFT OUTER JOIN olarak da kullanılabilir.

Burada görüleceği üzere FROM ifadesi ile birinci tablomuzu ve ardından LEFT JOIN (veya LEFT OUTER JOIN) ile ikinci tablomuzu belirtmiş oluyoruz. ON ile hangi alanların eşitleneceği gösterilmektedir. Birinci tabloda ki bütün kayıtlar seçilir. Buna karşılık eşitlenen alanlara bakılarak ikinci tablodan sadece birinci tabloda olan kayıtlar seçilir.



Birinci tablomuz Musteriler olsun

id	Adi_soyadi
1	Salih ESKİOĞLU
2	Ayhan ÇETİNKAYA
3	Serkan ÖZGÜREL
4	İlhan ÖZLÜ

İkinci tablomuz olan Satışlar ise aşağıdaki gibi olsun.

id	Satilan_mal	Satis_fiyati
1	Buzdolabı	1200
2	LCD TV	1800
5	LCD TV	1750
8	Çamaşır Makinesi	950

```
SELECT *
FROM Musteriler LEFT JOIN Satislar
ON Musteriler.id = Satislar.id
```

id	Adi_soyadi	Satilan_mal	Satis_fiyati
1	Salih ESKİOĞLU	Buzdolabı	1200
2	Ayhan ÇETİNKAYA	LCD TV	1800
3	Serkan ÖZGÜREL		
4	İlhan ÖZLÜ		

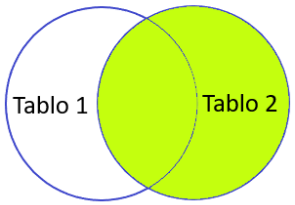
- RİGHT JOİN

RIGHT JOIN ile iki adet tablomuzdaki kayıtları belli bir kritere göre birleştirebiliriz. Burada asıl olan ikinci tablodaki kayıtlardır. Birinci tablodan sadece ikinci tabloda olan kayıtlar alınır. Birinci tabloda olup ikinci tabloda olmayan alanların değeri boş (NULL) olarak gelecektir.

```
SELECT alan_ad(lari)
FROM tablo1 RIGHT JOIN tablo2
ON tablo1.alan_adi = tablo2.alan_adi
```

RIGHT JOIN yerine OUTER JOIN de kullanılabilir.

Burada görüleceği üzere FROM ifadesi ile birinci tablomuzu ve ardından RIGHT JOIN (veya OUTER JOIN) ile ikinci tablomuzu yani asıl kayıtların baz alınacağı tablomuzu belirtmiş oluyoruz. ON ile hangi alanların eşitleneceği gösterilmektedir. İkinci tabloda ki bütün kayıtlar seçilir. Buna karşılık eşitlenen alanlara bakılarak birinci tablodan sadece ikinci tabloda olan kayıtlar seçilir.



Birinci tablomuz Personel olsun.

id	Adi_soyadi	Departman_id
1	Salih ESKİOĞLU	2
2	Ayhan ÇETİNKAYA	
3	Serkan ÖZGÜREL	4
4	İlhan ÖZLÜ	1

İkinci tablomuz ise Departmanlar tablosu olsun.

id	Departman_ad
1	Muhasebe
2	Bilgi İşlem
3	Satış
4	Pazarlama

```
SELECT *
FROM Personel RIGHT JOIN Departmanlar ON personel.departman_id = departmanlar.id
```

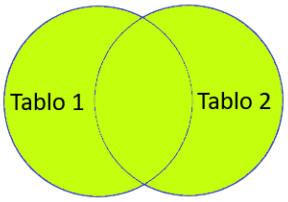
id	Adi_soyadi	Departman_id	id	Departman_ad
4	İlhan ÖZLÜ	1	1	Muhasebe
1	Salih ESKİOĞLU	2	2	Bilgi İşlem
			3	Satış
3	Serkan ÖZGÜREL	4	4	Pazarlama

- FULL JOIN

FULL JOIN ile iki adet tablomuzdaki kayıtları belli bir kritere göre birleştirebiliriz.

Burada her iki tablomuzdaki bütün kayıtlar seçilir.

FULL JOIN yerine FULL OUTER JOIN de kullanılabilir.



Örnek olarak aşağıdaki gibi Personel isimli tablomuz olsun.

id	Adi_soyadi
1	Salih ESKİOĞLU
2	Ayhan ÇETİNKAYA
3	Serkan ÖZGÜREL
4	İlhan ÖZLÜ

İkinci tablomuz ise Satışlar tablosu olsun.

satici_id	Urun	Satis_Fiyati
1	Buzdolabı	1200
1	Laptop	1750
3	LCD TV	1300
1	Ses Sistemi	750

```
SELECT *
FROM Personel FULL JOIN Satislar
ON personel.id = satislar.satici_id
```

id	Adi_soyadi	satici_id	Urun	Satis_Fiyati
1	Salih ESKİOĞLU	1	Buzdolabı	1200
2	Ayhan ÇETİNKAYA			
3	Serkan ÖZGÜREL	3	LCD TV	1300
4	İlhan ÖZLÜ			
1	Salih ESKİOĞLU	1	Laptop	1750
1	Salih ESKİOĞLU	1	Ses Sistemi	750

- HAVİNG

HAVING yapısı temelde WHERE ile aynı görevi yapmaktadır. GROUP BY ile kullanılır. WHERE ifadesi ile belirtilen kriter GROUP BY uygulanmadan önce geçerli olurken, HAVING ifadesi ile belirtilen kriter ise GROUP BY uygulandıktan sonra ortaya çıkan verileri filtrelemek için kullanılır.

id	Personel_adi	Sehir	Maas
1	Salih ESKİOĞLU	İstanbul	2100
2	Ayhan ÇETİNKAYA	Kocaeli	2500
3	Serkan ÖZGÜREL	Erzurum	1850
4	İlhan ÖZLÜ	İstanbul	2750

```
SELECT Sehir, SUM(Maas) AS Toplam_Maas FROM Personel
WHERE Maas > 2000
GROUP BY Sehir
```

Çıktı:

Sehir	Toplam_Maas
İstanbul	4850
Kocaeli	2500

```
SELECT Sehir, SUM(Maas) AS Toplam_Maas FROM Personel
WHERE Maas > 2000
GROUP BY Sehir
HAVING Sehir like 'İ%'
```

Çıktı:

Sehir	Toplam_Maas
İstanbul	4850

- UNION

UNION ile iki adet tablomuzdaki seçeceğimiz alanları birleştirerek tek bir tablo alanıymış gibi kullanabiliriz. UNION ile iki tablodaki alanlar birleştirilirken tekrarlayan kayıtlar bir defa alınır. Eğer tekrarlayan kayıtların alınması isteniyorsa UNION ALL kullanılmalıdır.

```
SELECT alan_ad(lari) FROM tablo1
UNION
SELECT alan_ad(lari) FROM tablo2
```

```
SELECT alan_ad(lari) FROM tablo1
UNION ALL
SELECT alan_ad(lari) FROM tablo2
```

- CASE WHEN

Basit CASE fonksiyonu genelde şu amaç için kullanılır. Mevcut bir veri satırında yer alan değerin daha sonra istediğimiz şekilde görüntülenmesini sağlayabiliriz. Kullanım olarak kalıbı:

CASE testifadesi

WHEN karşılaştırılacak ifade 1 THEN Bunun Geri Dönüş Değeri

WHEN karşılaştırılacak ifade 2 THEN Bunun Geri Dönüş Değeri

ELSE karşılaştırılacak son değer END AS kolonbaşlığı

```
SELECT ProductID, Name, ListPrice,
       CASE WHEN ListPrice < 100 THEN 'Ucuz'
       WHEN ListPrice >= 100 AND ListPrice < 500 THEN 'Orta'
       WHEN ListPrice >= 500 THEN 'Pahalı'
       END AS FiyatAciklama
FROM Production.Product
```

- PRIMARY KEY

PRIMARY KEY ile tablomuzdaki ilgili alanda benzersiz kayıtların tutulmasını istediğimiz durumlarda kullanılır.

```
CREATE TABLE Personel
(
id INT NOT NULL PRIMARY KEY,
adi_soyadi VARCHAR(20) ,
Sehir VARCHAR(20)
)
```

Birden fazla alanda kullanım biçimi;

```
CREATE TABLE Personel
(
id INT NOT NULL,
adi_soyadi VARCHAR (20) NOT NULL,
Sehir VARCHAR (20),
CONSTRAINT id_no PRIMARY KEY  (id,adi_soyadi)
)
```

```
ALTER TABLE Personel
ADD PRIMARY KEY (id)
```

Birden fazla alanda kullanım biçimi;

```
ALTER TABLE Personel
ADD CONSTRAINT  id_no PRIMARY KEY (id,adi_soyadi)
```

Burada dikkat edilecek nokta; ALTER ile sonradan bir alana PRIMARY KEY tanımlanırken ilgili alanda veya alanlarda NULL yani boş kayıt olmamalıdır.

PRIMARY KEY yapısını kaldırmak;

```
ALTER TABLE Personel
DROP CONSTRAINT id_no
```

- FOREIGN KEY

Temel olarak FOREIGN KEY yardımcı index oluşturmak için kullanılır. Bir tabloda "id" alanına PRIMARY KEY uygulayabiliriz. Ancak aynı tablodaki başka bir alan farklı bir tablodaki kayda bağlı çalışabilir. MS Access veri tabanlarında bu duruma "İlişkili Veri tabanı" deniliyor. İşte bu iki tablo arasında bir bağ kurmak gerektiği durumlarda FOREIGN KEY devreye giriyor. Böylece tablolar arası veri akışı daha hızlı olduğu gibi ileride artan kayıt sayısı sonucu veri bozulmalarının önüne geçilmiş olunur.

```
CREATE TABLE Satislar
(
id INT NOT NULL PRIMARY KEY,
Urun VARCHAR(20) ,
Satis_fiyati VARCHAR(20),
satan_id INT CONSTRAINT fk_satici FOREIGN KEY REFERENCES Personel(id)
)
```

! FOREIGN KEY tanımlaması yapılırken hangi tablodaki hangi alanla ilişkili olduğunu REFERENCES ifadesinden sonra yazmak gerekir.

Var olan Tabloya FOREIGN KEY Tanımlama;

```
ALTER TABLE Satislar
ADD CONSTRAIN fk_satici
FOREIGN KEY (satan_id)
REFERENCES Personel(id)
```

FOREIGN KEY Kaldırma;

```
ALTER TABLE Satislar
DROP CONSTRAINT fk_satici
```

- VIEW

View'ler select ifadesi ile tanımlanmış sanal tablolardır. Temel amacı tabloların içerisinden veri kümesi getirip ortaya çıkan sonucu sanal tabloymuş gibi yeniden sorgulayabilmemizi sağlamaktır. View'ler sadece tablolar içerisindeki verileri görüntülemeye yarar, bu nedenden ötürü view içerisinde verileri etkileyecek herhangi bir işlem yapılamaz.

İçerisinde sadece SQL Fonksiyonlarını, Join, Group By ve Where ifadelerini kullanabilirsiniz.

NELER KULLANILMAZ?

1. ORDER BY kullanamazsınız.
2. İsimsiz kolon bırakamazsınız. Örneğin SUM fonsiyonunu kullanacaksınız AS ile kesin bir isim vermek zorundasınız.
3. Birden fazla sorgu yazamazsınız yani sadece bir SELECT ile başlayan bir cümle yapabilirsiniz.
4. T-SQL kullanamazsınız.
5. INSERT, UPDATE veya DELETE kullanamazsınız.
6. Herhangi bir parametre yollayamazsınız.

```
CREATE VIEW view_adı
AS
SELECT sütun_adları
FROM temel_tablo
```

ÖRN:

```
CREATE VIEW MusteriTablo
AS
SELECT *
FROM Musteriler
```

Kullanımı:

```
SELECT * FROM MusteriTablo
SELECT * FROM MusteriTablo WHERE kullaniciAdi LIKE 'A%'
```

ÖRN:

```
CREATE VIEW view_adı
WITH ENCRYPTION
AS
SELECT sütun_adları
FROM temel_tablo
```

SQL Server'da nesneleri güvenlik sebebiyle kilitlemek isteyebiliriz. Böyle bir kullanımda WITH ENCRYPTION ifadesini kullanmalıyız. WITH ENCRYPTION ile oluşturulan nesneye DESIGN seçeneği ile erişilemez.

VIEW GÜNCELLEŞTİRME

```
Alter VIEW view_adı
AS
SELECT sütun_adları
FROM temel_tablo
```

- STORED PROCEDURE

Yapısal Yordamlar anlamına gelmektedir. Stored Procedure (SP) belirli bir işlevi, görevi yerine getirmek için özellikle yapılandırılmış bir veya daha fazla tablo, SP vs. ile ilişki kod parçacıklarıdır. Kısaca Derlenmiş SQL cümleciği diyebiliriz.
SP'ler veri tabanında saklanan SQL ifadeleridir. Programlama dillerinde olduğu gibi parametreler içerir. Bu parametrelere göre çalışıp sonuçları listeler. Bir SP içinde başka SP'leri çağırıp çalıştırabiliriz. Veri tabanlarında saklandığından dolayı daha hızlı çalışırlar. Normal kod ile çağırmak yerine kodla sadece SP'ye parametre gönderip çağırmak performans açısından mükemmeldir. SP'ler ilk çalıştıklarında derlenirler. Tekrar çalıştıklarında derlenmezler. Bu işlemde performans açısından güzel bir olaydır. SQL komutu çağırıldığında ayrıştırma, derleme ve çalıştırma aşamalarından geçmektedir. SP'ler daha önceden derlendikleri için, normal kullanılan bir SQL sorgusundan çok daha performanslı olup ayrıca ağ trafiğini de yormayan yapısıyla programlamada çok kullanılan bir kod parçacığıdır.

ÖRN: Musteriler tablosundaki kayıtları listeler.

```
CREATE PROCEDURE MusteriListe
AS
SELECT * FROM Musteriler
```

Çalıştırılması: `EXECUTE MusteriListe`

ÖRN: Musteriler tablomuza veri ekleme amaçlı "KisiEkle" adında bir prosedür yazalım ve çalıştıralım.

```
CREATE PROCEDURE KisiEkle
(
	@MusteriID char(5),
	@TCKimlikNo char(11),
	@Ad varchar(50),
	@Soyad varchar(50),
	@Sehir varchar(2)
)

AS

INSERT INTO Musteriler(MusteriID, TCKimlikNo, Ad, Soyad, Sehir)
VALUES (@MusteriID,@TCKimlikNo,@Ad, @Soyad, @Sehir)
```

Çalıştırılması: `EXECUTE KisiEkle  'ist10','12369856324','Mustafa','Acar', 34`

ÖRN: Benzer şekilde eklediğimiz kayıtları silebilmek için bir "sil" prosedürü yazalım ve çalıştıralım.

```
CREATE PROCEDURE MusteriSil
(
	@MusteriID char(5)
)
AS
DELETE FROM Musteriler WHERE MusteriID=@MusteriID
```

Çalıştırılması: `EXECUTE MusteriSil 'ist10'`

ÖRN: Eklemiş olduğumuz veriler üzerinde ID belirterek kolaylıkla Ad kaydını değiştirebileceğimiz "MusteriGuncelle" adında prosedürü yazalım ve çalıştıralım.

```
CREATE PROCEDURE MusteriGuncelle
(
	@Ad nvarchar(50),
	@MusteriID char(5)
)

AS

UPDATE Musteriler SET Ad=@Ad
WHERE MusteriID=@MusteriID
```

Çalıştırılması: `EXECUTE MusteriGuncelle 'Ömer', 'ist10'`

ÖRN: Tablomuzda arama yapmak amacıyla kullanılacak basit bir arama prosedürü yazalım ve çalıştıralım.

```
CREATE PROCEDURE MusteriAra
(
	@Ad nvarchar(50)
)

AS

SELECT * FROM Musteriler  WHERE Ad LIKE '%' + @Ad + '%'

EXECUTE MusteriAra 'Ö'
```

- TRIGGER

TRIGGER Türkçe anlamı olarak Tetikleyici demektir. TRIGGER, STORED PROCEDURE gibi SQL Server içinde bileşen olarak bulunmaktadır. Triggerların kullanım amacı, tablo üzerinde bir işlem gerçekleştiğinde (INSERT, UPDATE, DELETE) başka bir işlem daha yapılmak istendiği zaman kullanılır.

ÖRN: TRIGGER Oluşturmak

```
CREATE TRIGGER trigger_adi
ON tablo_adi

{FOR|AFTER|INSTEAD OF} {INSERT|UPDATE|DELETE}

AS

BEGIN

--SQL ifadeler

END
```

AFTER: Insert, Update yada Delete işlemi gerçekleştirildiğinde yani işlem yapıldıktan sonra tetiklenir. Sadece tablolar için tanımlanabilir.

INSTEAD OF: Belirlenen işlem gerçekleşirken devreye girip, SQL sorgusu yerine çalıştırılır.

INSERT|UPDATE|DELETE: Tabloda hangi SQL işlemi sırasında çalıştırılacağını belirtiyoruz.

BEGIN ... END: Trigger çalıştırıldıktan sonra yürütülecek olan SQL cümlelerini bu aralıkta yazıyoruz.

ÖRN: Musteriler tablosuna kayıt eklendikten sonra Müşterileri listeleyen tetikleyici.

```
CREATE TRIGGER MusteriListele
ON Musteriler
AFTER INSERT
AS
BEGIN
SELECT * FROM Musteriler
END
```

Aşağıdaki SQL komutu ile kayıt ekledikten sonra otomatik olarak Müşteriler listelenecektir.

```
INSERT INTO Musteriler (MusteriID, TCKimlikNo, Ad, Soyad, Sehir)
VALUES ('sak06', 32698653269, 'Talha', 'Akyalçın', 54)
```

ÖRN: Musteriler tablosunda bir kayıt silindiğinde diğer tabloya ekler.

```
CREATE TRIGGER KMusteriTakip ON Musteriler
AFTER DELETE
AS BEGIN
INSERT INTO KMusteri select MusteriID, TCKimlikNo, Ad, Soyad, Sehir FROM deleted
END
```

TRIGGER GÜNCELLEME

```
ALTER TRIGGER trigger_adi
ON tablo_adi

AFTER veya INSTEAD OF (INSERT , UPDATE , DELETE)
AS

BEGIN

--SQL ifadeler

END
```

TRIGGER ENABLE/DISABLE

```
ENABLE TRIGGER MusteriListele ON Musteriler

DISABLE TRIGGER MusteriListele ON Musteriler
```

TÜM TRIGGERLARI AKTİFLEŞTİRME/PASİFLEŞTİRME

```
ENABLE TRIGGER ALL ON tablo_adı

DISABLE TRIGGER ALL ON tablo_adı
```

TRIGGER SİLME

```
DROP TRIGGER trigger_adı
```

- FUNCTIONS

Fonksiyonlar tamamen işimizi kolaylaştırmak adına sürekli olarak tekrarladığımız SQL sorgularına tek bir noktadan erişmemizi sağlar. Buda bize hızlı bir erişim imkânı, hızlı bir hata kontrol mekanizması, çabuk müdahale, sorgu tekrarlamama gibi imkânları verir.

ÖRN:

```
CREATE FUNCTION MBirlestir (@kelime1 varchar(50), @kelime2 varchar(50), @kelime3
char(11))

RETURNS varchar(113)

AS
BEGIN

RETURN rtrim(@kelime1) + space(1) + rtrim(@kelime2) + space(1) + rtrim(@kelime3)

END
```

Kullanımı: `SELECT dbo.MBirlestir (Ad, Soyad, TCKimlikNo) FROM Musteriler`

ÖRN:

```
CREATE FUNCTION TOPLAMA(@SAYI1 INT, @SAYI2 INT)

RETURNS INT -- RETURN DEĞERININ INT OLACAĞI BELİRTİLİYOR

AS
BEGIN

    DECLARE @TOPLAM INT
    SET @TOPLAM = @SAYI1+ @SAYI2
    RETURN @TOPLAM

END
```

Kullanımı: `SELECT dbo.TOPLAMA(25,15)`