

# SQL KOMUTLARI

## 1) DATA MANİPÜLASYON KOMUTLARI

**SELECT** = Veritabanındaki tablolardan kayıtları çeker

**SELECT** KOLON1,KOLON2,KOLON3,....

**FROM** TABLOADI

**WHERE** <ŞARTLAR>

Örnek 1 :

**SELECT\* FROM** CUSTOMERS yazarsan tüm tabloyu çeker

Örnek 2 :

**SELECT** ID,CUSTOMERNAME,CITY **FROM** CUSTOMERS

(Sadece ID,CUSTOMERNAME ve CITY'yi çektin)

**UPDATE** = Bir tablodaki kaydın bir ya da daha fazla alanını günceller, değiştirir

**UPDATE** TABLOADI

**SET** COLUMN1=VALUE1,COLUMN2=VALUE2...

**WHERE** <ŞARTLAR>

Örnek 1 :

**SELECT \* FROM** CUSTOMERS

**UPDATE** CUSTOMERS

**SET** NATION='TR'

(Herkesin milletini TR yaptın)

Örnek 2 :

**SELECT \* FROM** CUSTOMERS

**UPDATE** CUSTOMERS

**SET** NATION='TR' , AGE=24

(Herkesin milletini TR ve yaşını 24 yaptın)

Örnek 3 :

**SELECT \* FROM** CUSTOMERS

**UPDATE** CUSTOMERS

**SET** NATION='TR' , AGE=**DATEDIFF**(YEAR,BIRTHDATE,**GETDATE**())

(Herkesin milletini TR , yaşlarını ise güncel yaş olarak yaptın)

**GETDATE**=Şimdiki Tarih

**UPDATE**=Güncelleme

**CONVERT**=Dönüştürmek

**INSERT** = Tabloya yeni kayıt ekler

**INSERT INTO TABLOADI**

(KOLON1,KOLON2,KOLON3,...)

**VALUES**

(DEĞER1,DEĞER2,DEĞER3,...)

Örnek 1 :

**SELECT\* FROM CUSTOMERS**

**INSERT INTO CUSTOMERS**

(CUSTOMERNAME,CİTY,BİRTHDATE,DİSTRCİT,GENDER)

**VALUES**

('LEYLA TORAMAN','MALATYA','1972.05.06','PÜTÜRGE','K')

**DELETE** = Bir tablodan kayıt siler

**DELETE FROM TABLOADI**

**WHERE** <ŞARTLAR>

Örnek 1 :

**SELECT \* FROM CUSTOMERS**

**DELETE FROM CUSTOMERS**

(Tablo tamamen silindi ve otomatik artan alan varsa en son kaldığı değerden devam eder)

**DELETE FROM SALES WHERE ID:11503**

**TRUNCATE** = Tabloyu ilk oluşturduğumuz hale getirir, tablonun içeriğini boşaltır

**SELECT \* FROM CUSTOMERS**

**TRUNCATE FROM CUSTOMERS**

(Tablonun ilk oluşturulduğu hale geldi)

**Not** = DELETE komutu tablodan belirli bir satırı silmek için kullanılırken, TRUNCATE komutu tablodan tüm satırları kaldırmak için kullanılır.

**WHERE ŞARTI** = Filtreme özelliği

**SELECT** KOLON1,KOLON2,KOLON3,....

**FROM** TABLOADI

**WHERE** <ŞARTLAR>

Örnek 1 :

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL' (Tablodan sadece İstanbulluları çekersin)

**=** Eşittir

**<>** Eşit değildir

**>** Büyüktür

**<** Küçüktür

**>=** Büyüktür ya da eşittir

**<=** Küçüktür ya da eşittir

**BETWEEN** Arasındadır (**WHERE** AGE **BETWEEN** 20 AND 30)

**LIKE** İle başlar, İle biter, İçerir

**SELECT \* FROM** CUSTOMERS

(**WHERE** CUSTOMERNAME **LIKE** 'SALİH%')

(İsmi SALİH ile başlayanları getir)

(**WHERE** CUSTOMERNAME **LIKE** '%TORAMAN')

(Soyismi TORAMAN ile bitenleri getir)

(**WHERE** CUSTOMERNAME **LIKE** '%SALİH%')

(İçinde SALİH geçenleri getir)

(**WHERE** CUSTOMERNAME **LIKE** '-A%')

(2.harfi A ile başlayanları getir)

**IN** İçindedir

**WHERE** CITY **IN** ('İSTANBUL','ANKARA','BURSA')

(Şehri İSTANBUL,ANKARA VE BURSA olanları getir)

**NOT LIKE** İle başlamaz, İle bitmez, İçermez

**NOT LIKE = ! (Ünlem)**

(**WHERE** CUSTOMERNAME **NOT LIKE** 'SALİH%')

(İsmi SALİH ile başlamayanları getir)

**NOT IN** İçinde değildir

## AND / OR OPERATÖRLERİ

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL' **AND** GENDER='E'

(Doğru bir önermedir çünkü bir kişinin şehri İstanbul ve cinsiyeti Erkek olabilir)

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL' **AND** DISTRICT='BAĞCILAR' **AND** DISTRICT='ŞİŞLİ'

(Yanlış bir önermedir çünkü bir kişinin ilçesi hem Bağcılar hem de Şişli olamaz)

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL' **AND** GENDER='E' **AND** BIRTHDATE **BETWEEN**  
'1990.01.01' **AND** '2022.01.01'

(Şehri İstanbul, cinsiyeti Erkek ve doğum tarihi 1990.01.01 ile 2022.01.01 arasında olanlar)

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL' **AND** DISTRICT='ÜSKÜDAR' **OR** DISTRICT='SARIYER'

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL' **OR** CITY='İZMİR'

Not: **WHERE** - -CITY IN ('İSTANBUL','İZMİR') yani başına 2 tane tire (- -) koyduğun zaman bu satırı yorum satırı haline getiriyor yani olmasa da olur bir etkisi yok

**DISTINCT**= Tekrar eden satırları tekilliştirmek amacıyla kullanılan bir komut

**SELECT DISTINCT** KOLON1,KOLON2,KOLON3,....

**FROM** TABLOADI

**WHERE** <ŞARTLAR>

Örnek 1 :

**SELECT DISTINCT** CITY,DISTRICT **FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL'

(İstanbul için ilçeleri gösterir ve tekrar eden İstanbulları kaldırır)

## ORDER BY

**SELECT** KOLON1,KOLON2,KOLON3,....

**FROM** TABLOADI

**WHERE** <ŞARTLAR>

**ORDER BY** KOLON1 ASC, KOLON2 DESC, KOLON3 ASC

ASC= Küçükten büyüğe doğru sıralama ya da A'dan Z'ye sıralama

DESC= Büyükten küçüğe doğru sıralama

Örnek 1 :

**SELECT \* FROM** CUSTOMERS

**ORDER BY ID ASC** (ID'ler küçükten büyüğe doğru sıralandı)

**SELECT \* FROM** CUSTOMERS

**ORDER BY ID DESC** (ID'ler büyükten küçüğe doğru sıralandı)

**SELECT \* FROM** CUSTOMERS

**ORDER BY CUSTOMERNAME ASC** (İsimler A'dan Z'ye sıralandı)

**SELECT \* FROM** CUSTOMERS

**ORDER BY CUSTOMERNAME** (İsimler **ASC** yazmasan da A'dan Z'ye sıralandı)

**SELECT \* FROM** CUSTOMERS

**ORDER BY CITY,CUSTOMERNAME**

(İlk olarak şehirler A'dan Z'ye sıralandı sonra ise müşteri isimleri A'dan Z'ye sıralandı)

**SELECT \* FROM** CUSTOMERS

**ORDER BY CITY ASC,CUSTOMERNAME DESC**

(İlk olarak şehirler A'dan Z'ye sıralandı sonra ise müşteri isimleri Z'den A'ya sıralandı)

**SEL \* FROM** CUSTOMERS

**ORDER BY CITY,DISTRICT,CUSTOMERNAME**

(İlk olarak şehirler A'dan Z'ye sıralandı sonra ise ilçeler A'dan Z'ye sıralandı en son ise müşteri isimleri A'dan Z'ye sıralandı)

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL'

**ORDER BY CITY,DİSTRİCT,CUSTOMERNAME**

**SELECT \* FROM** CUSTOMERS

**WHERE** CITY='İSTANBUL'

**ORDER BY 2** (ID yerine 2 yazman ile aynı şey çünkü ID 2.sütunda yani direk

**ORDER BY** sütun sayısı yazabilirsin)

**TOP KOMUTU** = Temel itibariyle bir veri setinde dönen kayıtların tamamını görmek yerine bunu bir kısmını görmemizi sağlar mesela 1000 tane kayıt varsa 10 tanesini görmek isteyebiliriz, TOP komutu bunu bize sağlar

**SELECT TOP N** KOLON1,KOLON2,KOLON3,....

**FROM** TABLOADI

**WHERE** <ŞARTLAR>

Örnek 1 :

**SELECT TOP 100 \* FROM** CUSTOMERS (ilk 100 satırı getirir)

Örnek 2 :

**SELECT TOP 100 \* FROM** CUSTOMERS

**ORDER BY** CUSTOMERNAME (A'dan başlayıp ilk 100 satırı getirir)

Örnek 3 :

**SELECT TOP 50 PERCENT \* FROM** CUSTOMERS

**ORDER BY** CUSTOMERNAME (500 satır getirdi PERCENT 10 ile çarpıyormuş gibi)

## 2) VERİTABANI MANİPÜLASYON KOMUTLARI

**CREATE** = Bir veritabanı nesnesini oluşturur

**ALTER** = Bir veritabanı nesnesinin özelliğini geliştir

**DROP** = Bir veritabanı nesnesini siler

**CREATE DATABASE** = Yeni veritabanı oluşturur

**ALTER DATABASE** = Bir veritabanının özelliklerini değiştirir

**CREATE TABLE** = Yeni bir tablo oluşturur

**ALTER TABLE** = Bir tablonun özelliklerini değiştirir

**DROP TABLE** = Bir tabloyu tamamen siler

**CREATE INDEX** = Index oluşturur

**DROP INDEX** = Index'i siler

## AGGREGATE FUNCTIONS (SUM, MIN, MAX, AVG, COUNT)

**SELECT**

SUM(PRICE), COUNT(ID), MIN(PRICE), MAX(PRICE), AX(PRICE),AVG(PRICE)

**FROM** TABLO ADI

**SUM**; bir tabloda ilgili sütuna ait değerlerin toplamını hesaplayan komut

**COUNT**; bir tabloda istenen nitelikteki değer kaç adet olduğunu verir

**AVG**; bir tabloda ilgili sütuna ait değerlerin ortalamasını hesaplayan komut

**AMOUNT**=Ürün adedi

Örnek 1:

**SELECT SUM**(AMOUNT) **FROM** SALES

(Toplam satılan ürün adetini gösterir)

**SELECT COUNT (\*)** **FROM** SALES

(Toplam satır sayısını gösterir)

**SELECT AVG**(AMOUNT) **FROM** SALES

(1 satışta ortalama satılan ürün adetini gösterir)

**SELECT MIN**(AMOUNT), **MAX**(AMOUNT) **FROM** SALES

(Minimum ve maksimum alınan ürün adetini gösterir)

**SELECT \* FROM** SALES **ORDER BY** AMOUNT

(Müşterilerin tek seferde kaç adet ürün aldığını gösterir)

**SELECT \* FROM** SALES **ORDER BY** TOTALPRICE

(Toplam tutar alanını gösterir)

**SELECT MIN**(TOTALPRICE), **MAX**(TOTALPRICE), **COUNT**(FICHENO),

**SUM**(TOTALPRICE), **AVG**(TOTALPRICE) **FROM** SALES

(Tek seferde okutulan en düşük tutar, en yüksek tutar, toplam hareket sayısı, toplam ciro ve ortalama bir satırın bedeli bulunur)

**SELECT MIN**(TOTALPRICE), **MAX**(TOTALPRICE), **COUNT**(FICHENO),

**SUM**(TOTALPRICE), **AVG**(TOTALPRICE) **FROM** SALES **WHERE** CITY='BURSA'

(Bursa da tek seferde okutulan en düşük tutar, en yüksek tutar, toplam hareket sayısı, toplam ciro ve ortalama bir satırın bedeli bulunur)

# GROUP BY

Tablolarımızı belirli kriterlere göre gruptandırmak istediğimiz zaman kullandığımız yapılarıdır.

```
SELECT KOLON1, KOLON2...  
SUM(PRICE), COUNT(ID), MIN(PRICE), MAX(PRICE), AX(PRICE),AVG(PRICE)  
FROM TABLO ADI  
GROUP BY KOLON1,KOLON2..
```

Örnek 1 :

```
SELECT CITY  
MIN(TOTALPRICE) , MAX(TOTALPRICE) , COUNT(FICHENO) ,  
SUM(TOTALPRICE), AVG(TOTALPRICE) FROM SALES GROUP BY CITY
```

(Her bir şehir için tek seferde okutulan en düşük tutar, en yüksek tutar, toplam hareket sayısı, toplam ciro ve ortalama bir satırın bedeli bulunur)

```
SELECT CITY  
MIN(TOTALPRICE) AS MINPRICE,  
MAX(TOTALPRICE) AS MAXPRICE,  
COUNT(FICHENO) ROWCOUNT_,  
SUM(TOTALPRICE) TOTAL PRICE,  
AVG(TOTALPRICE) AVG PRICE,  
FROM SALES GROUP BY CITY
```

(Sebebi sütun başında yazan “No column name” yazısını anlamlandırmak)

```
SELECT CITY  
MIN(TOTALPRICE) AS MINPRICE,  
MAX(TOTALPRICE) AS MAXPRICE,  
COUNT(FICHENO) ROWCOUNT_,  
SUM(TOTALPRICE) TOTAL PRICE,  
AVG(TOTALPRICE) AVG PRICE,  
FROM SALES GROUP BY CITY ORDER BY CITY
```

(Şehirleri alfabetik sıralayıp her bir şehir için tek seferde okutulan en düşük tutar, en yüksek tutar, toplam hareket sayısı, toplam ciro ve ortalama bir satırın bedeli bulunur)



```
SELECT CITY
```

```
MIN(TOTALPRICE) AS MINPRICE,
```

```
MAX(TOTALPRICE) AS MAXPRICE,
```

```
COUNT(FICHENO) ROWCOUNT_,
```

```
SUM(TOTALPRICE) TOTAL PRICE,
```

```
AVG(TOTALPRICE) AVG PRICE,
```

```
FROM SALES GROUP BY CITY ORDER BY SUM(TOTALPRICE)
```

(En çok satış yapan şehirleri sırasıyla sıralayıp tek seferde okutulan en düşük tutar, en yüksek tutar, toplam hareket sayısı, toplam ciro ve ortalama bir satırın bedeli bulunur)

```
SELECT TOP 10 CITY,
```

```
SUM(TOTALPRICE) AS TOTALPRICE
```

```
FROM SALES
```

```
GROUP BY CITY
```

```
ORDER BY SUM(TOTALPRICE) DESC
```

(En çok satış yapan **10** şehiri **yukarıdan aşağı** sıralar)

## GROUP BY UYGULAMALARI

Örnek: Bir şehrin gün bazlı satış rakamlarını getirme

```
SELECT CONVERT (DATE, '2019-01-01 08:46:10.000')
```

(Sadece tarih bilgisini çeker yani 2019-01-01 olarak çeker)

```
SELECT CONVERT (TIME, '2019-01-01 08:46:10.000')
```

(Sadece saat bilgisini çeker yani 08:46:10.000 olarak çeker)

```
SELECT CONVERT (DATETIME, '2019-01-01 08:46:10.000')
```

(Tarih ve saat bilgisini çeker yani 2019-01-01 08:46:10.000 olarak çeker)

```
SELECT CITY, DATE, SUM(TOTALPRICE) AS TOTALPRICE
```

```
FROM SALES
```

```
WHERE CITY='ANKARA'
```

```
GROUP BY CITY,DATE
```

```
ORDER BY CITY,DATE
```

(Ankara şehrinin **günlere göre sırasıyla toplam cirosunu** gösterir ve toplam ciro sütununun adı **TOTALPRICE** olarak gösterildi)

Örnek: Bir günün şehir bazlı satış rakamlarını getirme

```
SELECT CITY, DATE, SUM(TOTALPRICE) AS TOTALPRICE
FROM SALES
WHERE DATE='2019-01-01'
GROUP BY DATE,CITY
ORDER BY DATE,CITY
```

(2019-01-01 tarihinin şehirlere göre sırasıyla toplam cirosunu gösterir ve toplam ciro sütununun adı TOTALPRICE olarak gösterildi)

```
SELECT DATE,CITY, SUM(TOTALPRICE) AS TOTALPRICE
FROM SALES
WHERE DATE='2019-01-01'
GROUP BY DATE,CITY
ORDER BY DATE, SUM(TOTALPRICE) DESC
```

(2019-01-01 tarihinin şehirlere göre en çok satanlar sırasıyla toplam cirosunu gösterir ve toplam ciro sütununun adı TOTALPRICE olarak gösterildi)

## NOT

DATE\_ sütunu 2019-01-01 08:07:24.100 şeklinde tarih ve saat olarak ise  
UPDATE CUSTOMERS SET DATE2=CONVERT(DATE,DATE\_) yazarak  
DATE2 tablosunu sadece tarihlerden oluşturdun.

UPDATE=GÜNCELLEME  
CONVERT=DÖNÜŞTÜRMEK

## Örnek: Şehirlerin aylara göre satış rakamlarını getirme

```
SELECT DATEPART(MONTH,'2019-08-02')
```

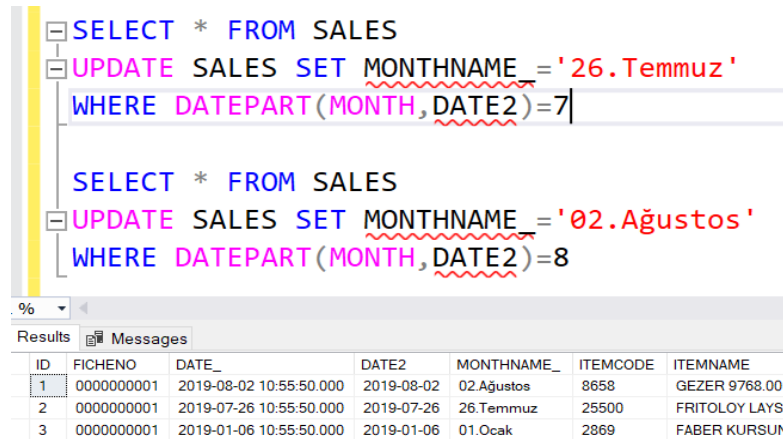
(Bize ay numarasını gönderir yani 8 i gönderir)

```
SELECT * FROM SALES
```

```
UPDATE SALES SET MONTHNAME_='26.Temmuz'
```

```
WHERE DATEPART(MONTH,DATE2)=7
```

(**Temmuz** ayına denk gelenler için MONTHNAME alanı 26.Temmuz olarak değişti)



```
SELECT * FROM SALES
UPDATE SALES SET MONTHNAME_='26.Temmuz'
WHERE DATEPART(MONTH,DATE2)=7

SELECT * FROM SALES
UPDATE SALES SET MONTHNAME_='02.Ağustos'
WHERE DATEPART(MONTH,DATE2)=8
```

ID	FICHENO	DATE_	DATE2	MONTHNAME_	ITEMCODE	ITEMNAME
1	0000000001	2019-08-02 10:55:50.000	2019-08-02	02.Ağustos	8658	GEZER 9768.00.
2	0000000001	2019-07-26 10:55:50.000	2019-07-26	26.Temmuz	25500	FRITOLOY LAYS
3	0000000001	2019-01-06 10:55:50.000	2019-01-06	01.Ocak	2869	FABER KURSUN

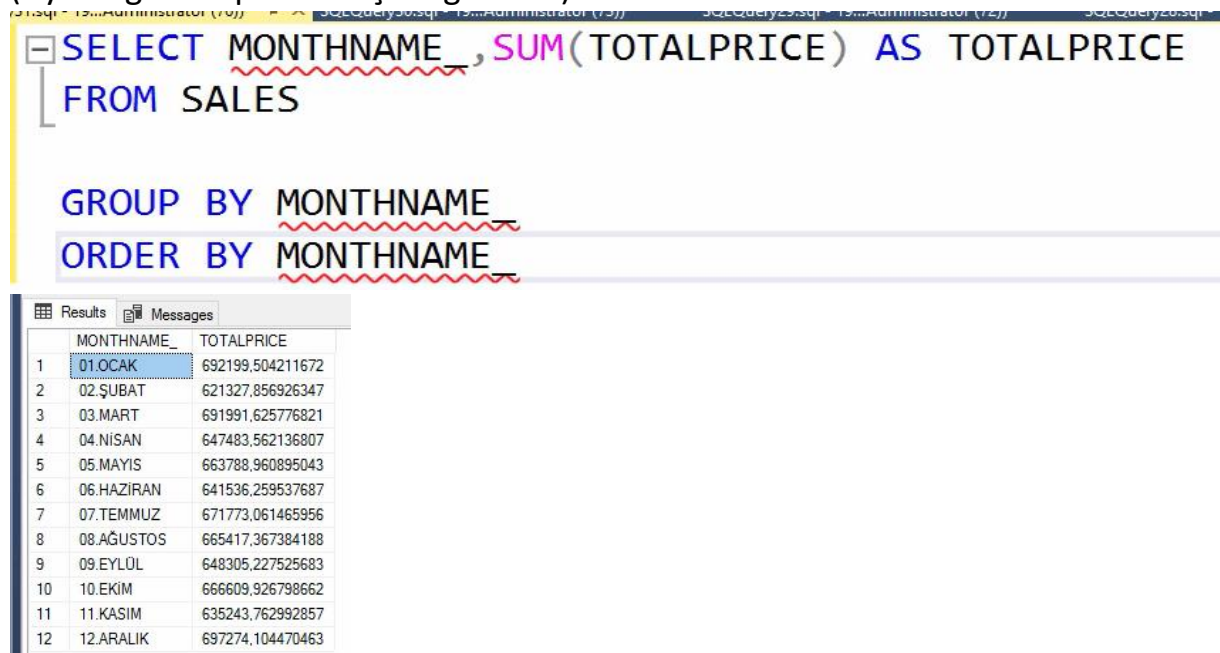
```
SELECT MONTHNAME_, SUM(TOTALPRICE) AS TOTALPRICE
```

```
FROM SALES
```

```
GROUP BY MONTHNAME_
```

```
ORDER BY MONTHNAME_
```

(Aylara göre toplam satışları gösterir)



```
SELECT MONTHNAME_, SUM(TOTALPRICE) AS TOTALPRICE
FROM SALES

GROUP BY MONTHNAME_
ORDER BY MONTHNAME_
```

	MONTHNAME_	TOTALPRICE
1	01.OCAK	692199.504211672
2	02.ŞUBAT	621327.856926347
3	03.MART	691991.625776821
4	04.NISAN	647483.562136807
5	05.MAYIS	663788.960895043
6	06.HAZİRAN	641536.259537687
7	07.TEMMUZ	671773.061465956
8	08.AĞUSTOS	665417.367384188
9	09.EYLÜL	648305.227525683
10	10.EKİM	666609.926798662
11	11.KASIM	635243.762992857
12	12.ARALIK	697274.104470463

## Şehirlerin aylara göre satışları

```
SELECT CITY, MONTHNAME_, SUM(TOTALPRICE) AS TOTALPRICE
FROM SALES
```

```
GROUP BY CITY, MONTHNAME_
ORDER BY CITY, MONTHNAME_
```

236 %

	CITY	MONTHNAME_	TOTALPRICE
1	Adana	01.OCAK	26870,163392
2	Adana	02.ŞUBAT	22335,062763196
3	Adana	03.MART	28708,767834032
4	Adana	04.NISAN	27612,98324
5	Adana	05.MAYIS	26812,13440226
6	Adana	06.HAZİRAN	22032,29044
7	Adana	07.TEMMUZ	26317,01868
8	Adana	08.AĞUSTOS	24345,6834752
9	Adana	09.EYLÜL	20889,2718
10	Adana	10.EKİM	27807,86795144
11	Adana	11.KASIM	22915,153653148
12	Adana	12.ARALIK	30999,36454536
13	Adıyaman	01.OCAK	4141,624
14	Adıyaman	02.ŞUBAT	2321,90848
15	Adıyaman	03.MART	6022,872
16	Adıyaman	04.NISAN	3263,3208

## Aylara göre en çok satış yapan şehirlerin sıralanışı

```
SELECT MONTHNAME_, CITY, SUM(TOTALPRICE) AS TOTALPRICE
FROM SALES
```

```
GROUP BY CITY, MONTHNAME_
ORDER BY MONTHNAME_, SUM(TOTALPRICE) DESC
```

	MONTHNAME_	CITY	TOTALPRICE		MONTHNAME_	CITY	TOTALPRICE
1	01.OCAK	İstanbul	199753,132871284	81	02.ŞUBAT	İstanbul	181636,954933812
2	01.OCAK	Ankara	68068,672429324	82	02.ŞUBAT	Ankara	66337,919646032
3	01.OCAK	İzmir	52909,376123152	83	02.ŞUBAT	İzmir	40239,77056
4	01.OCAK	Bursa	31053,87632	84	02.ŞUBAT	Bursa	25011,327428432
5	01.OCAK	Adana	26870,163392	85	02.ŞUBAT	Adana	22335,062763196
6	01.OCAK	Konya	20901,1596	86	02.ŞUBAT	Gaziantep	19984,14424
7	01.OCAK	Gaziantep	20885,234758628	87	02.ŞUBAT	Antalya	16740,291244
8	01.OCAK	Kayseri	16624,20004	88	02.ŞUBAT	Konya	13162,13432
9	01.OCAK	Mersin	13370,774754	89	02.ŞUBAT	Eskişehir	11426,79552
10	01.OCAK	Diyarbakır	13105,06044	90	02.ŞUBAT	Mersin	11362,47476
11	01.OCAK	Eskişehir	12029,6944	91	02.ŞUBAT	Diyarbakır	10853,3476
12	01.OCAK	Antalya	10846,488992224	92	02.ŞUBAT	Kayseri	10675,653914
13	01.OCAK	Samsun	8905,75264	93	02.ŞUBAT	Samsun	9212,5645428
14	01.OCAK	Elazığ	8621,988	94	02.ŞUBAT	Van	8844,566
				95	02.ŞUBAT	Şanlıurfa	8778,2864

Örnek: Ürün kategorilerine göre satış rakamlarını getirme

```
SELECT CATEGORY1,CATEGORY2,CATEGORY3,CATEGORY4
```

```
SUM(TOTALPRICE) AS TOTALPRICE
```

```
COUNT(*) AS ROWCOUNT_,
```

```
SUM(AMOUNT) AS TOTALAMOUNT
```

```
FROM SALES
```

```
GROUP BY CATEGORY1,CATEGORY2,CATEGORY3,CATEGORY4
```

```
ORDER BY CATEGORY1,CATEGORY2,CATEGORY3,CATEGORY4
```

(Her bir ürünün **toplam cirosunun** ne kadar olduğu , **toplam satır sayısı** ve **adet olarak** ne kadar satıldığı getirilmiştir)

Örnek: Mağazaların müşteri sayılarını getirme

```
SELECT
```

```
FICHENO, COUNT(*)
```

```
FROM SALES WHERE MONTHNAME='01.Ocak'
```

```
AND CITY='ADANA'
```

FICHENO	DATE1	DATE2	MONTHNAME	IT
0000000043	2019-01-22 11:19:10.000	2019-01-22	01.OCAK	9
0000000043	2019-01-11 11:19:10.000	2019-01-11	01.OCAK	2
0000000043	2019-01-24 11:19:10.000	2019-01-24	01.OCAK	1
0000000043	2019-01-01 11:19:10.000	2019-01-01	01.OCAK	9
0000000158	2019-01-07 13:34:29.000	2019-01-07	01.OCAK	2
0000000158	2019-01-04 13:34:29.000	2019-01-04	01.OCAK	1
0000000314	2019-01-21 08:55:12.000	2019-01-21	01.OCAK	6

388 rows

(Aynı fişten 4 satır işlem gerçekleştirildiğini ve 388 tane satır geldi yani toplam barkod sayısı 308'dir)

```
SELECT
```

```
FICHENO, COUNT(*)
```

```
FROM SALES WHERE MONTHNAME='01.Ocak'
```

```
AND CITY='ADANA'
```

```
GROUP BY FICHENO
```

	FICHENO	(No column name)
1	0000000043	4
2	0000000158	2
3	0000000322	1
4	0000001167	1
5	0000001240	1
6	0000001323	1
7	0000001325	3
8	0000001403	2

306 rows

(Aynı fişte yapılan işlemler birleştirilir ve 306 tane satır geldiğini gelir) Yani burada gerçekte 306 tane fiş kesilmiş ve kimisinde 4 kere barkod okutulmuş kimisinde 2 kimisinde 1 ve bunlar tekilleştirildiğinde satır sayısı 306 ya düştü sonuç olarak Adana mağazasında gerçek müşteri sayısı 306'dır)

```

SELECT CITY,
COUNT(*)
FROM SALES WHERE MONTHNAME='01.Ocak'
GROUP BY CITY
ORDER BY CITY

```

```

SELECT CITY,
COUNT (*)
FROM SALES WHERE MONTHNAME = '01.Ocak'
GROUP BY CITY
ORDER BY CITY

```

	CITY	(No column name)
1	Adana	388
2	Adıyaman	63
3	Afyonkarahisar	42
4	Ağrı	39
5	Aksaray	51
6	Amasya	20
7	Ankara	1056
8	Antalya	164

(Şehirlere göre tekrar eden satırlarla birlikte toplam barkod sayısı gösterildi)

```

SELECT CITY,
COUNT(DISTINCT FICHENO)
FROM SALES WHERE MONTHNAME='01.Ocak'
GROUP BY CITY
ORDER BY CITY

```

```

SELECT CITY,
COUNT(DISTINCT FICHENO)
FROM SALES WHERE MONTHNAME = '01.Ocak'
GROUP BY CITY
ORDER BY CITY

```

	CITY	(No column name)
1	Adana	306
2	Adıyaman	49
3	Afyonkarahisar	34
4	Ağrı	30
5	Aksaray	38
6	Amasya	16
7	Ankara	851
8	Antalya	142

(Tekrar eden satırlar tekilleştirildi şehirlere göre gerçek müşteri sayısı)

```

SELECT CITY,
COUNT(DISTINCT CUSTOMERNAME) AS UNIQUECUSTOMER,
COUNT(DISTINCT FICHENO) AS CUSTOMERCOUNT,
COUNT(*) AS ITEMCOUNT
FROM SALES WHERE MONTHNAME='01.Ocak'
GROUP BY CITY
ORDER BY CITY

```

```

SELECT CITY,
COUNT(DISTINCT CUSTOMERNAME) AS UNIQUECUSTOMER,
COUNT(DISTINCT FICHENO) AS CUSTOMERCOUNT,
COUNT(*) AS ITEMCOUNT
FROM SALES WHERE MONTHNAME = '01.Ocak'
GROUP BY CITY
ORDER BY CITY

```

	CITY	UNIQUECUSTOMER	CUSTOMERCOUNT	ITEMCOUNT
1	Adana	303	306	388
2	Adıyaman	49	49	63
3	Afyonkarahisar	34	34	42
4	Ağrı	30	30	39
5	Aksaray	38	38	51
6	Amasya	16	16	20
7	Ankara	845	851	1056
8	Antalya	142	142	164

(Adanada toplam müşteri sayısı 306 ama tekil müşteri sayısı 303 yani 3 müşteri 2 kere gelmiş ve okutulan barkod sayısı 388'dir)



Örnek: Belli bir cironun üzerinde satış yapan şehirleri getirme

```
SELECT CITY, SUM(TOTALPRICE)
FROM SALES
GROUP BY CITY
ORDER BY SUM(TOTALPRICE) DESC
(Toplam cirolara göre şehirleri listeledin)
```

```
SELECT CITY, SUM(TOTALPRICE)
FROM SALES
GROUP BY CITY
HAVING SUM(TOTALPRICE) > 40000
ORDER BY SUM(TOTALPRICE) DESC
(40000 üzerinde toplam cirosu olan şehirleri sıraladın)
```

```
SELECT CITY, SUM(TOTALPRICE)
FROM SALES
WHERE CITY IN ('ANKARA','İSTANBUL','BURSA','İZMİR')
GROUP BY CITY
HAVING SUM(TOTALPRICE) > 40000
ORDER BY SUM(TOTALPRICE) DESC
(Bu şehirler arasında toplam cirosu 40000 üzerinde olanları
sıraladın)
(Burada verdiği bir şeye bir şart arıyorsan bunu HAVING e
vermelisin)
```

# VERİ TİPLERİ

bigint	Minimum: $-2^{63}$ (-9,223,372,036,854,775,808)	8 Byte
	Maksimum: $2^{63}-1$ (9,223,372,036,854,775,807)	
int	Minimum: $-2^{31}$ (-2,147,483,648)	4 Byte
	Maksimum: $2^{31}-1$ (2,147,483,647)	
smallint	Minimum: $-2^{15}$ (-32,768)	2 Byte
	Maksimum: $2^{15}-1$ (32,767)	
tinyint	Minimum: 0	1 Byte
	Maksimum: 255	
bit	0 ya da 1 değerini alır.	Eğer tabloda 8 ya da daha az bit kolonu varsa 1 byte, 8'den fazla ise 2 byte yer kaplar.

A	B	C	D	E	F
KOLON ADI	VERİ TİPİ	KAPLADIĞI ALAN	SATIR SAYISI	KAPLADIĞI ALAN (BYTE)	KAPLADIĞI ALAN (MB)
ID	INT	4	2.000.000	8.000.000	=E2/1024/1024
COUNTRYID	TINYINT	1	2.000.000		
CITYID	SMALLINT	2	2.000.000		
GENDER	BIT	1	2.000.000		
					0,00

A	B	C	D	E	F
KOLON ADI	VERİ TİPİ	KAPLADIĞI ALAN	SATIR SAYISI	KAPLADIĞI ALAN (BYTE)	KAPLADIĞI ALAN (MB)
ID	INT	4	2.000.000	8.000.000	7,63
COUNTRYID	TINYINT	1	2.000.000	2.000.000	1,91
CITYID	SMALLINT	2	2.000.000	4.000.000	3,81
GENDER	BIT	1	2.000.000	2.000.000	1,91
					15,26



decimal/ numeric	Minimum: $-10^{38} + 1$	Hassasiyetine göre diskte kapladığı alan değişir.
		1'den 9'a kadar Hassasiyet için: 5 byte
	Maksimum: $10^{38} - 1$ .	10'dan 19'a kadar Hassasiyet için: 9 byte
		20'den 28'a kadar Hassasiyet için: 13 byte
		29'dan 38'e kadar Hassasiyet için: 17 byte
money	Minimum: -922,337,203,685,477.5808 Maksimum: 922,337,203,685,477.5807	8 Byte
smallmoney	Minimum: -214,748.3648	4 Byte
	Maksimum: 214,748.3647	
float	-1.79308 ile -2.23308, 0	7 basamağa kadar 4 Byte
	2.23308 ile 1.79308	15 basamağa kadar 8 Byte
Real	-3.438 ile -1.1838, 0	4 Byte
	1.1838 ile 3.438	

decimal(18,0) yani maksimum 18 rakam olabilir

BALANCE	decimal(18, 0)
---------	----------------

BALANCE	Yuvarlar	BALANCE
1425,7		1426

BALANCE	Yuvarlar	BALANCE
1425,3		1425

BALANCE	decimal(18, 4)
---------	----------------

BALANCE	Virgülden sonra 4 hane	BALANCE
1425,3600		1425,0000

BALANCE	float
---------	-------

BALANCE	Virgülden sonra yuvarlamadı
1425,3679867945	

BALANCE	money
---------	-------

BALANCE	Virgülden sonra otomatik 4 hane	BALANCE
1425,6896		14253689835,0000

Yani otomatik olarak (18,4) gibi düşün

Column Name	Data Type	Allow Nulls
ID	int	<input type="checkbox"/>
NAME1	char(50)	<input checked="" type="checkbox"/>
NAME2	nchar(50)	<input checked="" type="checkbox"/>
NAME3	ntext	<input checked="" type="checkbox"/>
NAME4	nvarchar(50)	<input checked="" type="checkbox"/>
NAME5	nvarchar(MAX)	<input checked="" type="checkbox"/>
NAME6	text	<input checked="" type="checkbox"/>
NAME7	varchar(50)	<input checked="" type="checkbox"/>
NAME8	varchar(MAX)	<input checked="" type="checkbox"/>

char(50):

```
SELECT NAME1, NAME2, NAME3, NAME4, NAME5, NAME6, NAME7, NAME8 FROM CUSTOMERS2
```

```
ÖMER
```

nchar(50):

```
SELECT NAME1, NAME2, NAME3, NAME4, NAME5, NAME6, NAME7, NAME8 FROM CUSTOMERS2
```

```
ÖMER
ÖMER
```

text:

```
SELECT NAME1, NAME2, NAME3, NAME4, NAME5, NAME6, NAME7, NAME8 FROM CUSTOMERS2
```

ntext:

```
ÖMER
```

nvarchar(50):

```
ÖMER
```

nvarchar(MAX):

```
ÖMER
```

varchar(50):

```
ÖMER
```

varchar(MAX):

```
ÖMER
```

```
ÖMER
```

**NOT:** İkisi arasındaki farkı şöyle açıklayabiliriz

**char(50)** ve **nchar(50)** de bizim verdiğimiz karakteri sonuna ekliyor fakat **text**, **ntext**, **nvarchar(50)**, **nvarchar(MAX)**, **varchar(50)**, **varchar(MAX)** da ise sonuna herhangi bir karakter veya boşluk ekleme durumu yok isim hafızada ne kadarlık yer kaplıyorsa o kadarlık yer tutuyor.

**1)** Eğer tuttuğumuz alan sabit uzunlukta bir şey ise örneğin TC kimlik numarası, telefon numarası gibi bir şey ise **“char”** kullanmak performans açısından daha mantıklı olabilir.

**2)** Sınırı belli olmayan alanlarda örneğin isim, özgeçmiş gibi bir şey ise **“varchar”** kullanmak performans açısından daha mantıklı olabilir.

**NOT:** “n” uluslararası karakterleri desteklediği için eğer uluslararası dilleri destekleyen yazılım yapıyorsak **nchar** veya **nvarchar** türünde veri tipleri tanımlamamız gerekiyor.

Dezavantaj olarak da **char** veya **varchar** da her karakter 1 byte’lık yer kaplıyor fakat **nchar** ve **nvarchar** da her karakter 2 byte’lık yer tutuyor.

# TARİH-SAAT VERİ TİPLERİ

date	Minimum: 0001-01-01	4 Byte
	Maksimum: 9999-12-31	
smalldate	Minimum: 1900-01-01	3 Byte
	Maksimum: 2079-06-06	
datetime	Minimum: 1753-01-01 00:00:00.000	8 Byte
	Maksimum: 9999-12-31 23:59:59.997	
datetime2	Minimum: 0001-01-01 00:00:00.0000000	1-2 Hassasiyet İçin = 6 Byte
	Maksimum: 9999-12-31 23:59:59.9999999	3-4 Hassasiyet İçin = 7 Byte
		5-7 Hassasiyet İçin = 8 Byte
datetimeoffset	Minimum: 0001-01-01 00:00:00.0000000	1-2 Hassasiyet İçin = 8 Byte
	Maksimum: 9999-12-31 23:59:59.9999999	3-4 Hassasiyet İçin = 9 Byte
	Time zone offsetAralığı: -14:00 / +14:00	5-7 Hassasiyet İçin = 10 Byte
time	Minimum: 00:00:00.0000000	5 Byte(Default olarak kullanılırsa)
	Maksimum: 23:59:59.9999999	

# DİĞER VERİ TİPLERİ

image		Maksimum değeri: 2 <sup>31</sup> -1 (2,147,483,647) Byte
binary	0 ile 8000 arasında	Tanımlandığı değer kadar Byte.
		Binary(10) -> 10 Byte
varbinary	0 ile 8000 arasında	Tanımlandığı değer + 2 Byte
varbinary(MAX)	0 ile 2 147 483 647 arasında	Tanımlandığı değer + 2 Byte
		Maksimum değeri: 2 <sup>31</sup> -1 (2,147,483,647) Byte
sql_variant		Bazı veri tiplerinin değerlerini saklamak için kullanılır.
		Aşağıdakiler hariç:
		varchar(max), varbinary(max), nvarchar(max), xml, text,
		ntext, image, rowversion(timestamp), sql_variant,
Xml		geography, hierarchyid, geometry, User-defined types, datetimeoffset
		Xml veriler için kullanılır.
Table		Sonradan kullanım amacıyla bir sonuç
		kümesini saklamak için kullanılır.

uniqueidentifier	GUID(global olarak tekilliği garanti eder) veriyi tutar.
	select NEWID() script'ini çalıştırdığınızda aşağıdaki
	gibi bir GUID veri oluşturur.
	A4C5DB26-7F18-4B4F-A898-E7DE26A8446A
hierarchyid	Bazen veritabanlarında tekilliği sağlamak için kullanılır.
	Ama bu amaçla kullanıldığında genelde performansı düşürür.
geography	Hiyerarşik yapılarda, hiyerarşideki pozisyonları
	temsil etmek için kullanılır.
geometry	Dünyadaki koordinat sistemini tutar.
	Dünyanın eğimlerini de hesaba katarak.
	Euclidean (flat) sistemi ile koordinat sistemini tutar.
	Sadece 2 düzlem üzerinden hesaplanır.
	Dünyanın eğimlerini hesaba katmaz.

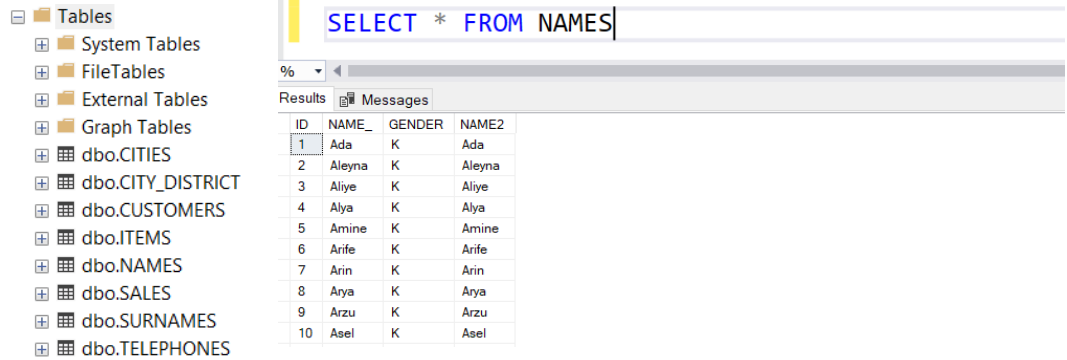
# İKİ TABLOYU BİRLEŞTİRME

`SELECT * FROM CUSTOMERS`

`SELECT CUSTOMERS.* FROM CUSTOMERS`

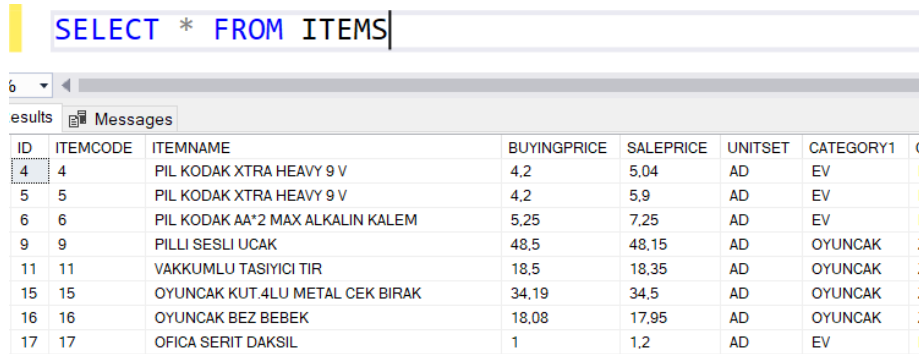
Bu iki komutta aynı şey

İki tabloyu birleştirelim



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Tables' folder is expanded, showing a list of tables including 'dbo.CITIES', 'dbo.CITY\_DISTRICT', 'dbo.CUSTOMERS', 'dbo.ITEMS', 'dbo.NAMES', 'dbo.SALES', 'dbo.SURNAMES', and 'dbo.TELEPHONES'. The 'dbo.NAMES' table is selected. On the right, the SQL query editor shows the query `SELECT * FROM NAMES`. Below the query editor, the 'Results' tab is active, displaying the data from the 'NAMES' table.

ID	NAME_	GENDER	NAME2
1	Ada	K	Ada
2	Aleyna	K	Aleyna
3	Aliye	K	Aliye
4	Alya	K	Alya
5	Amine	K	Amine
6	Arife	K	Arife
7	Arin	K	Arin
8	Arya	K	Arya
9	Arzu	K	Arzu
10	Asel	K	Asel



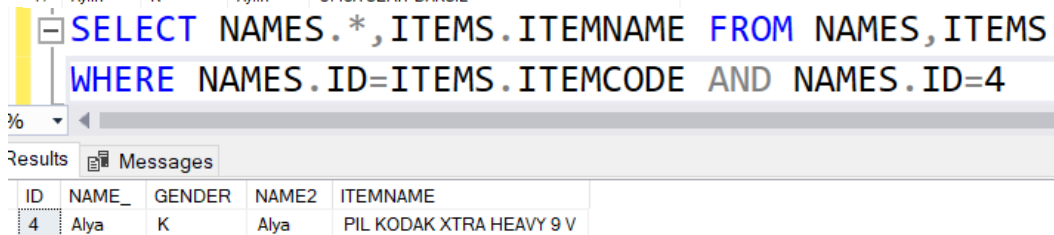
The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Tables' folder is expanded, showing a list of tables including 'dbo.CITIES', 'dbo.CITY\_DISTRICT', 'dbo.CUSTOMERS', 'dbo.ITEMS', 'dbo.NAMES', 'dbo.SALES', 'dbo.SURNAMES', and 'dbo.TELEPHONES'. The 'dbo.ITEMS' table is selected. On the right, the SQL query editor shows the query `SELECT * FROM ITEMS`. Below the query editor, the 'Results' tab is active, displaying the data from the 'ITEMS' table.

ID	ITEMCODE	ITEMNAME	BUYINGPRICE	SALEPRICE	UNITSET	CATEGORY1	CATEGORY2
4	4	PIL KODAK XTRA HEAVY 9 V	4,2	5,04	AD	EV	E
5	5	PIL KODAK XTRA HEAVY 9 V	4,2	5,9	AD	EV	E
6	6	PIL KODAK AA*2 MAX ALKALIN KALEM	5,25	7,25	AD	EV	E
9	9	PILLI SESLİ UÇAK	48,5	48,15	AD	OYUNCAK	Zİ
11	11	VAKKURLU TASIYICI TIR	18,5	18,35	AD	OYUNCAK	Zİ
15	15	OYUNCAK KUT.4LU METAL CEK BIRAK	34,19	34,5	AD	OYUNCAK	Zİ
16	16	OYUNCAK BEZ BEBEK	18,08	17,95	AD	OYUNCAK	Zİ
17	17	OFICA SERIT DAKSİL	1	1,2	AD	EV	K



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Tables' folder is expanded, showing a list of tables including 'dbo.CITIES', 'dbo.CITY\_DISTRICT', 'dbo.CUSTOMERS', 'dbo.ITEMS', 'dbo.NAMES', 'dbo.SALES', 'dbo.SURNAMES', and 'dbo.TELEPHONES'. The 'dbo.NAMES' table is selected. On the right, the SQL query editor shows the query `SELECT NAMES.*, ITEMS.ITEMNAME FROM NAMES, ITEMS WHERE NAMES.ID=ITEMS.ITEMCODE`. Below the query editor, the 'Results' tab is active, displaying the data from the 'NAMES' and 'ITEMS' tables.

ID	NAME_	GENDER	NAME2	ITEMNAME
4	Alya	K	Alya	PIL KODAK XTRA HEAVY 9 V
5	Amine	K	Amine	PIL KODAK XTRA HEAVY 9 V
6	Arife	K	Arife	PIL KODAK AA*2 MAX ALKALIN KALEM
9	Arzu	K	Arzu	PILLI SESLİ UÇAK
11	Asiye	K	Asiye	VAKKURLU TASIYICI TIR
15	Asya	K	Asya	OYUNCAK KUT.4LU METAL CEK BIRAK
16	Ayfer	K	Ayfer	OYUNCAK BEZ BEBEK
17	Aylin	K	Aylin	OFICA SERIT DAKSİL



The screenshot shows the SQL Server Enterprise Manager interface. On the left, the 'Tables' folder is expanded, showing a list of tables including 'dbo.CITIES', 'dbo.CITY\_DISTRICT', 'dbo.CUSTOMERS', 'dbo.ITEMS', 'dbo.NAMES', 'dbo.SALES', 'dbo.SURNAMES', and 'dbo.TELEPHONES'. The 'dbo.NAMES' table is selected. On the right, the SQL query editor shows the query `SELECT NAMES.*, ITEMS.ITEMNAME FROM NAMES, ITEMS WHERE NAMES.ID=ITEMS.ITEMCODE AND NAMES.ID=4`. Below the query editor, the 'Results' tab is active, displaying the data from the 'NAMES' and 'ITEMS' tables.

ID	NAME_	GENDER	NAME2	ITEMNAME
4	Alya	K	Alya	PIL KODAK XTRA HEAVY 9 V

**Not:** Getirmek istediğin sütunları uzun uzun yazmak yerine kısaltma kullanıyorsun.

```
SELECT NAMES.NAME_, NAMES.GENDER, ITEMS.ITEMNAME, ITEMS.CATEGORY1, ITEMS.CATEGORY2, ITEMS.CATEGORY3
FROM NAMES, ITEMS
WHERE NAMES.ID=ITEMS.ITEMCODE
```

	NAME_	GENDER	ITEMNAME	CATEGORY1	CATEGORY2	CATEGORY3
1	Alya	K	PIL KODAK XTRA HEAVY 9 V	EV	ELEKTRIK-ELEKTRONIK	PIL
2	Amine	K	PIL KODAK XTRA HEAVY 9 V	EV	ELEKTRIK-ELEKTRONIK	PIL
3	Arife	K	PIL KODAK AA*2 MAX ALKALIN KALEM	EV	ELEKTRIK-ELEKTRONIK	PIL
4	Arzu	K	PILLI SESLI UCAK	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR
5	Asiye	K	VAKKUMLU TASIYICI TIR	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR
6	Asya	K	OYUNCAK KUT.4LU METAL CEK BIRAK	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR
7	Ayfer	K	OYUNCAK BEZ BEBEK	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR

```
SELECT N.NAME_, N.GENDER, I.ITEMNAME, I.CATEGORY1, I.CATEGORY2, I.CATEGORY3
FROM NAMES N, ITEMS I
WHERE N.ID=I.ITEMCODE
```

	NAME_	GENDER	ITEMNAME	CATEGORY1	CATEGORY2	CATEGORY3
1	Alya	K	PIL KODAK XTRA HEAVY 9 V	EV	ELEKTRIK-ELEKTRONIK	PIL
2	Amine	K	PIL KODAK XTRA HEAVY 9 V	EV	ELEKTRIK-ELEKTRONIK	PIL
3	Arife	K	PIL KODAK AA*2 MAX ALKALIN KALEM	EV	ELEKTRIK-ELEKTRONIK	PIL
4	Arzu	K	PILLI SESLI UCAK	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR
5	Asiye	K	VAKKUMLU TASIYICI TIR	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR
6	Asya	K	OYUNCAK KUT.4LU METAL CEK BIRAK	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR
7	Ayfer	K	OYUNCAK BEZ BEBEK	OYUNCAK	ZEKA GELISTIRICI	OYUNCAKLAR

```
SELECT N.NAME_, N.GENDER, I.ITEMNAME, I.CATEGORY1, I.CATEGORY2, I.CATEGORY3
FROM NAMES N, ITEMS I
WHERE N.ID=I.ITEMCODE AND N.ID=4
```

	NAME_	GENDER	ITEMNAME	CATEGORY1	CATEGORY2	CATEGORY3
1	Alya	K	PIL KODAK XTRA HEAVY 9 V	EV	ELEKTRIK-ELEKTRONIK	PIL

# JOIN

```
SELECT A.KOLON1, A.KOLON2,  
B.KOLON3, B.KOLON4
```

```
FROM TABLO1 A
```

**P.K = PRIMARY KEY**

```
JOIN TABLO2 B ON A.PK_KOLON =
```

```
B.FK_KOLON F.K= FOREIGN KEY
```

```
SELECT N.*, I.ITEMNAME FROM NAMES N  
JOIN ITEMS I ON N.ID=I.ITEMCODE
```

ID	NAME	GENDER	NAME2	ITEMNAME
4	Alya	K	Alya	PIL KODAK XTRA HEAVY 9 V
5	Amine	K	Amine	PIL KODAK XTRA HEAVY 9 V
6	Arife	K	Arife	PIL KODAK AA*2 MAX ALKALIN KALEM
9	Arzu	K	Arzu	PILLI SESLI UCAK
11	Asiye	K	Asiye	VAKKURLU TASIYICI TIR

**Örnek:**

```
SELECT N.*, I.ITEMNAME
```

```
FROM NAMES N
```

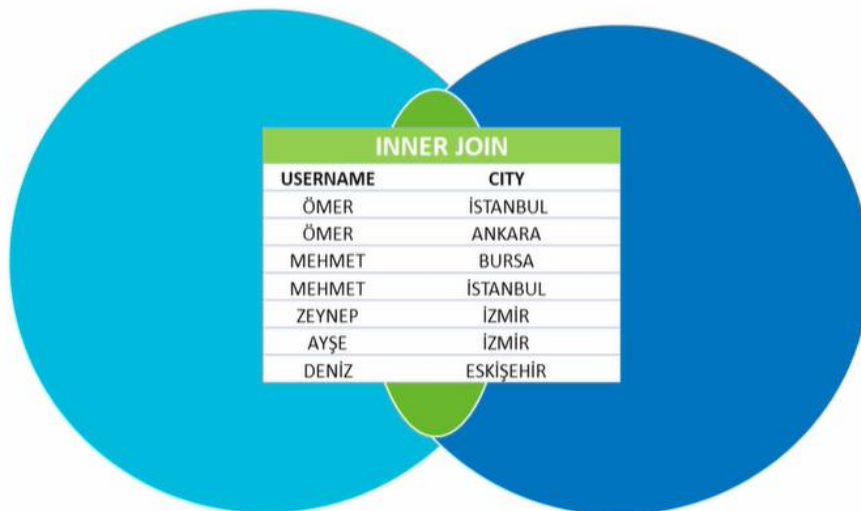
```
JOIN ITEMS I ON N.ID =I.ITEMCODE
```

**JOIN = INNER JOIN**

# JOIN TÜRLERİ

**1) INNER JOIN = İki kümenin kesişimi**

USERS	
ID	USERNAME
1	ÖMER
2	MEHMET
3	ZEYNEP
4	AYŞE
5	DENİZ
6	KADİR
7	ECRA

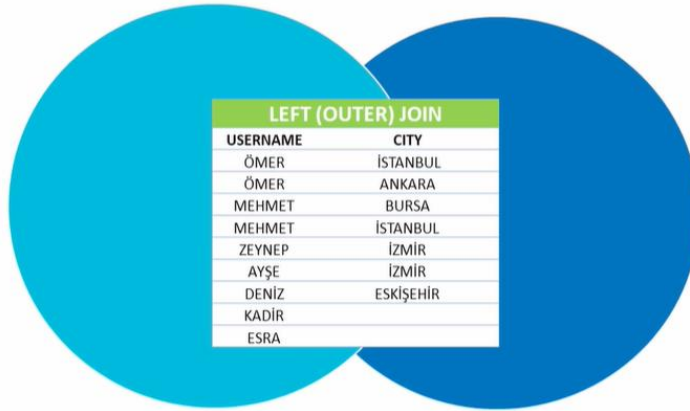


ADDRESS		
ID	USERID	CITY
1	1	İSTANBUL
2	1	ANKARA
3	2	BURSA
4	2	İSTANBUL
5	3	İZMİR
6	4	İZMİR
7	5	ESKİŞEHİR
8	8	KONYA
9	10	ADANA



## 2) LEFT JOIN (OUTER JOIN) = Sol taraftaki tablonun tamamını ve kesişimlerini getirir

USERS	
ID	USERNAME
1	ÖMER
2	MEHMET
3	ZEYNEP
4	AYŞE
5	DENİZ
6	KADİR
7	ESRA



LEFT (OUTER) JOIN	
USERNAME	CITY
ÖMER	İSTANBUL
ÖMER	ANKARA
MEHMET	BURSA
MEHMET	İSTANBUL
ZEYNEP	İZMİR
AYŞE	İZMİR
DENİZ	ESKİŞEHİR
KADİR	
ESRA	

ADDRESS		
ID	USERID	CITY
1	1	İSTANBUL
2	1	ANKARA
3	2	BURSA
4	2	İSTANBUL
5	3	İZMİR
6	4	İZMİR
7	5	ESKİŞEHİR
8	8	KONYA
9	10	ADANA

## 2) RIGHT JOIN (OUTER JOIN) = Sağ taraftaki tablonun tamamını ve kesişimlerini getirir

USERS	
ID	USERNAME
1	ÖMER
2	MEHMET
3	ZEYNEP
4	AYŞE
5	DENİZ
6	KADİR
7	ESRA



RIGHT (OUTER) JOIN	
USERNAME	CITY
ÖMER	İSTANBUL
ÖMER	ANKARA
MEHMET	BURSA
MEHMET	İSTANBUL
ZEYNEP	İZMİR
AYŞE	İZMİR
DENİZ	ESKİŞEHİR
	KONYA
	ADANA

ADDRESS		
ID	USERID	CITY
1	1	İSTANBUL
2	1	ANKARA
3	2	BURSA
4	2	İSTANBUL
5	3	İZMİR
6	4	İZMİR
7	5	ESKİŞEHİR
8	8	KONYA
9	10	ADANA

## 2) FULL JOIN = Her iki tablonun da tamamını getirir

USERS	
ID	USERNAME
1	ÖMER
2	MEHMET
3	ZEYNEP
4	AYŞE
5	DENİZ
6	KADİR
7	ESRA



FULL JOIN	
USERNAME	CITY
ÖMER	İSTANBUL
ÖMER	ANKARA
MEHMET	BURSA
MEHMET	İSTANBUL
ZEYNEP	İZMİR
AYŞE	İZMİR
DENİZ	ESKİŞEHİR
KADİR	
ESRA	
	KONYA
	ADANA

ADDRESS		
ID	USERID	CITY
1	1	İSTANBUL
2	1	ANKARA
3	2	BURSA
4	2	İSTANBUL
5	3	İZMİR
6	4	İZMİR
7	5	ESKİŞEHİR
8	8	KONYA
9	10	ADANA

# JOIN TÜRLERİ UYGULAMALARI

## ÖRNEK:

```
SELECT U.ID , U.USERNAME , A.ADDRESSTEXT
FROM USERS U JOIN ADDRESS A   U.ID = A.USERID
```

```
SELECT U.ID , U.USERNAME , A.ADDRESSTEXT
FROM USERS U INNER JOIN ADDRESS A   U.ID = A.USERID
```

(Sadece kesişimler geldi)

```
SELECT * FROM USERS
SELECT * FROM ADDRESS
```

ID	USERNAME_	ADDRESSTEXT
1	ÖMER	İSTANBUL
2	MEHMET	ANKARA
3	ZEYNEP	BURSA
4	AYŞE	İSTANBUL
5	DENİZ	İZMİR
6	KADİR	İZMİR
7	ESRA	ESKİŞEHİR
8		KONYA

```
SELECT U.ID,U.USERNAME_,A.ADDRESSTEXT
FROM USERS U JOIN ADDRESS A ON U.ID=A.USERID
```

ID	USERNAME_	ADDRESSTEXT
1	ÖMER	İSTANBUL
1	ÖMER	ANKARA
2	MEHMET	BURSA
2	MEHMET	İSTANBUL
3	ZEYNEP	İZMİR
4	AYŞE	İZMİR
5	DENİZ	ESKİŞEHİR

```
SELECT U.ID,U.USERNAME_,A.ADDRESSTEXT
FROM USERS U INNER JOIN ADDRESS A ON U.ID=A.USERID
```

ID	USERNAME_	ADDRESSTEXT
1	ÖMER	İSTANBUL
1	ÖMER	ANKARA
2	MEHMET	BURSA
2	MEHMET	İSTANBUL
3	ZEYNEP	İZMİR
4	AYŞE	İZMİR
5	DENİZ	ESKİŞEHİR

```
SELECT U.ID , U.USERNAME , A.ADDRESSTEXT
FROM USERS U FULL JOIN ADDRESS A   U.ID = A.USERID
```

```
SELECT U.ID,U.USERNAME_,A.ADDRESSTEXT
FROM USERS U FULL JOIN ADDRESS A ON U.ID=A.USERID
```

ID	USERNAME_	ADDRESSTEXT
1	ÖMER	İSTANBUL
1	ÖMER	ANKARA
2	MEHMET	BURSA
2	MEHMET	İSTANBUL
3	ZEYNEP	İZMİR
4	AYŞE	İZMİR
5	DENİZ	ESKİŞEHİR
6	KADİR	NULL
7	ESRA	NULL
NULL	NULL	KONYA
NULL	NULL	ADANA



```
SELECT U.ID , U.USERNAME , A.ADDRESSTEXT
FROM USERS U LEFT JOIN ADDRESS A  U.ID = A.USERID
```

(Sol tarafındaki tablonu tamamı ve kesişimleri gelir)

**NOT:** JOIN'in sol tarafında kalan LEFT tablosu sağ tarafında kalan RIGHT tablosu

```
SELECT U.ID,U.USERNAME_,A.ADDRESSTEXT
FROM USERS U LEFT JOIN ADDRESS A ON U.ID=A.USERID
```

ID	USERNAME_	ADDRESSTEXT
1	ÖMER	İSTANBUL
1	ÖMER	ANKARA
2	MEHMET	BURSA
2	MEHMET	İSTANBUL
3	ZEYNEP	İZMİR
4	AYŞE	İZMİR
5	DENİZ	ESKİŞEHİR
6	KADİR	NULL
7	ESRA	NULL

```
SELECT U.ID , U.USERNAME , A.ADDRESSTEXT
FROM USERS U RIGHT JOIN ADDRESS A  U.ID = A.USERID
```

(Sağ tarafındaki tablonu tamamı ve kesişimleri gelir)

```
SELECT U.ID,U.USERNAME_,A.ADDRESSTEXT
FROM USERS U RIGHT JOIN ADDRESS A ON U.ID=A.USERID
```

ID	USERNAME_	ADDRESSTEXT
1	ÖMER	İSTANBUL
1	ÖMER	ANKARA
2	MEHMET	BURSA
2	MEHMET	İSTANBUL
3	ZEYNEP	İZMİR
4	AYŞE	İZMİR
5	DENİZ	ESKİŞEHİR
NULL	NULL	KONYA
NULL	NULL	ADANA

# STRING İŞLEMLERİ

1) **ASCII** = İçerisinde aldığı karakterin ASCII türünden değerini verir

`SELECT ASCII ('A') = 65`    `SELECT ASCII ('B') = 65`    `SELECT ASCII ('2') = 65`

SELECT ASCII ('A')

(No column name)
65

SELECT ASCII ('B')

(No column name)
66

SELECT ASCII ('2')

(No column name)
50

2) **CHAR** = İçerisine aldığı değer karşılığını verir yani ASCII'nın tersidir.

`SELECT CHAR ('65') = A`    `SELECT CHAR ('66') = B`    `SELECT CHAR ('50') = 2`

SELECT CHAR ('65')

(No column name)
A

SELECT CHAR ('66')

(No column name)
B

SELECT CHAR ('50')

(No column name)
2

2) **SUBSTRING** = Bir stringin içerisinde belli bir noktadan belli bir noktaya olan alanı almamızı sağlar.

`SELECT SUBSTRING ('SALİH TORAMAN',1,4)`

SELECT SUBSTRING ('SALİH TORAMAN',1,4)

(No column name)
SALI

1.karakterden itibaren 4 tane alır

`SELECT * FROM CITIES WHERE SUBSTRING (CITY,LEN(CITY),1)='S'`

SELECT \* FROM CITIES WHERE SUBSTRING (CITY,LEN(CITY),1)='S'

ID	COUNTRYID	CITY
13	1	BİTLİS
36	1	KARS
58	1	SİVAS


Son karakteri S ile bitenleri getirir

**4) CHARINDEX** = Bir stringin içerisinde başka bir stringi buldurup onun pozisyonun bize söyler.

```
SELECT CHARINDEX ('H','SALİH TORAMAN',1)
```

H harfini SALİH TORAMAN'IN içerisinde 1'den başlayarak ara

```
SELECT CHARINDEX ('H','SALİH TORAMAN',1)
```



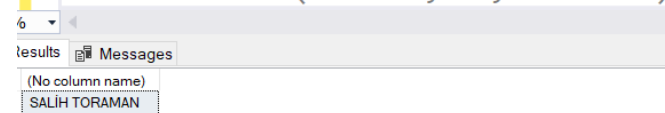
(No column name)
5

**5) CONCAT**= İki veya daha fazla stringin yanyana birleştirilmesini sağlar fakat normal şartlarda string birleştirme işlemini yanyana yazarken + işaretini de kullanabiliriz.

```
SELECT CONCAT ('SALİH',' ','TORAMAN')
```

```
SELECT 'SALİH' + 'TORAMAN'
```

```
SELECT CONCAT ('SALİH',' ','TORAMAN')
```



(No column name)
SALİH TORAMAN

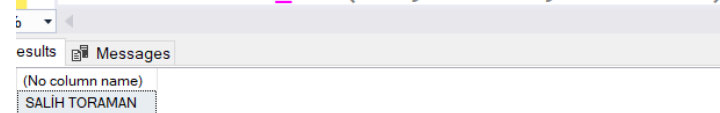
```
SELECT 'SALİH' + 'TORAMAN'
```



(No column name)
SALİHTORAMAN

```
SELECT CONCAT_WS (' ','SALİH','TORAMAN')
```

```
SELECT CONCAT_WS (' ','SALİH','TORAMAN')
```



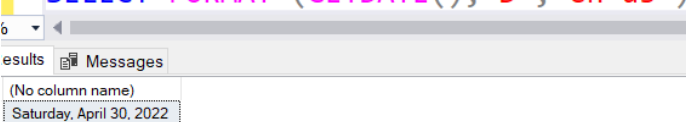
(No column name)
SALİH TORAMAN

**6) FORMAT** = Sayı ya da tarih değerlerini istediğimiz dile uygun şekilde yazdırmamızı sağlar.

```
SELECT FORMAT (GETDATE() , 'D' , 'en-us')
```

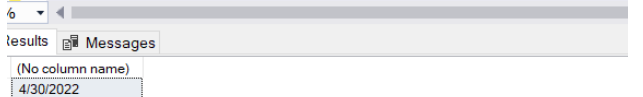
```
SELECT FORMAT (GETDATE() , 'd' , 'en-us')
```

```
SELECT FORMAT (GETDATE() , 'D' , 'en-us')
```



(No column name)
Saturday, April 30, 2022

```
SELECT FORMAT (GETDATE() , 'd' , 'en-us')
```

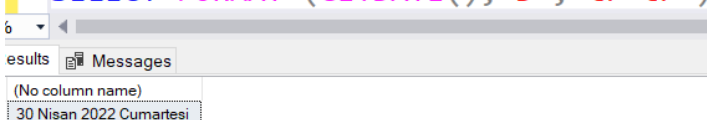


(No column name)
4/30/2022

```
SELECT FORMAT (GETDATE() , 'D' , 'tr-tr')
```


```
SELECT FORMAT (GETDATE() , 'd' , 'tr-tr')
```

```
SELECT FORMAT (GETDATE() , 'D' , 'tr-tr')
```



(No column name)
30 Nisan 2022 Cumartesi

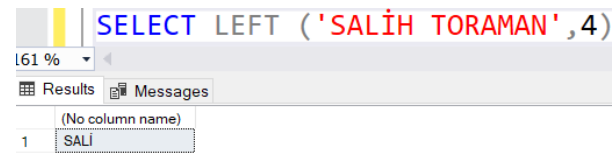
```
SELECT FORMAT (GETDATE() , 'd' , 'tr-tr')
```



(No column name)
30.04.2022

**7) LEFT,RIGHT,LEN** = LEFT soldan , RIGHT sağdan istediğimiz kadar karakter almamızı sağlar. LEN ise stringin uzunluğunu verir.

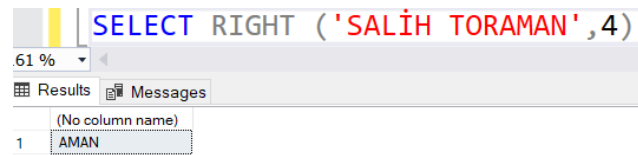
```
SELECT LEFT ('SALİH TORAMAN',4)
SELECT LEFT ('SALİH TORAMAN',4)
```



The screenshot shows a SQL query editor with the query `SELECT LEFT ('SALİH TORAMAN',4)` and its results. The results table has one row with the value 'SALI'.

(No column name)
SALI

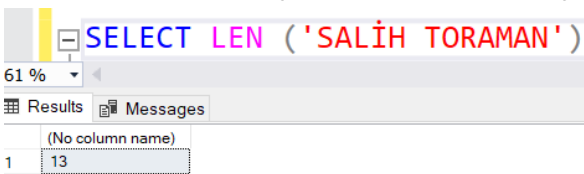
```
SELECT RIGHT('SALİH TORAMAN',4)
SELECT RIGHT ('SALİH TORAMAN',4)
```



The screenshot shows a SQL query editor with the query `SELECT RIGHT ('SALİH TORAMAN',4)` and its results. The results table has one row with the value 'AMAN'.

(No column name)
AMAN

```
SELECT LEN ('SALİH TORAMAN')
SELECT LEN ('SALİH TORAMAN')
```

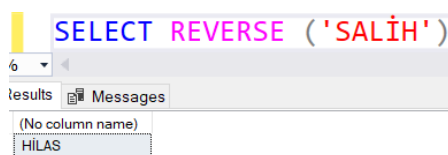


The screenshot shows a SQL query editor with the query `SELECT LEN ('SALİH TORAMAN')` and its results. The results table has one row with the value '13'.

(No column name)
13

**8) REVERSE,REPLICATE,LOWER,UPPER** = REVERSE tersten, REPLICAT tekrar ettirir. LOWER küçük harf, UPPER ise büyük harf yapar.

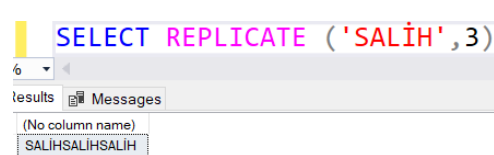
```
SELECT REVERSE ('SALİH')
SELECT REVERSE ('SALİH')
```



The screenshot shows a SQL query editor with the query `SELECT REVERSE ('SALİH')` and its results. The results table has one row with the value 'HİLİS'.

(No column name)
HİLİS

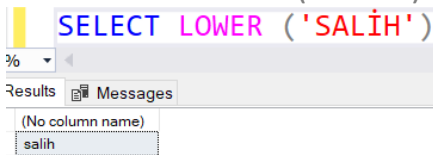
```
SELECT REPLICATE ('SALİH',3)
SELECT REPLICATE ('SALİH',3)
```



The screenshot shows a SQL query editor with the query `SELECT REPLICATE ('SALİH',3)` and its results. The results table has one row with the value 'SALİHSALİHSALİH'.

(No column name)
SALİHSALİHSALİH

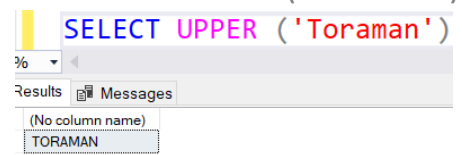
```
SELECT LOWER ('SALİH')
SELECT LOWER ('SALİH')
```



The screenshot shows a SQL query editor with the query `SELECT LOWER ('SALİH')` and its results. The results table has one row with the value 'salih'.

(No column name)
salih

```
SELECT UPPER ('Toraman')
SELECT UPPER ('Toraman')
```



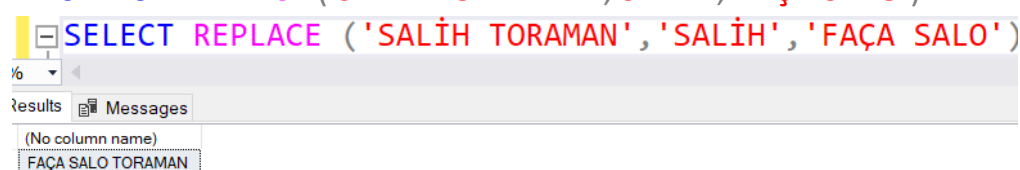
The screenshot shows a SQL query editor with the query `SELECT UPPER ('Toraman')` and its results. The results table has one row with the value 'TORAMAN'.

(No column name)
TORAMAN

**9) TRIM,LTRIM,RTRIM** = TRIM boşlukları temizler. Soldaki boşlukları silmek için LTRIM, Sağdakileri silmek için RTRIM

**10) REPLACE** = Bir şey ile bir şeyi yer değiştiren fonksiyon

```
SELECT REPLACE ('SALİH TORAMAN','SALİH','FAÇA SALO')
SELECT REPLACE ('SALİH TORAMAN','SALİH','FAÇA SALO')
```



The screenshot shows a SQL query editor with the query `SELECT REPLACE ('SALİH TORAMAN','SALİH','FAÇA SALO')` and its results. The results table has one row with the value 'FAÇA SALO TORAMAN'.

(No column name)
FAÇA SALO TORAMAN

# SİSTEM DATABASE'LERİ

## 1) MASTER DB

- Sistem Konfigürasyonu
- Kullanıcılar
- Veritabanları
- Sistem Dosyaları
- Collation Bilgisi

gibi SQL Server sisteminin temel konfigürasyon bilgilerini tutar.

## 2) MODEL DB

- Şablon veritabanıdır.
- Her oluşturulacak veritabanı ModelDB'nin bir kopyası olarak oluşturulur
- Her veritabanında otomatik olarak olmasını istediğimiz tipler, tablolar, fonksiyonlar vs varsa bu veritabanının içine konulabilir.

## 3) MSDB

- SQL Server Agent servisinin kullanıldığı veritabanıdır.
- Periyodik olarak çalıştırılan her türlü işlem (Job,Scheduler,Alert'ler gibi) burada tutulur.

## 4) TEMP DB

- Geçici tabloların oluşturulduğu işlemler burada gerçekleşir.
- Geçici bilgilerin saklandığı standart veritabanıdır.
- Kullanıcı veritabanlarından çekilen dataları group by, order by, sum, count, max, min gibi komutlarla özetlemek amacıyla kullandığımız aggregation işlemleri için de TEMP DB kullanılır.
- # ile başlayan geçici tablo oluşturma
- ## ile başlayan geçici tablo oluşturma

## TSQL KODLARI

**CREATE DATABASE** TEST (TEST adında yeni bir databases oluşturur)

**DROP DATABASE** TEST (TEST database'ini komple siler)

**CREATE TABLE** ISIMLER (ID **INT IDENTITY**(1,1), ISIM **VARCHAR**(20))

**INSERT INTO** ISIMLER (ISIM) **VALUES** ('ÖMER')

(TEST database'ini açtıktan sonra ISIMLER tablosu oluşturdu ve ISIMLER tablosuna ISIM kolonunu ekledi)

(ISIMLER tablosunun ISIM kısmına ÖMER ismini ekledi)

**ALTER TABLE** ISIMLER **ADD** TELEFON **VARCHAR**(12)

(ISIMLER tablosuna TELEFON kolonunu ekledi)

**ALTER TABLE** ISIMLER **ALTER COLUMN** TELEFON **VARCHAR**(16)

(VARCHAR(12) yi VARCHAR(16) olarak değiştirdi)

**DROP TABLE** ISIMLER

(ISIMLER tablosu tamamen silinir)

**ALTER TABLE** ISIMLER **DROP COLUMN** TELEFON

(ISIMLER tablosunda TELEFON kolonu tamamen silinir)

**DECLARE** @ISIM **AS VARCHAR**(100) yazarak değişken tanımladı

**DECLARE** @ISIM **AS VARCHAR**(100)

**SELECT** 'SALIH' yazarsan 1.satır SALIH olarak gelir

**DECLARE** @SAYI **AS INTEGER**

**SELECT** '15' yazarsan 1.satır 15 olarak gelir

**DECLARE** @SAYI **AS INTEGER**

**SET** @SAYI=15

**DECLARE** @SAYI2 **AS INTEGER**

**SET** @SAYI2=30

**SELECT** @SAYI **AS** SAYI1 , @SAYI2 **AS** SAYI2 ,

yazarsan 1.satır 15 , 2.satır 30 olarak gelir ve kolonlar SAYI1 ve SAYI2

```
DECLARE @SAYI AS INTEGER
SET @SAYI=15
DECLARE @SAYI2 AS INTEGER
SET @SAYI2=30
DECLARE @TOPLAM AS INTEGER
SET @TOPLAM=@SAYI + @SAYI2
SELECT @SAYI AS SAYI1 , @SAYI2 AS SAYI2 , @TOPLAM AS TOPLAM
(yazarsan yukarıdakine ek olarak 3.kolonda ikisinin toplamı gelir)
```

```
DECLARE @KELIME AS VARCHAR(100)
SET @KELIME='MERHABA DÜNYA'
SELECT SUBSTRING(@KELIME,1,7) = MERHABA
(gelir çünkü 1 den başlar 7 harf getirir)    Not : boşluklarıda sayar
```

```
DECLARE @KELIME AS VARCHAR(100)
SET @KELIME='MERHABA DÜNYA'
SELECT SUBSTRING(@KELIME,9,5) = DÜNYA
(gelir çünkü 9 dan başlar 5 harf getirir)
```

```
DECLARE @KELIME AS VARCHAR(100)
SET @KELIME='MERHABA DÜNYA'
SELECT LOWER (@KELIME)
(string ifadeyi tamamen küçük harfli yapar yani "merhaba dünya" olur)
```

```
DECLARE @KELIME AS VARCHAR(100)
SET @KELIME='merhaba dünya'
SELECT UPPER(@KELIME)
(string ifadeyi tamamen büyük harfli yapar yani "MERHABA DÜNYA" olur)
```

```
DECLARE @AD AS VARCHAR(100)='SALIH'
DECLARE @SOYAD AS VARCHAR(100)='TORAMAN'
SELECT @AD+ ' '+@SOYAD = SALIH TORAMAN    (boşluk var)
```

```
DECLARE @KELIME AS VARCHAR(100)
SET @KELIME='MERHABA DÜNYA'
SELECT LEFT (@KELIME,7) = MERHABA
```

```
DECLARE @KELIME AS VARCHAR(100)
SET @KELIME='MERHABA DÜNYA'
SELECT RIGHT (@KELIME,5) = DÜNYA
```

```
DECLARE @KELIME AS VARCHAR(100)= ' MERHABA DÜNYA '
SELECT LTRIM (@KELIME) =MERHABA DÜNYA
(soldaki boşlukları temizler)
```

```
DECLARE @KELIME AS VARCHAR(100)= ' MERHABA DÜNYA '
SELECT RTRIM (@KELIME) = MERHABA DÜNYA
(sağdaki boşlukları temizler)
```

```
DECLARE @KELIME AS VARCHAR(100)= ' MERHABA DÜNYA '
SELECT TRIM (@KELIME)=MERHABA DÜNYA
(hem soldaki hem sağdaki boşlukları temizler)
```

```
DECLARE @KELIME AS VARCHAR(100)= 'MERHABA DÜNYA'
SELECT REPLACE ('MERHABA DÜNYA' , 'D' , 'G') = MERHABA GÜNYA
(içerisindeki D harflerini G ile değiştirir)
```

```
DECLARE @KELIME AS VARCHAR(100)= 'MERHABA DÜNYA'
SELECT REPLACE (@KELIME,'MERHABA' , 'HELLO') = HELLO DÜNYA
(içerisindeki MERHABA yı HELLO ile değiştirdi)
```

```
DECLARE @KELIME AS VARCHAR(100)= 'MERHABA DÜNYA'
SELECT LEN(@KELIME) = 13
(kelimenin uzunluğunu gösterir)
```

```
DECLARE @KELIME AS VARCHAR(100)= 'MERHABA DÜNYA'
SELECT LEN('MERHABA DÜNYA') = 13
(kelimenin kendisinde yazabilirsin, kelimenin uzunluğunu getirdin)
```



**NOT= @KELIME yazmak yerine MERHABA DÜNYA olarak da yani kelimenin kendisini de yazabilirsin.**

```
DECLARE @KELIME AS VARCHAR(100)= 'MERHABA DÜNYA'  
SELECT * FROM string_split(@KELIME,' ') = MERHABA  
DÜNYA
```

(string ifadeyi içinde belirli bir ayrıca göre parçalar ve tablo haline çevirir)

(buradaki ayrıç “boşluk” tu yani boşluğa göre çevirdi)

```
DECLARE @TARIH AS DATETIME  
SET @TARIH='2019-01-01 16:35:28'  
SELECT @TARIH = 2019-01-01 16:35:28
```

```
DECLARE @TARIH AS DATETIME  
SET @TARIH='2019-01-01 16:35:28'  
DECLARE @TARIH2 AS DATETIME  
SET @TARIH2= DATEADD (MONTH,6,@TARIH)  
SELECT @TARIH, @TARIH2 = 2019-01-01 16:35:28 ve 2019-01-01  
16:35:28  
(ay eklicem, 6 ay eklemek istiyorum,@TARIH değişkenime eklicem)
```

```
DECLARE @TARIH AS DATETIME  
SET @TARIH = '2019-01-01 16:35:28'  
DECLARE @TARIH2 AS DATETIME  
SET @TARIH2= DATEADD (DAY,8,@TARIH)  
SELECT @TARIH, @TARIH2 = 2019-01-01 16:35:28 ve 2019-01-09  
16:35:28
```

```
DECLARE @TARİH AS DATETIME
SET @TARİH = '2019-01-01 16:35:28'
DECLARE @TARİH2 AS DATETIME
SET @TARİH2 = '2020-01-01 16:35:28'
SELECT DATEDIFF (YEAR, @TARİH, @TARİH2) = 1
(@TARİH ve @TARİH2 arasındaki yıl farkını getirir)
```

```
DECLARE @TARİH AS DATETIME
SET @TARİH = '2019-01-01 16:35:28'
DECLARE @TARİH2 AS DATETIME
SET @TARİH2 = '2020-01-01 16:35:28'
SELECT DATEDIFF (WEEK, @TARİH, @TARİH2) = 52
(@TARİH ve @TARİH2 arasındaki hafta farkını getirir)
```

```
SELECT DATEDIFF (DAY, '2019-01-01' , '2020-01-01' ) = 365
SELECT DATEFROMPARTS (2020,8,21) = 2020-08-21
```

```
SELECT GETDATE() = 2022-07-01 21:43:37
(anlık tarih ve saat bilgisini getirir)
```

```
DECLARE @TARİH AS DATE
SET @TARİH = '1998-04-25'
SELECT DATEDIFF (YEAR, @TARİH, GETDATE()) = 24
(tarihe göre kişinin güncel yaşını hesaplar)
```

```
DECLARE @TARİH AS DATETIME
SET @TARİH = '2019-01-01 16:35:28'
SELECT DATEPART (HOUR, @TARİH) = 16
(stringimizdeki yani @TARİH kısmında istediğimiz yeri getirir)
```

```
DECLARE @TARİH AS DATETIME
SET @TARİH = '2019-01-01 16:35:28'
SELECT DATEPART (MINUTE, @TARİH) = 35
(stringimizdeki yani @TARİH kısmında istediğimiz yeri getirir)
```

## SQL SERVER'DA INDEX KAVRAMI

- Veritabanı yönetim sistemlerinin temel misyonu veriyi yönetmektir.
- Genel ortalamaya baktığımızda Veritabanı Yönetim Sistemlerinin %95 oranında okuma, %5 oranında yazma yaptığını görüyoruz.
- Bu durumda okuma performansını arttıracak her türlü işlem veritabanı sunucunun performansının arttırılmasında doğrudan etkili olacaktır.
- Bir tabloda birden fazla Clustered Index bulunabilir
- Bir tabloda yalnız 1 adet Clustered Index bulunabilir
- Tablonun fiziken sıralı hali Clustered Index'e göredir.
- Primary Key olarak işaretlenen alanlar Clustered Index'i oluşturur.
- Clustered Index'ler Unique (Tekil) dir ve tekrar etmez.

**Index Rebuild** = Bozulan indexleri yeniden sıralamaktır.

**Index Fragmentation** = Index page'lerdeki datanın tamamının sıralı olmasıdır.

## Trigger

- Veritabanı tablosunda bir işlem gerçekleştiğinde başka işlemin otomatik olarak gerçekleşmesi anlamına gelir.
- Burada işlem olarak kastedilen data manipölasyonudur.
- Data manipölasyonları ise Insert, Update ve Delete işlemleridir.
- Yazılan triggerlar Insert, Update ve Delete işlemlerinden sonra otomatik çalışan yapılardır.
- Triggerların içinde sanal olarak oluşan Inserted ve Deleted tabloları vardır
- Inserted tablosu yeni eklenen kaydın ya da update edildiğinde değişen kaydın yeni değerini tutar.
- Deleted tablosu ise silinen kaydı ya da değişen kaydın eski değerini tutar.
- Trigger'lar genelde otomatik toplam hesaplama, son değeri alma ya da loglama amacı ile kullanılır.

# Stored Procedure

-T-SQL cümlelerinin SQL Server'ın hafızasına kaydedilerek derlendiği ve derlenmiş hallerinin çalıştırıldığı yapılardır.

## Faydaları

- Execution plan çıkarıldığı ve derlendiği için ilk 4 aşamayı atlar ve daha hızlı çalışır.
- Client Server mimarisinin aksine toplu işlemler kendi içerisinde çalıştığı için sorgular network hızında değil ram hızında çalışır.
- Yazılım güncellemeden değişiklikler yapılabilir yani kolay güncellenir.
- SQL Injection saldırılarına karşı kesin çözümdür yani güvenlidir.
- Güvenlidir, kritik raporlar için Stored Procure bazında yetki verilebilir.
- Herhangi bir programlama dilinde yazılabilecek hemen hemen her türlü komut burada yazılıp çalıştırılabilir yani yeteneklidir.
- Stored Procedure'ler birbiri içerisinde çağırılabilir yani esnektir.
- Performansı ölçülebilir. Kaç kez çalıştırılmış en son ne zaman çalıştırılmış gibi bilgiler görülebilir.
- Oluşturulduğu anda derlendiği için ADHoc Query'lere göre daha hızlıdır.
- Bir procedure içinden başka bir procedure de çağırılabilir.
- Bir çok sorguyu birleştirerek tek seferde tek isimle çağırmamızı sağlar.

## View'lar

- Uzun SQL sorgularını tek bir komut altında çağırmamızı sağlar.
- Kolon isimlerini Türkçeleştirme imkanı verir.
- View'lar içinde select, insert, update, delete işlemleri kullanılır.
- Birden fazla tablonun join yapılarak tek tablodan çekiliyor gibi çekilmesini sağlar.
- Select \* From View\_Name şeklinde kullanılır.
- Viewlar parametre almazlar.

# YEDEKLEME YÖNTEMLERİ

## FULL BACKUP

- Tüm veritabanının yedeğinin alınması işlemidir.
- Sistem çalışırken online olarak alınabilir
- Sıkıştırma parametresi ile %95'lere kadar sıkıştırılarak alınabilir.
- Sadece backup alınan zamana dönülebilir.

## DIFFERENTIAL BACKUP

- Differential backup son alınan full backup ile fark yedeğidir.
- Full backup'a göre daha az yer kaplar.
- Sistem çalışırken online olarak alınabilir.
- Sıkıştırma parametresi ile %95'lere kadar sıkıştırılarak alınabilir.
- Sadece backup alınan zamana dönülebilir.

## TRANSACTION LOG BACKUP

- Transaction log dosyasının yedeklenmesidir.
- En son alınan backup hangisi ise (Full, Differential ya da Transaction Log farketmez) onunla arasındaki değişimi alır.
- Sistem çalışırken online olarak alınabilir.
- Sıkıştırma parametresi ile %95'lere kadar sıkıştırılarak alınabilir.
- Saniyelik olarak istediğimiz ana dönme imkanı verir.
- MDF dosyada data, LDF dosyada Transaction Log tutulur.

## SQL SERVER AGENT KAVRAMI

-SQL Server'da otomatik ve periyodik olarak çalışmasını istediğimiz işlemler için kullandığımız yapıdır. Bu yapılar şu şekildedir ;

Yedek alma	Mail gönderme
Index bakım planları	Otomatik raporlar oluşturma
Database shrink planları	Veri ambarı doldurma

- Geri planda SQL Server Agent hizmetini kullanır.
- Job, Schedule, Alert gibi kavramlar vardır.
- Tüm konfigürasyonunu MSDB sistem database'inde tutar.
- SQL Server Agent bir windows servisi.

# PERİYODİK BAKIM PLANLARI

## Index Bozulmaları :

- SQL Server'da yeni kayıtlar oluştuğunda, kayıtlar silindikçe ve güncellendikçe indexler bozulur. Bozulan indexleri düzeltmek için 2 işlemden biri yapılır.
- Index Rebuild ya da Index Reorganize işlemi yapılır.
- Index Rebuild** = Bozulan indexleri yeniden sıralamaktır.

## İstatistik Güncellemeleri :

- SQL Server'da sistem doğru indexi bulabilmek için istatistiklere bakar ve SP\_UPDATESTATS komutu ile çalıştırılır
- Yeni kayıtlar oluştuğunda, kayıtlar silindikçe ve güncellendikçe bir indexte hangi kaydın ne kadar olduğu (kayıt frekansı) değişir.
- İstatistikler ne kadar güncel ise sistem doğru indexi o kadar iyi bulur
- Her gün çalıştırılmalıdır.

## Database Shrink :

- SQL Server'da bir database'e yeni kayıt eklendikçe database büyür ama kayıt silindikçe küçülmez.
- Eğer büyüyen bir veritabanı küçültülmek isteniyorsa shrink işlemi gerçekleştirilir.
- Database'deki boşluk alanları temizlenerek DB boyutunun küçültülür.

## Database Backup :

- Bir veya birden fazla database'in yedeğinin alınma işlemidir.
- Full, Differential ya da Transaction Log backup olarak istenen şekilde otomatik olarak çalışacak backup işlemidir.

## Check Database Integrity :

- Database index ve table page'lerde herhangi bir sıkıntı olup olmadığını kontrol eder yani database'de bir bozukluk olup olmadığına bakar.

**Primary Key** = Bir tablodaki satırı tanımlayan anahtar kolon veya kolonlardır

**Primary Key Constraint** = SQL'de bir sütundaki tüm değerlerin farklı olmasını sağlar

**HOSTNAME** = SQL Server Profiler'da bir bilgisayarın hareketleri izlemek istenildiğinde buradan filtre verilir.

**HOST\_NAME()** = SQL Server'a bağlanan kullanıcının bilgisayarının adını getirmeye yarayan fonksiyondur.

**PROGRAM\_NAME()** = SQL Server'a hangi uygulamadan bağlanılmışsa onun adını getirmeye yarayan fonksiyondur.

**SUSER\_NAME()** = SQL Server'a bağlanan kullanıcının kullanıcı adını getirmeye yarayan fonksiyondur.

**ROW\_NUMBER()** = SQL sorgusuna satır numarası ekleyen fonksiyon.

**WITH (NOLOCK)** = Açık bir transactionda bulunan bir tablodan lock'a takılmadan veri okumak için kullanılır.

**COMMIT TRAN** = Açık bir transactionda yapılan tüm işleri tamamlayarak kaydedilmesini sağlayan komut.

**ROLLBACK TRAN** = Açılan bir transactionda yapılan tüm işleri geri alır

**NEWID()** = GUID türünde benzeri olmayan bir değer üretir.

**@@IDENTITY** = Tabloya yapılan son insert işleminden oluşan otomatik artan alan değeri gösterir.

**@@FETCH\_STATUS** = Cursor'de döngünün içinde ya da döngünün bittiği bilgisini tutar.

**Maintanance Clean Up Task** = Belli bir günden öncesine ait olan yedek dosyalarının silinmesini sağlar.

**SCHEDULE** = SQL Agent içinde bir takım işler için oluşturulan zaman dilimine verilen isim.

**Page Life Expectancy (PLE)** = Bir page'in ram de ortalama aldığı süreyi ifade eder.

**Fill Factor** = Tablodaki index page'lerin doluluk oranıdır.

**Fill Factor 70 olan bir tablo** = Bir tablodaki index page'lerin %70 i doludur.

**İstatistik** = Bir index'in içindeki alanları içeren satırların ne sıklıkta tekrar ettiği bilgisidir.

**SET STATISTICS IO ON** = Çalıştırılan sorguda ne kadar page okuma / yazma işlemi yapıldığı bilgisini görmemizi sağlar.

**OLTP** = Birden fazla veritabanı işleminin hatasız yerine getirilmesi işlemi olup, hata olduğunda geri alma mekanizmasına sahiptir.

**Average Disk Que Len** = Diskte sırada bekleyen ortalama işlem sayısı

**Raiseerror** = Kullanıcıya bizim tanımladığımız bir hata mesajı gönderir

**SQL Server Configuration Manager** = SQL Server servislerini yapılandırmamızı sağlar.

**SQL Server Failover Cluster** = İki sunucunun aynı storage üzerinde biri aktifken diğeri pasif şekilde çalışması ve felaket anında diğeri sunucunun devreye girmesi işlemidir.



**Maintenance Plan** = Otomatik ve periyodik olarak çalışacak şekilde sistemde index rebuild, shrink, istatistik güncelleme gibi işlemlerin yapılması işlemidir.

**Execution Plan** = Bir sorgu çalıştırılırken hangi tabloda hangi indexin kullanılacağıının belirlenmesi işlemidir.

**SQL Profiler** = Sistemde çalışan sorguları izlememizi sağlayan araçtır.

**SQL Server Database Engine** = Portu dinleyip gelen SQL komutlarını çalıştıran ve client bilgisayara sonucu döndüren SQL Server'ın ana hizmetidir.

**SQL Server Developer Edition ;**

- Ücretsiz bir sürümdür.
- SQL Server Enterprise Edition ile aynı özelliklere sahiptir
- Sadece geliştirme ortamında kullanılabilir.

**SQL Server 2019 Express Edition sürümü ;**

- Ücretsiz bir sürümdür
- Donanımsal olarak sınırlamaları vardır.
- Hem geliştirme ortamında hem de canlı ortamda kullanılabilir.
- Maksimum 1.4 GB ram kullanımına izin verilir.
- SQL Agent hizmeti de ücretsiz gelir.

SQL Server'da Trace başlatmak SQL Profiller ile çalışan sorguları izlemek demektir.

SQL Server'da donanım olarak Ram Bus hızı ne kadar arttırılırsa sistem performansı aynı oranda artar.

**Dataset**, SQL üzerinden temel crud işlemlerini uzun adonet sorgularına gerek kalmadan, bunları hazır olarak veren yapıdır.

Veri kümesi olarak çevrilebilir.

CRUD = Create – Read – Update – Delete

CRUD = Ekleme – Listeleme – Güncelleme – Silme

**Prosedür**, uzun SQL sorgularını tek kelimelik komutlara sığdıran yapılardır.

Programlama dillerindeki metotlara benzerler.

Creat komutu ile oluşturulurlar.

Execute komutu ile çağrılırlar.

