

## SQL

- Sql'de Fonksiyonların Üzerine Gelindiğinde kullanımını bize anlatır.
- += ekle demek.
- Sql'de virgün eş fonksiyonları 'cümle içinde' ayırmak için kullanılır, komut sonuna virgül olmaya gerek yoktur. Aksi taktirde kod çalışmaz ve tüm komutlar için hata alır.
- Messages kısmında select \* from'dan sonra affected 10 row. Gösterilen kayıtları ifade eder. Aynı zamanda, diğer işlemler için de belirtir.
- Sütunlar arasında direk a-b, axb,a/d şeklinde işlemler yapılabilir. Parantez kullanılarak birkaç işlem de yapılabilir.
- Sütun aynı zamanda 1.2 şeklinde çarpılabilir. Yüzde hesaplanabilir.
- Sql'de değişken isimleri @ sembolü ile tanımlanır.
- SELECT 500 %9, 9 ile bölümünden kalanı verir.
- Ek Hahbytes Fonksiyonu = Key yaratmak için her birine farklı değerler atar.
- Sorguyu son aşamada temp tabloya basarak almak performansı düşürür.

## Yeni Veritabanı Oluşturma ve Tabloları

- Sunucu altında, Database üstüne gelip, sağ tuşla new database denir. Database ismi (veritabanı ismi) yazılıp enter denilir.

- Tablo oluşturmak için tablolar sağ-klik new table denilir.
- Tablo isimleri verildi.
- Tablo datatypelar belirlenir.
- Control S denilip kaydedilir.
- Ana tablo ismi sorulduğunda giriş yapılır.
- Tables Klasörüne gelinip, Sağ klikle Edit Top 200 denilir, veri girişleri yapılır.

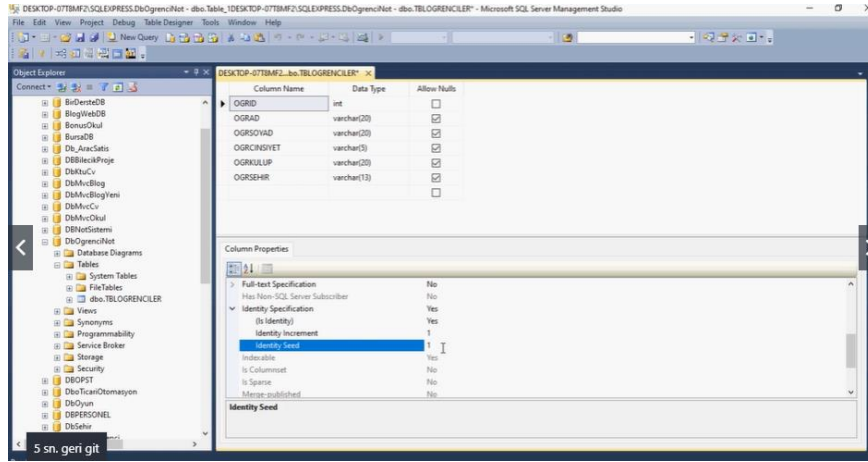
**Not:** Hizmetler> Sql Dosyasına sağ tuş yapıp çalışıp çalışmadığını kontrol edebilirsiniz.

**Not:** Sql'de sorgunun neresini seçersen orası çalışır. Tam satır olmasına gerek yok.

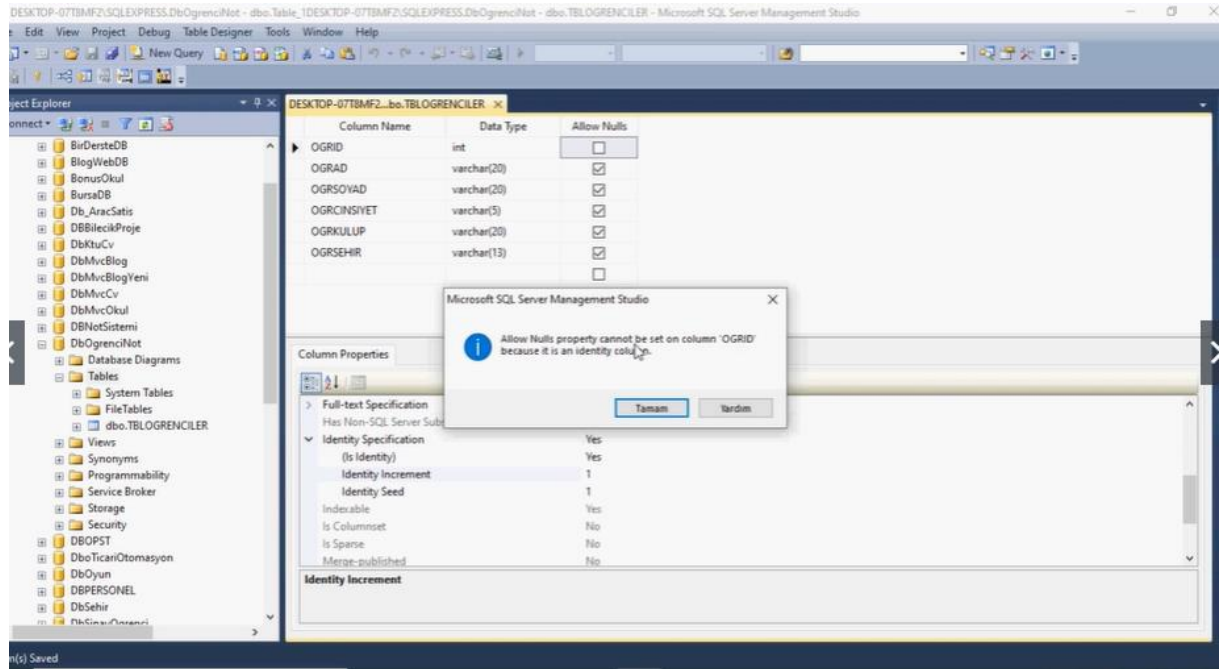
### Tablo Üzerinde Değişiklik Yapma

- Tablo açıldıktan sonra (sağ click design)
- Üst taraftan tool bölümünden options'a tıklanır.
- En alttan 3. Sırada Designers kısmından, Kutucuklu olan boş kısımda Prevent saving changes that require table tıklı kalırsa değişikliğe izin vermeyecek. Kaldırabiliriz.

### Identity Alanını Aktifleştirme



- Column Properties Kısımından (Is Identity) yes şeklinde seçilir.
- Identity Incresement: Kaçar kaçır artacağı seçilir
- Identity Seed: (Tohum) Başlangıç anlamına gelir.
- **Ctrl + S** ile kaydedildiğinde
- 1. Sutünün Allow Nulls kısmı otomatik olarak boş gelir.



## Tabloya Veri Girişi

- Tabloya sağ tıklayıp Edit Top 200 Rows tıklarsak, asıl veri tabanı olan kısım yani 200 veri gözükür.
- Kalan verileri sonraki sayfaya atacak.

- Ayarlar kısmından bu 200 ROWS kısmı değiştirilebilir.
- Bu sayfada sağ tık execute yaparsak bu sayfayı yeniler.
- Veri girişi yapıp alta indikçe Sql'de kaydolur.

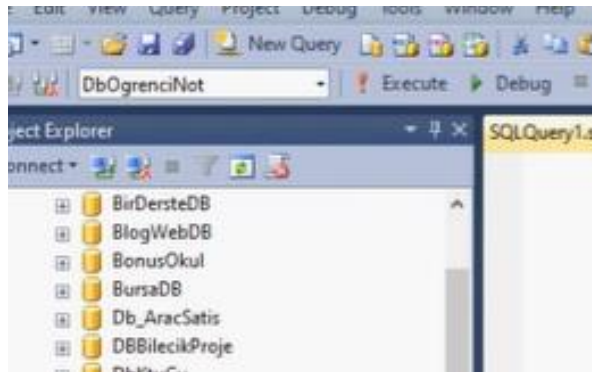
## DDL ve DML Komutları

(Data Defination Language- Data Tanımlama Dili)

Tablo Üzerinde Değişiklikler Yaparlar (Genel Sütunlar)

### DDL Komutları

- **Create** : Oluşturmak için
- **Alter** : Düzenlemek için
- **Drop** : Silmek için



- Sol üstteki alan veri tabanı seçili olan alandır. Çalışırken doğru databasede olduğumuzdan emin olmalıyız.

### Create Database

- Veri tabanı oluşturmak için kullanılır.

**USE:** Kullanılacak databasei seçer.

```
USE #burakk
```

**PRIMARY KEY:** Sutun isminden sonra hemen yazılır

```
CREATE TABLE Burak
```

```
( Ad, Tinyint IDENTITY (1,1) PRIMARY KEY,  
  Soyad, varchar
```

```
CREATE DATABASE #burakk
```

**IDENTITY:** Sutun isminden sonra hemen yazılır

```
CREATE TABLE Burak
```

```
( Ad, Tinyint IDENTITY (1,1) PRIMARY KEY,  
  Soyad, varchar
```

```
CREATE DATABASE #burakk
```

**CHECK:** Kontrol eder.

```
CREATE TABLE Burak
```

```
( Ad, Tinyint IDENTITY (1,1) PRIMARY KEY,  
  Soyad, varchar(50), CHECK(URUN>0)
```

**DEFAULT:** Verinin durumunu kontrol eder o şekilde getirir.

**CREATE TABLE** Burak

( Ad, Tinyint IDENTITY (1,1) PRIMARY KEY,  
Soyad, varchar(50), CHECK(URUN>0)  
Okul, bit DEFAULT '1')

## Create Table

- Tablo oluşturmak için farklı bir yöntemdir
- İlk tablo adı yazılır.
- Parantez içinde sütun isimleri ve veritürleri yazılır.

```
CREATE TABLE tblidersler  
(  
  DERSID TINYINT,  
  DERAD VARCHAR(20)  
)
```

- Çalıştırdıktan sonra yeni tablo oluşur. Table üstüne sağ tuş refresh denilince yeni tablo oluşur.
- Oluşan tablonun üstüne design dersek tabloyu düzenleyebiliriz.
- Aşağıda açılan column propertiesten. Identity Specification'dan YES'e tıklayarak DERSID otomatik artan haline getirilir.

## Alter Table

- Düzenleme işlemi için kullanılıyor.
- Örneğin bir tabloda bir sütunun veri tipini değiştirmek için kullanılabilir.
- Örneğin bir tabloda sütun yazmayı unuttuk, eklemek için kullanabiliriz.
- Parantez kullanmamıza gerek yoktur.

```
ALTER TABLE TBLDERSLER  
ADD KONTENJAN smallint
```

- Aynı zamanda silmek için de kullanabiliriz.
- Kolon silmek istiyorsak, COLUMN olarak eklemeliyiz.

```
ALTER TABLE TBLDERSLER  
DROP COLUMN KONTENJAN
```

### DROP TABLE

- Tablo silmek istiyorsak.

```
DROP TABL
```

## DML Komutları

### Data Manipulation Language – Veri İşleme Dili

### Veri Üzerinde Yapılan Değişiklikler (Özel Tek Veriler)

**Select:** Seçme listeleme işlemlerine yarar.

**Insert:** Ekleme işlemi için

**Update:** Güncelleme

**Delete:** Silme işlemi

### Select:

- En fazla kullanımı Select \* From Table
- \*ifadesi → ALL
- From → Den Dan

**Not:** Syntax 5 Rows Affected. (5 Kayıt Etkilendi)

### Where:

- Şartlı sorgulamalardır.
- Where şartı arama yaptığımız değerlerin tamamının olup olmadığını kontrol eder.
- Like ise harf ve ya rakam arayabilir.
- E-ticaret sitelerinde çokça görülür.
- Kelimeler tırnak içerisinde kullanılır.

**And:** (or)

- Operatördür istediğiniz kadar kullanabilirsiniz.

**Or:** (and)

- And operatörüyle aynı anlamdadır.

**Decimal (x,y)**

--x: Toplam Basamak Sayısı

--y: Virgülden Sonra Basamak Sayısı

-- Ortalama: 78,65

```
SELECT (SINAV1 + SINAV2 + SINAV3) / 3 FROM TBLNOTLAR
```

- Sütunlar toplanabilir
- Sütunlar bölünebilir.

```
SELECT (SINAV1 + SINAV2 + SINAV3) / 3 FROM  
TBLNOTLAR
```

- F5 ile bir kere daha çalıştırırsak, bir alt sıradaki öğrencinin sınavlarının ortalamasını verir.
- Yani select komutu, sütunları toplayıp ortalamasını aldığımızda, satır satır sıra ile giderek sonuç verir.

**Insert (INSERT INTO)**

- İçerisine ekle demek.
- Insert Into şeklinde kullanılır.
- Insert Into Table (Column1,Column2,Column3..)

Values (V1V2,V3),.....)



- Hiyerarşik bir atama sistemi mevcuttur. Column1 Value1 ile Column2 Value2 ile Column3 Value3 ile atanacak şekilde işlemler gerçekleşir.

```
CREATE TABLE #BURAK
(OGRAD VARCHAR(20), OGRSOYAD VARCHAR(20))
```

```
INSERT INTO #BURAK
(OGRAD)
```

```
VALUES
('Fener')
```

```
DROP TABLE #BURAK
```

```
SELECT * FROM #BURAK
```

Şeklinde ekler.

OGRAD	OGRSOYAD
Fener	NULL

- Birden çok veri eklerden sütunlar anlatarak gidilirse null olarak devam eder.

### Delete:

- Delete From Table tüm kayıtları siler. (Tüm Tablodaki)
- Eğer tek bir kayıt (veri) silmek istiyorsak where kullanarak şart yazmalıyız.

```
CREATE TABLE #BURAK
(OGRAD VARCHAR(20), OGRSOYAD VARCHAR(20))
```

```
INSERT INTO #BURAK
(OGRAD)
```

```
VALUES  
( 'Fener' )
```

```
DELETE FROM #BURAK WHERE OGRAD = 'fener'  
DROP TABLE #BURAK
```

```
SELECT * FROM #BURAK
```

- Sql'de yapılan bu silme işlemi verileri tamamen yok etmez pasif hale getirir çünkü bu veriler başka tablolarla ilişkilidir.
- İlişkisel tablolar zarar görmesin diye, hem de bellek performans ilişkisi olumsuz etkilenmesin diye. ID işlemi yeniden başlamaz, kalan ID numaralarının etkilenmesine izin verilmez.
- Yeni bir kayıt girerken ID 3 silindiğinde onu pasif hale getirir ve ID 4 olarak kayıt devam eder.
- Delete ile silinen ID'lerin sıralaması Design kısmından düzenlenebilir.
- Identity Specification'dan (Is Identity) özelliği no yapılır sıra girilir. Daha sonra tekrar aktif edip artan olsun denilir. Bu şekilde yapıldığında artarak devam eder.

### Truncate TABLE:

- TRUNCATE TABLE şeklinde kullanılır
- Syntax bu şekilde ve farklıdır
- Tüm verileri siler.
- Veriler geri getirilemez.
- Bu sebeple yedeklenerek çalışılmalıdır.

```
CREATE TABLE #BURAK  
(OGRAD VARCHAR(20), OGRSOYAD VARCHAR(20))
```

```
INSERT INTO #BURAK  
(OGRAD)
```

```
VALUES  
( 'Fener' )
```

```
DELETE FROM #BURAK WHERE OGRAD = 'fener'  
DROP TABLE #BURAK
```

```
SELECT * FROM #BURAK
```

```
TRUNCATE TABLE #BURAK
```

### Update:

- Syntax hem Delete hem Insert'e yakın
- Delete' Where ile yakın.
- Insert'e yapı olarak yakın. (Sıra sıra ekleme olarak)
- UPDATE Komutu yazarken kesinlikle where kullanılmalıdır.
- Eğer Where kullanılmazsa tüm sütunu verilen değerle yapar.
- Syntax:

```
CREATE TABLE #BURAK  
(OGRAD VARCHAR(20), OGRSOYAD VARCHAR(20))
```

```
INSERT INTO #BURAK  
(OGRAD)
```

```
VALUES  
( 'Fener' )
```

```
UPDATE #BURAK SET URUNAD = 'SU ISTICI' WHERE URUNAD =  
'KETTLE'
```

- Kettle'ı su ısıtıcıya dönüştürür.

```
UPDATE #BURAK SET OGRAD = 'FB' WHERE ID = 4
```

- ID 4 olanları FB'ye dönüştürür.

```
UPDATE TBLURUNLER SET URUNSATISFIYAT +=500 WHERE  
KATEGORIID= 1
```

- Kategori ID 1 olan ürünlerin fiyatını 500 arttırır.
- += ekle demek.

**SET:** Ayarla anlamında

- ID belirlenmemişse WHERE komutu çalışmaz, tablo ayarlarından (Design) ID belirlenmelidir.
- UPDATE Sorgusunda where kısmında eşitlikte verilebilir.
- Örn:

```
UPDATE #BURAK SET OGRAD = 'FB' WHERE Ortalama > 10
```

- Tablo çağırdıktan sonra diğer tablodan çağrılanları düşürmek için update ve alt sorgu kullanılır.

```
UPDATE TBLURUNLER SET URUNSTOK = URUNSTOK - (SELECT  
SUM(ADET) FROM TBLHAREKET WHERE URUN = 1) WHERE  
URUNID = 1
```

- Where kısıtı tüm sütundaki değerlerden çıkarmasını istemediğimiz için yazılır. Parantezden sonra yazılması işlem sırası için doğru..
- Çok fazla ürün için yapacak olsaydık döngü kullanmamız gerekirdi.

### Tablo Değerini Alt Sorgu ve Update İle Yazdırma:

```
UPDATE KASA SET TOPLAM = (SELECT SUM (TUTAR) FROM  
TBLHAREKET)
```

- KASA tablosu TBLHAREKET tablosunun sonucunu döndürür

## Gruplandırma ve Operatörler

**Hazır Fonksiyonlar:** Bazı aritmetik işlemler için uzun uzun formül yazmak yerine sadece fonksiyonu çağırarak işlemleri kolay yapmayı sağlar.

```
SELECT Fonksiyon(Sutun isim ve ya *) FROM Tablo
```

**Count:** Kayıt sayısını verir

```
SELECT COUNT(*) FROM #BURAK
```

(No column name)
1

```
SELECT COUNT(*) AS 'Toplam Kayıt' FROM #BURAK
```

Toplam Kayıt
1

```
SELECT COUNT(*) AS 'Toplam Kayıt' FROM #BURAK WHERE  
Number = 5
```

**Sum:**

```
SELECT SUM(Number), AVG(Number), FROM #BURAK
```

- Şeklinde beraber kullanılabilir.

**Avg:**

```
SELECT AVG(Number), AVG(Sınav2), FROM #BURAK
```

Min:

```
SELECT Min(number), Min(Sınav2), FROM #BURAK
```

Max:

```
SELECT MAX(number) FROM #BURAK
```

Gruplandırma (Group By):

Groupby> Having> Like

```
SELECT Sütun, COUNT(*) FROM Tablo GROUP BY Sütun  
(Aynı)
```

- Syntax: Select'in yanına sütun yazılmalıdır. Fonksiyonları yazdığımız şekilde yazarsak Gruplandırırken, sayar fakat grupları göstermez.
- Hem birkaç sütun çağrılıp hem de farklı fonksiyonlar kullanılırsa group by için hata verebilir. Hata da net bir şekilde yazıyor.

```
SELECT DEPOREF, COUNT(*) FROM warehouse..DiM_DEPO  
GROUP BY DEPOREF
```



```
SELECT DEPOTİP, COUNT(*) FROM warehouse..DİM_DEPO  
GROUP BY DEPOTİP
```

### Having Komutu: (WHERE)

- Gruplandırmadan sonra şart yazılması amacıyla kullanılır.
- Gruplandırma işleminin sonucunda bir şart yazmak istiyorsak where'le yazamıyoruz. Burda having devreye giriyor.

```
SELECT DEPOTİP, COUNT(*) FROM warehouse..DİM_DEPO  
GROUP BY DEPOTİP HAVING DEPOTİP = 2
```

- Fakat grupladıktan sonra harfe göre çağırma istiyorsak. Like komutu devreye giriyor.

### Like Komutu: (Öncesinde Where)

- Öncesinde where kullanılmalı.
- Where şartı arama yaptığımız değer in tamamının olup olmadığını kontrol eder.
- Like ise harf veya rakam arayabilir.
- Like'ın yanında AND komutu kullanılabilir.
- NOT LIKE olarakta kullanabilirsin.
- '%an%' şeklinde kullanılır.

```
SELECT * FROM warehouse..DİM_DEPO WHERE KOD LIKE  
'%a%' AND raporgrup = -1
```

### Distinct Komutu:

- Tekrarsız değer getirmektedir.
- Syntax:

```
SELECT DISTINCT Sütunadı FROM Tablo
```

```
SELECT DISTINCT sehiradi FROM warehouse..DİM_DEPO
```

```
SELECT COUNT(DISTINCT(sehiradi)) FROM  
warehouse..DİM_DEPO
```

- Count içerisine DISTINCT yazılabilir.
- DISTINCT parantezsiz de kullanılabilir.

```
SELECT COUNT(DISTINCT sehiradi ) FROM  
warehouse..DİM_DEPO
```

### Order By Komutu: (Sıralama: A-Z ve Z-A)

- Sıralama işlemleri yapmamızı sağlar.
- İki şekilde sıralar
- A'dan Z'ye, asc
- Z'den A'ya. desc
- Gruplandırma işlemlerinden sonra kullanılır.
- Distinct komutunda çokça kullanılır.

```
SELECT * FROM Warehouse..DİM_DEPO ORDER BY kod
```

```
SELECT * FROM Warehouse..DİM_DEPO ORDER BY kod  
asc
```

```
SELECT * FROM Warehouse..DİM_DEPO ORDER BY kod  
desc
```

### Like İşaretleri:

- '%ir\_%' : ir en son olmayacak anlamında.
- '%[A,B]%' : İçerisinde A ya da B geçenleri getirir. Virgül ve ya anlamını içerir.
- '[a-m]%' : Adı a'dan başlayıp m'y kadar olan harfleri getirir.
- 

```
SELECT * FROM warehouse..DİM_DEPO WHERE kod LIKE  
'%GE%'
```

```
SELECT * FROM warehouse..DİM_DEPO WHERE kod LIKE  
'%[N,I]'
```

### Select TOP Komutu: (select top 3 \* from tablo)



- Select top 5 \* from tablo şeklinde kullanılır.
- Aşağıdaki kayıtları getirmek için order by sütun desc şeklinde yazılır.

Syntax:

```
SELECT TOP 10 * FROM warehouse..DİM_DEPO ORDER BY
tanım desc
```

### Select Top Percent Komutu:

- Yüzde olarak getirir, diğer komutlarla kullanıldığında tersten de TOP olarak çağrılır

```
SELECT TOP 65 PERCENT * FROM warehouse..DİM_DEPO
```

### IN Komutu: (Or komutunun Parantezli(var) Hali)

- Syntax:

```
SELECT * FROM warehouse..DİM_DEPO WHERE kod IN(
'MDYA', 'M001', 'K002', 'IDUZ')
```

### Not IN Komutu: İçermeyenleri getirir. Syntax aynıdır.

**Between Komutu:** Parantez kullanılmaz, and ve değer. Harfler için de geçerli. İkinci değer dahil edilmez

- Syntax: `SELECT * FROM warehouse..DİM_DEPO WHERE deporef BETWEEN 110 AND 150`
- Harfler üzerinde de çalışıyor.
- Yazılan ikinci değer dahil edilmez.

```
SELECT * FROM warehouse..DİM_DEPO WHERE kod  
BETWEEN 'A' AND 'd'
```

**Not Between Komutu:** İçermeyenleri getirir Syntax aynı.

## İlişkili Tablolar ve Alt Sorgular

### İlişkili Tablo:

- Veri karmaşasını önlüyor.
- Veri tekrarını önlüyor.
- Bellek performansı sağlıyor. (İsim Soyisim tutmak yerine ID'leri tutmak gibi, bellekte performans arttırır)

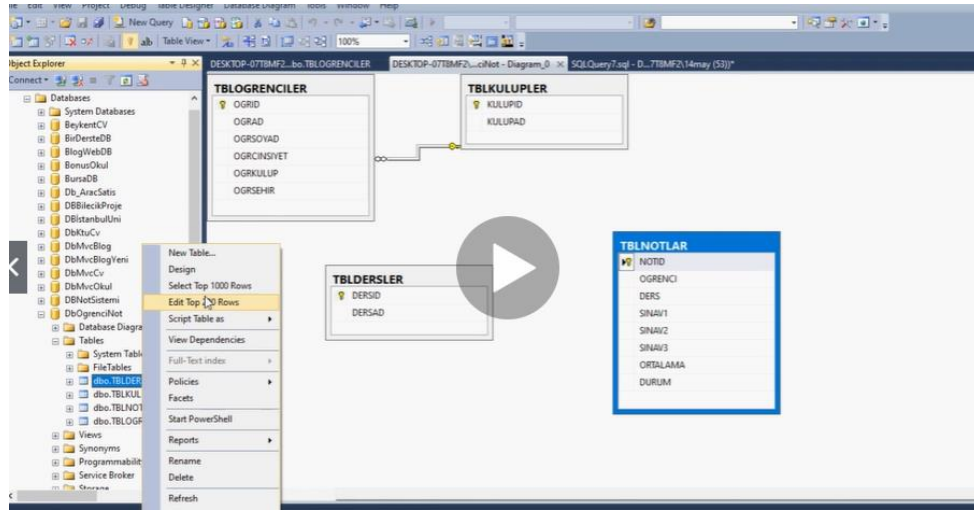
**İlişki Türleri** (İlişki türlerinde eşlenilen sütunların veri türleri eşit olmalı.)

- Bire Bir
- Bire Çok (En çok kullanılan)
- Çoka çok (Çok az tercih edilen)  
(İlişki türlerinde eşlenilen sütunların veri türleri eşit olmalı.)

### İlişki Oluşturmak İçin (Database Diagram)

- Database Diagram alanına, sağ tıklanılır. New Database Diagram oluşturulur.
- İlişkili olacak tablolar sıra sıra seçilir.

- Primary Key(Birincil Anahtar) seçilmeli. Bu tek olan alan olmalıdır.
- Bu alanı seçerken tablodan sağ klik design kısmına geliriz.
- Kolon isminin üstüne gelinip, Set Primary key'e tıklanır.
- Diğer bağlanılacak tabloda Primary Key seçilir.
- Ortak olan alanlardan bağlamalar yapılır.
- Sonsuzluk işareti tarafında kalan kısımda birden çok veri diğer tek bir koşulu sağlayabilir demek.
- İlişkili tablolara veri eklerken Primary Key'de tanımlı olmayan alan eklenirken bu diğer tablolarda yoksa ekleme yapılmaz.
- İlişki alanına yeni bir tablo eklemek için herhangi bir alana sağ tıklanır Add Table denir.



- SSF şeklinde tüm tabloyu çağırdığımızda **sütunlardaki değerler rakam olarak gözükür.**

### Alt Sorgular:

- İç içe geçmiş sorgulardır.
- Syntax:

```
SELECT * FROM Warehouse..DİM_DEPO WHERE Deporef =
(SELECT * FROM Warehouse..Dim_Urun WHERE kod =
'M002')
```

- `SELECT * FROM WareHouse..DİM_DEPO WHERE Deporef = (SELECT MAX(deporef) FROM WareHouse..DİM_DEPO )`

- Alt sorgu birden fazla sonuç döndürüyorsa IN kullanılır.

```
SELECT * FROM TBLHAREKET WHERE URUN IN(SELECT URUN
ID FROM TBLURUNLER WHERE KATEGORI = 1)
```

- İççe geçmiş sorgulardır.
- İççe geçmiş 3 sorgu yazılabilir.

```
SELECT * FROM TBLHAREKET WHERE URUN
IN(SELECT URUN ID FROM TBLURUNLER WHERE
KATEGORI =
(SELECT KATEGORIID FROM TBLKATEGORIID FROM
TBLKATEGORI
WHERE KATEGORIAD = 'BEYAZ EŞYA'))
```

- İççe geçmiş 3 sorgu yazılabilir.

## Tablo Değerini Alt Sorgu ve Update İle Yazdırma:

```
CREATE TABLE #KASA (
toplam DECIMAL (18,2)
)
```

```
INSERT INTO #KASA VALUES (0)
```

- Sütun adı yazmadan ekleme yapabiliriz.
- ID olmayan alanlarda values yazmadan da işlemi halledebiliriz.

```
UPDATE KASA SET TOPLAM = (SELECT SUM (TUTAR) FROM TBLHAREKET)
```

- KASA tablosu tblhareket tablosunun sonucunu döndürür.

## Birleştirme İşlemi

**Inner Join:** (O tabloda o alan yoksa diğer tablodan inner join FAKAT ORTAK OLANLAR GELİR )

- İlk tabloda olmayanlar gelmez. Sadece kesişim gelir.
- Diğer tablodan ortak olan kesişim alanıyla birleştirilir, ortak bir alan olmak zorunda.
- Sadece ortak alanlar gelir. İlk tabloda ve ya ikinci tabloda özel olan alanlar gelmezler.

Syntax:

```
SELECT BuyerGrupref FROM Warehouse..Dim_Urun
INNER JOIN Warehouse..DİM_Depo ON
Warehouse..Dim_Urun.UrunRef =
Warehouse..DİM_Depo..Grupref
```

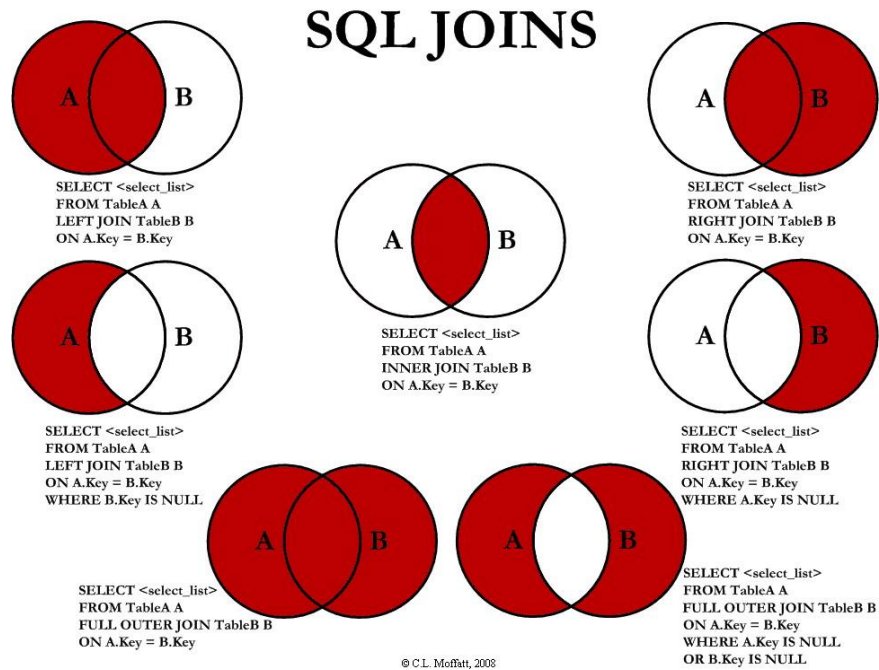
- Selectten sonra farklı sütunlar için farklı tablolardan gelecek verileri bağlamak istiyorsak, her biri için alt alta bağlantı kurulabilir.

```
SELECT BuyerGrupref, Tanim, Renkkod, Deporef
```

```

FROM Warehouse..Dim_Urun
INNER JOIN Warehouse..DIM_Depo ON
Warehouse..Dim_Urun.UrunRef
Warehouse..DIM_Depo..Grupref
INNER JOIN Warehouse..DIM_Depo ON .....

```



### Left Join:

- Birleştirme işlemini ilk yazılan tablonun hepsini alarak yap. Diğer tablodan sadece kesişimleri al.
- Syntax aynı.

### Right Join:

- Birleştirme işlemini ilk yazılan tablonun ortak alanlarını alarak yap. Diğer tablodan hepsini al.
- Syntax aynı.

### Full (outer) Join:

- Tüm alanları alarak birleştirme işlemi yap.
- Syntax aynı.

### Union: (iki Tabloyu Tamamen Birleştirir)

Syntax:

```
SELECT * FROM Warehouse..DİM_DEPO
UNION
SELECT * FROM Warehouse..Dim_Urun
```

### Aritmetik Fonksiyonlar

**Aritmetik İşlemler:** (Tablo Üzerinden Direk Select Yazdıktan Sonra Aritmetik İşlemler Kullanılabilir)

- Syntax:
 

```
SELECT 20+4 AS 'Toplam', 20-4 AS Fark, 20*4 AS 'Çarpım', 20/4 AS 'Bölüm'
```
- ```
SELECT Deporef/2 FROM Warehouse..DİM_DEPO
```

### Matematiksel İşlemler:

**ABS:** (Absolute) Mutlak Değer

- Syntax

```
SELECT ABS (-85)
```

**POWER:** Üs alma komutudur.

- Syntax

```
SELECT POWER(5,2)
```

- 5 in karesini alır.
- Tablo Bazında da kullanılabilir.

```
SELECT POWER(Deporef,2) FROM Warehouse..DİM_DEPO
```

**SQRT:** Karekökünü alır.

- Syntax

```
SELECT SQRT(625)
```

- Ondalıklı değer de döndürür.

```
SELECT SQRT(221)
```

**FLOOR(Taban):** Tabana Yuvarlama Komutu



- Syntax:

```
SELECT FLOOR(4.86)
```

**CEILING (TAVAN):** Tavana Yuvarlama Komutu

- Syntax:

```
SELECT CEILING(5.75)
```

## Alfabetik Fonksiyonlar

**UPPER:** Büyük harfe çevirir.

- Syntax: Kesme işareti ile kullanılır.

```
SELECT UPPER ('burak')
```

- Tüm sütunu da büyük harfe çevirebilir.

```
SELECT UPPER(Tanim) as 'Tanim' FROM  
WareHouse..DİM_DEPO
```

- Sütun adı silinir. As eklenerek eklenebilir.

**LOWER:** Küçük harfe çevirir.

- Syntax: Kesme işareti ile kullanılır.

```
SELECT LOWER ('burak')
```

- Tüm sütunu da büyük harfe çevirebilir.

```
SELECT LOWER(Tanim) as 'Tanim' FROM  
WareHouse..DİM_DEPO
```

- Sütun adı silinir. As eklenerek eklenebilir.

**SUBSTRING:** İstediğimiz karakter sayısını getirir.

- Sütunda yapmadan.

```
SELECT SUBSTRING('burak',2,5)
```

- Syntax: Sütun adı, kaçınıcı karakterden başlayacağı, kaçınıcı, kaç karakter alacağımız.

```
SELECT SUBSTRING(Tanim,2,5) FROM WareHouse..  
DİM_DEPO
```

```
SELECT SUBSTRING(Tanim,2,5) + '.' + tanim FROM  
WareHouse.. DİM_DEPO
```

**LEFT:** Soldan kaç karakter alacağını belirleriz.

- Sütunda yapmadan.

Syntax:

```
SELECT LEFT('burak',3 )
```

- Syntax: Sütun adı, kaç karakter alacağımız.

```
SELECT LEFT(Tanim,2) FROM WareHouse..DİM_DEPO
```

**LEN:** Bir ifadenin kaç karakter olduğunu verir.

- Sütunda yapmadan.
- İfadenin sağ tarafındaki boşlukları saymaz.

Syntax:

```
SELECT LEN('Tanim')
```

- Syntax: Sütunun adı parantez içerisinde.

```
SELECT LEN(Tanim) FROM Warehouse..DİM_DEPO
```

- Fonksiyon kullanımları esnektir.

```
SELECT tanim, LEN(Tanim) FROM Warehouse..DİM_DEPO
```

- Belli karakter uzunluktakileri sutundan çağırmak istiyorsak parantez içerisine sütun adı yazılıp sayıya eşitlenmelidir.

```
SELECT tanim, LEN(tanim) FROM Warehouse..DİM_DEPO  
WHERE LEN(tanim)=10 OR LEN(tanim) = 11
```

**LTRİM:** Sol taraftaki boşluğu sıfırlar.

```
SELECT LTRIM('FENERBAHCE')
```

**RTRIM:** Sağ taraftaki boşluğu sıfırlar.

```
SELECT RTRIM(' FENERBAHCE ')
```

**Replace:** 3 parametreyle çalışır. Kelimeyi ve ya harfi değiştirir.

```
SELECT REPLACE('FENERBAHCE', 'BAHCE', 'FENER')
```

- Ana cümle, değiştirilecek kısım, yerine koyulacak eleman.

```
SELECT REPLACE('FENERBAHCE', 'e', 'a')
```

**CharIndex:** 2 parametreyle çalışır. Aradığınız bir ifadenin kaçınıcı sırada olduğunu verir.

```
SELECT CHARINDEX('a', 'SQL DERSLERİNE DEVAM  
EDİYORUZ')
```

- Eğer belli bir karakterden sonrasını aramak istersek 3. Komut devreye girer.

```
SELECT CHARINDEX('e', 'SQL DERSLERİNE DEVAM  
EDİYORUZ', 20)
```

- Sütunda kullanmak istediğimiz zaman.

```
SELECT CHARINDEX('a', tanım) FROM  
WareHouse..DİM_DEPO
```

**Reverse:** İfadeyi tersten yazdırır.

```
SELECT REVERSE('fenerbahce')
```

- Sütun olarak tersten yazdırır.

```
SELECT REVERSE(tanim) FROM WareHouse..DİM_DEPO
```

## GENEL TEKRAR

Veri Ekleme İşlemleri:

DML Komutları Tekrar

UPTADE Kullanımı

## Prosedürler

- Prosedürler uzun sql sorgularını tek bir kelime ile çağırmaya yarar. Programlama dillerindeki metotlar gibidir.
- C# ve ya başka programlama dillerindeki gibi uzun yazılan kodlarda kullanılır.

- Örneğin çokça inner join kullandık. Bunu procedure ile kısaltabiliriz.
- Syntax: Örneğin Create Procedure veya Create Proc

```
CREATE PROC/
CREATE PROCEDURE HAREKETLER
AS

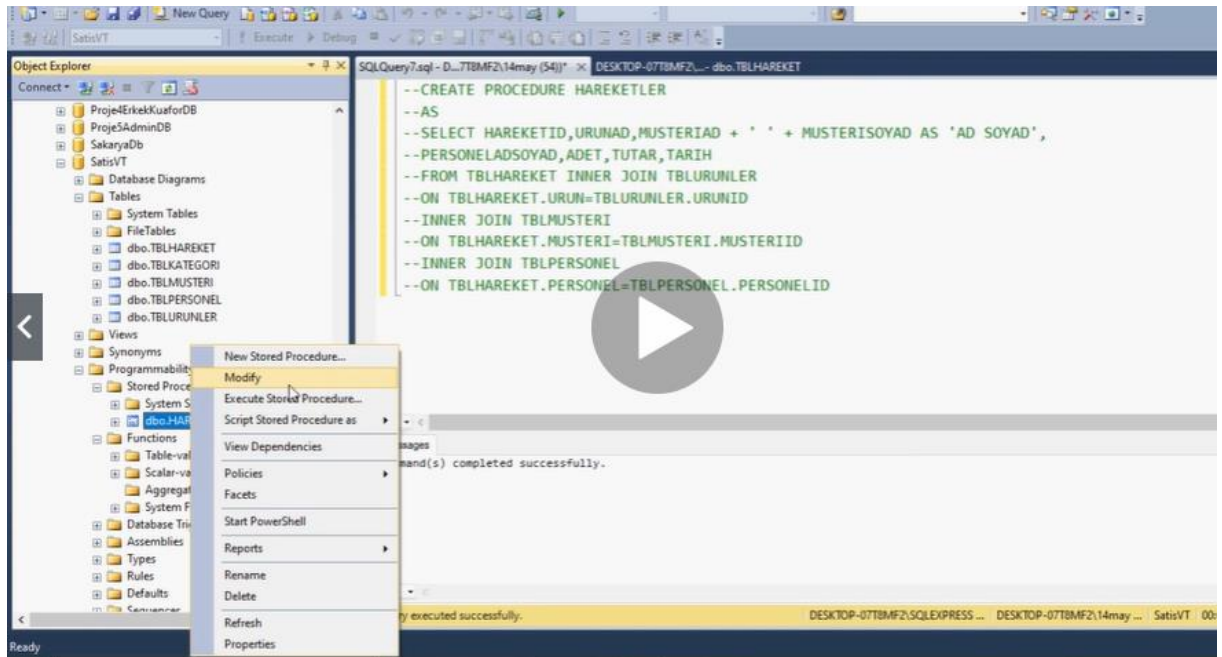
SELECT BuyerGrupref,Tanim,Renkkod,Deporef
FROM WareHouse..Dim_Urun
INNER JOIN WareHouse..DİM_Depo ON
WareHouse..Dim_Urun.UrunRef =
WareHouse..DİM_Depo..Grupref

INNER JOIN WareHouse..DİM_Depo ON .....
```

- Çalıştırmak için
- Syntax:

```
EXECUTE HAREKETLER
```

- Yazılan procedureler projenin altında programmability kısmının altında store procedures alanına kaydolurlar. Üzerine sağ tuş modify dersek, o store prosedüre ait satırları görebiliriz.



## Prosedür Silme ve Güncelleme

### Drop Proc:

```
CREATE PROCEDURE HAREKETLER
AS
```

```
SELECT BuyerGrupref,Tanim,Renkkod,Deporef
FROM WareHouse..Dim_Urun
INNER JOIN WareHouse..DİM_Depo ON
WareHouse..Dim_Urun.UrunRef
WareHouse..DİM_Depo..Grupref
```

```
INNER JOIN WareHouse..DİM_Depo ON .....
```

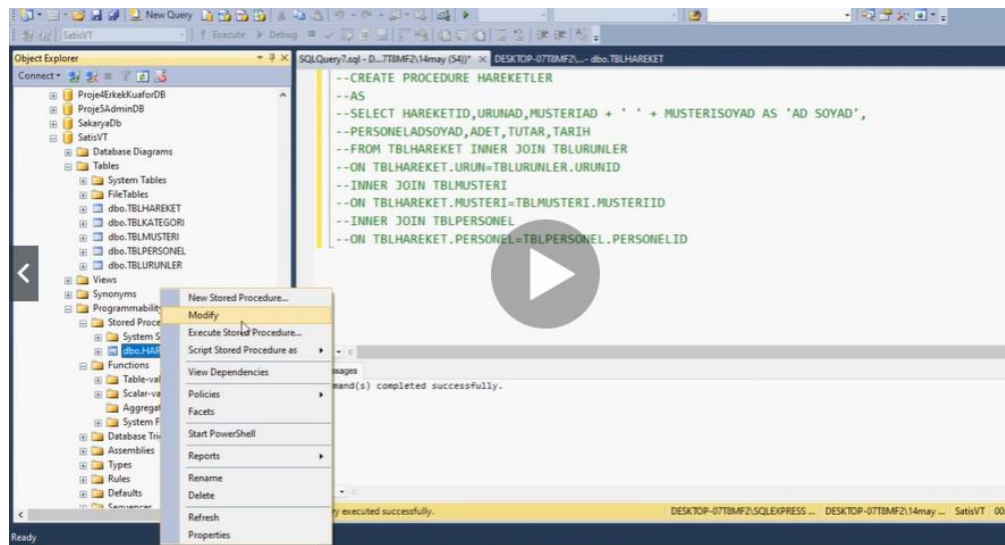
```
EXECUTE HAREKETLER
```

- Silmek için.

```
DROP PROC HAREKETLER
```

## Alter Proc: (Direk Create yerine alter yazarak)

- Yazılı proceduren ustune gelip direk alter yazarsak değiştirebiliriz.
- Stored Procedures kısmından kayıtlı procesdulerle girilir ve kopyalanır, kopyalandığında alter şeklinde gelir.
- Değiştirilmek veya eklenmek istenen alan değiştirilir. Alter komutuyla çalıştırılır ve değişiklik gerçekleşmiş olur.



ALTER PROCEDURE HAREKETLER  
AS

```
SELECT BuyerGrupref,Tanim,Renkkod,Deporef
FROM WareHouse..Dim_Urun
INNER JOIN WareHouse..DIM_Depo ON
WareHouse..Dim_Urun.UrunRef =
WareHouse..DIM_Depo..Grupref
INNER JOIN WareHouse..DIM_Depo ON .....
EXECUTE HAREKETLER
```



## Parametrelili Procedure:

- Değişkenler @ sembolüyle tanımlanır.
- Alter komutundan sonra kullanılır.
- Alterden sonra parametrenin neye eşit olacağını seçeriz.
- Daha sonra veri türünü belirleriz.
- Where şart yazılır, parametreye (değere) neyi gönderirsek onu getirsin.
- Parametre eşitliğini verdikten sonra virgül koyarakta yeni parametre tanımlayabiliriz.

```
ALTER PROCEDURE HAREKETLER  
@Deger = varchar (50) = 'Buzdolabı'
```

```
AS  
SELECT BuyerGrupref,Tanim,Renkkod,Deporef  
FROM WareHouse..Dim_Urun  
INNER JOIN WareHouse..DİM_Depo ON  
WareHouse..Dim_Urun.UrunRef =  
WareHouse..DİM_Depo..Grupref  
INNER JOIN WareHouse..DİM_Depo ON ..
```

```
WHERE = URUNAD = @Deger
```

- EXECUTE diyerek çalıştırılır ve verdiğimiz parametrenin eşit olduğunu basar.

```
EXECUTE HAREKETLER
```

## Parametre Değiştirmek

- Execute prosedür @parametre = 'Yeni Değer'
- Syntax:

```
EXECUTE HAREKETLER @Deger = 'Laptop'
```

## Date Sorguları

### GETDATE: Bugunun tarihini verir

GETDATE()

şeklinde kullanılır.

### Datapart: İki tarih arasındaki veriyi getirir.

- Where komutundan sonra,
- Datapart fonksiyonu (DAY, Sütun ismi)  
(MONTH, Sütun ismi)  
(YEAR, Sütun ismi)
- Sonrasında, BETWEEN kıstas AND kıstas  
Şeklinde yazılır.

```
SELECT * FROM warehouse..DİM_DEPO WHERE  
DATEPART (DAY, AcilisTarihi ) BETWEEN 1  
AND 3 AND deporef = 1000
```

### Datename:

- Verilen tarihi isim olarak yazdırır.
- Tarih değeri kıstas olarak girilmelidir.
- Month ayı name olarak verir.

- DAY olarak da kullanılabilir. Sayı olarak gün.
- WEEKDAY olarak da kullanılabilir, haftanın günü.
- Fonksiyonların arasında virgül olmalıdır.

SELECT

```
DATENAME(MONTH, GETDATE()),
DATENAME(MONTH, '2019.01.15'),
DATENAME(WEEKDAY, GETDATE()),
DATENAME(DAY, GETDATE()),
DATENAME(WEEKDAY, '2020.06.28')
```

- Bir gün hatırlanmadığında o günün o sene hangi güne denk geldiği bulunabilir.

Datediff:

- Verilen tarihlerin arasındaki farkı istenilen türde verir
- Tarih değeri kıstas olarak girilir.
- Dateneden farklı olarak 2 tarih girilir.

```
SELECT DATEDIFF(YEAR, '2010.10.25',
GETDATE())
```

```
SELECT DATEDIFF(MONTH, '2010.10.25',
GETDATE())
```

- Dateneden farklı olarak 2 tarih girilir.
- Sütunun tamamının farkı bulunabilir.

```
SELECT DATEDIFF(month, acilistarihi,
getdate()) FROM warehouse..DİM_DEPO
```

- Veya where komutu kullanılarakta belirli verilerin farkı alınabilir.

```
SELECT DATEDIFF(month, acilistarihi,
getdate()) FROM warehouse..DİM_DEPO
WHERE deporef =1000
```

### Dateadd:

- Verilen tarihlerin üzerine ekleme yapar.
- Tarih değeri kıstas olarak girilir.
- Datediff ile aynı şekilde 2 kıstas girilir.
- Tarih değerinden sonra eklenecek tarih miktarı
- Ne zamana ekleneceği girilir.

```
SELECT DATEADD(YEAR, 5, GETDATE())
```

### Cast: Fonksiyonu

Conve

### Tetikleyiciler

**Trigger:** Değişiklikleri aktif olarak tablodan alır

- İç içe geçmiş sorgularda bir tabloda olan herhangi bir değişikliğin bir başka tabloya **aktif olarak** yansımaları için kullanılır.

```
CREATE TABLE #TBLSAYAC
(ISLEM int )
```

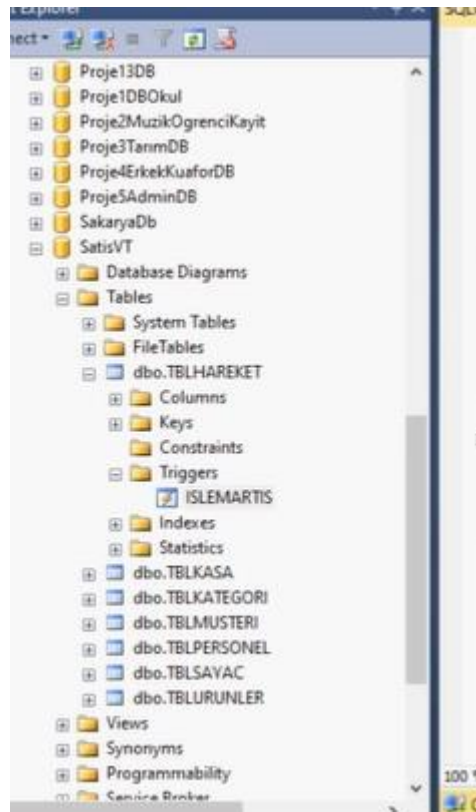
```
INSERT INTO #TBLSAYAC VALUES(0)
```

```
UPDATE #TBLSAYAC SET ISLEM = (SELECT COUNT (*) FROM  
TBLHAREKET)
```

```
CREATE TRIGGER ISLEMARTIS ON TBLHAREKET  
AFTER INSERT  
AS
```

```
UPDATE #TBLSAYAC SET ISLEM = ISLEM +1
```

- Eğer işlem 1 artarsa diğer tabloda da arttır demek.
- Önce alt sorgular ve update komutu ile bağlantılı sorgu oluşturulur.
- Daha sonra son tabloda işlem sayısı arttığında artışı görebilmek için trigger komutu yazılır.
- Sonrasında UPDATE komutu ile önceki sütuna ekleme yapılır.
- Ayrıca View Object kısmında Tables alanının altında triggers olarak kaydolur
- Sağ tuş modify dersek açabiliriz.
- Alter komutu ile güncelleyebiliriz.



## View

**View:** Sanal tablolar

- Prosedure gore dezavantajı dışardan parametre almamasıdır. Prosedurde parametre alabilir
- Sanal tablolardır.
- Prosedurlere benzeyen fakat ondan farkı olan bir yapı.
- Birleştirme işlemlerinde uzun sql sorgularını, tek bir değer, tek bir komut içerisine toplar.

```
CREATE VIEW #TEST1
```

```
AS
```

```
SELECT * FROM warehouse..Dim_Depo
```

- Oluşturduktan sonra bu alana gelir.



- 2. kısımda yani ortadaki kısımda verilerin sıralama türleri neye göre sıralanacağı gözükür,
- 2. Kısım Aynı zamanda kıstasları yani where alanını da içerir. Where kısmında değişiklik yapıp filtreyi değiştirebiliriz.
- Alias kısımları takma adları gösterir. Bu alanlar visual studioya geçildiğinde çokça kullanılacaklar.
- 3. En alt kısımda bu tabloya ait yukarıda seçilen sütunlara göre sorgu kısmı mevcut.
- Select \* from view ismi şeklinde çalıştırılır.

**SELECT \* FROM #TEST1**

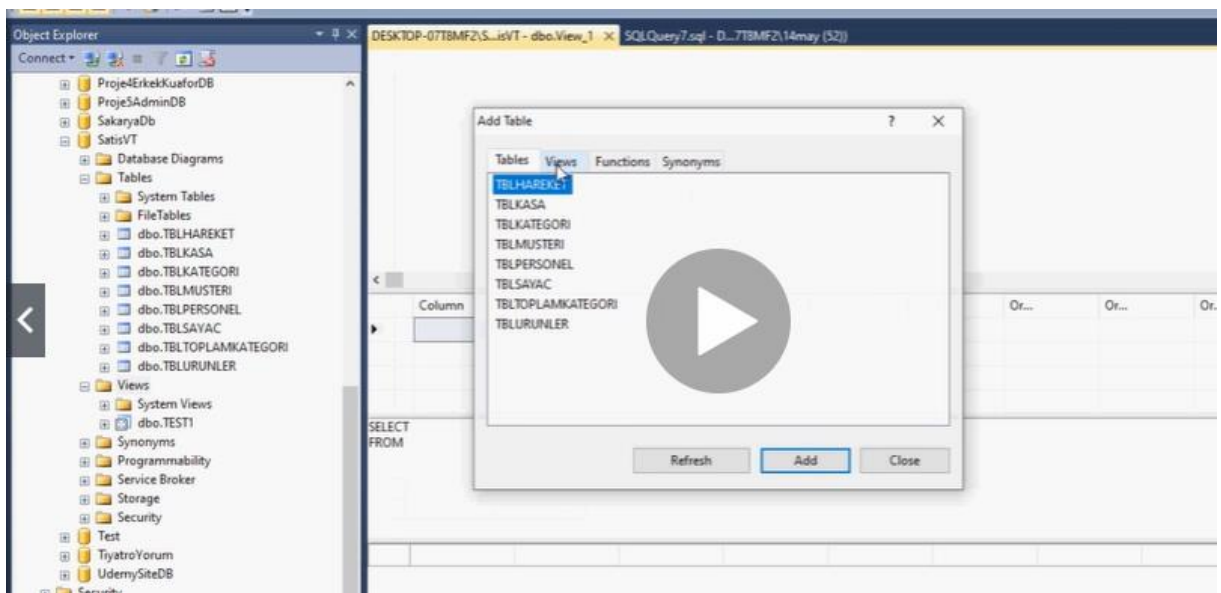
- Güncelleme işlemi Alter komutu ile yapılır.

**ALTER VIEW TEST1**

**AS**

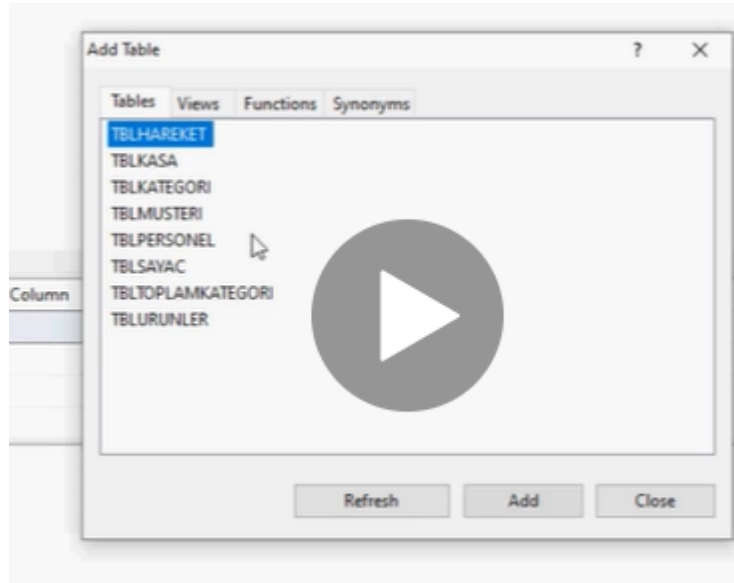
**SELECT \* FROM warehouse..Dim\_Depo WHERE DepoRef <15**

## View Sihirbaz Kullanımı:

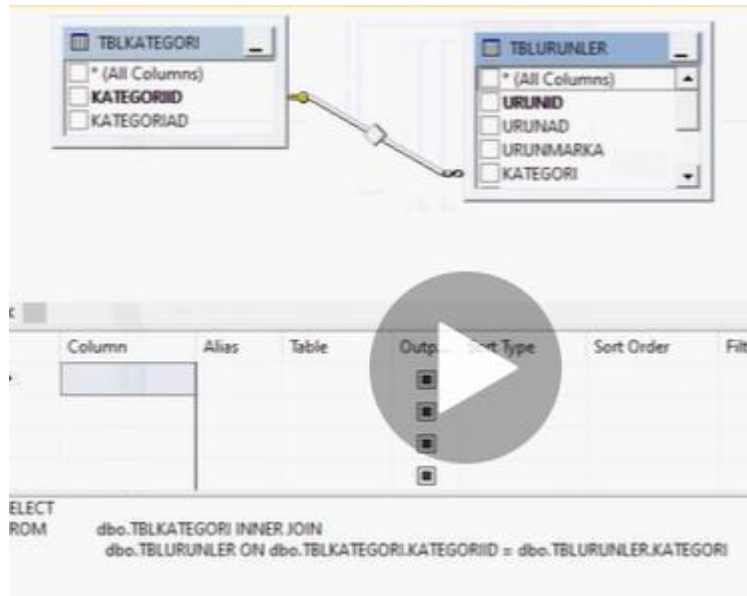




- View üstünde Sağ tuş – new view dedikten sonra birleştirme işlemi üzerinden yapılır.
- Birleştirirken view da eklenebilir.



- Boş alana sağ tuşa tıklayarak add table diyerek tablolar karşımıza çıkar.



- İlişki bu alandan verilir.
- Eğer otomatik ilişki verdiyse soldaki Diagram kısmında o ilişki önceden verilmiştir.
- Sorgudaki Tüm değişiklikleri sihirbaz gösterir.

### Norhwind Veri Tabanı Sorguları

- Microsoft tarafından hazırlanmış, kendini geliştirmek isteyenlerin kullandığı veri tabanıdır.

### T-SQL Komutları (Transact)

- Değişkenleri içerir

#### Declare: Değişken Tanımlama

- Declare etmek, tanımlamak anlamına gelir.
- Sql'de değişken isimleri @ sembolü ile tanımlanır.
- Declare dedikten sonra türü belirlenir.

**DECLARE** @değişken **INT**

- Tür belirlendikten sonra set ile değer atanır

```
DECLARE @değişken INT
```

```
SET @değişken = 24
```

```
SELECT @değişken
```

- Declare – set – select beraber çalıştırılmalıdır.
- Birkaç değişken tanımlama;
- Declare dedikten sonra değişkenler arası virgül olmalı. Fakat en sonuncusundan sonra olmamalı.
- Her bir sayı için ayrıca set komutu kullanılmalı.
- Setler arası virgül kullanılmaz

```
DECLARE @sayi2 INT, @sayi3 INT, @toplam INT
```

```
SET @sayi2 = 3
```

```
SET @sayi3 = 4
```

```
Set @toplam = @sayi2 + @sayi3
```

```
SELECT @toplam
```

- Değişkenler yani declare komutu tablolar üzerinde de kullanılabilir.
- Örn: Yazılan sayının rakamlarını toplama.

```
DECLARE @sayi INT, @birler INT, @Onlar INT, @yuzler  
INT, @toplam INT
```

```
SET @sayi =456
```

```
SET @birler = @sayi %10
```

```
SET @onlar =( @sayi /10) &10
```

```
SET @yuzler = @sayi /100
```

```
SET @toplam = @birler + @onlar + @yuzler
```

```
SELECT @birler,@Onlar, @yuzler , @toplam
```

- SELECT 500 %9, 9 ile bölümünden kalanı verir.
- Setlerden sonra virgün kullanılmaz.

### Tablo İçin Declare Değişkenler:

- Değişken kullanarak tablodan bir verinin değerini getirebiliriz.
- Declare içerisine alt sorgu yazılmasıyla.

```
DECLARE @maxref INT
```

```
SET @maxref = (SELECT MAX(Deporef) FROM  
WareHouse..Dim_Depo)
```

```
SELECT @maxref
```

- Örn, maximum olan ürünün tüm bilgilerini subquary ile çağırabiliriz.

```
DECLARE @maxref INT
```

```
SET @maxref = (SELECT MAX(Deporef) FROM  
WareHouse..Dim_Depo)
```

```
SELECT * FROM WareHouse..Dim_Depo WHERE DepoRef = @maxref
```

### Sistem Fonksiyonlar:

- Çok işe yarar değildir.
- C# tarafında 2 komut olan version ve server name çok işe yarayacaktır.
- Servername önemli kalan, error sayısı, dil, max bağlantı, versyon, identity, textsize burdan bulunabilir.
- Ayrıca USE komutu o tabloyu

```
SELECT  
@@SERVERNAME,  
@@ERROR,  
@@DEFAULT_LANGID,  
@@LANGUAGE,  
@@MAX_CONNECTIONS,  
@@VERSION,  
@@IDENTITY,
```

```
@@TEXTSIZE,  
@@VERSION
```

- **USE** komutu database değiştirirken kullanılır.

### Print Kullanımı:

- Sql de önemli bir yeri vardır.
- Tablo değil'de, Bilgi bazlı ekrana yazı yazmak istersek print yazarız.

```
DECLARE @sayi2 INT, @sayi3 INT, @toplam INT
```

```
SET @sayi2 = 3
```

```
SET @sayi3 = 4
```

```
Set @toplam = @sayi2 + @sayi3
```

```
PRINT @toplam
```

```
PRINT 'Sonuç Budur'
```

**Tablo Tipi Değişkenler:** Geçici tablo oluşturur.

- Bir değişken yardımıyla birden fazla tablo değeri tutulabilmesine olanak sağlıyor.
- Geçici tablo oluşturmak gibi düşünülebilir.

```
declare @Kisiler table
```

```
(  
  KisiID int identity (1,1),  
  KisiAD varchar(10),  
  KisiSehir varchar(15)  
)
```

```
insert into @Kisiler (KisiAd,Kisisehir) values ('Ali',  
'Malatya')
```

```
insert into @Kisiler (KisiAd,Kisisehir) values  
( 'Emel', 'Trabzon')
```

```
Select * from @kisiler
```

**If Else Kullanımı:** If eğer, Else değilse.

```
if (1<5)
print 'Merhaba'
```

```
else
print 'Selam'
```

- If içerisine subquery şeklinde de yazılabilir.
- Kullanımı esnek.
- Else tersi durum olduğu için olumsuzunu uzun uzun yazmamak amaçlı kullanılır.

```
IF (select Sum(BuyerGrupRef) from warehouse..Dim_Urun) >
100
```

```
Print 'Toplam Buyer Grup Ref Sayısı 100den Büyük'
```

```
Else
```

```
Print 'Toplam Buyer Grup Ref Sayısı 100den Küçük'
```

**Kullanımı:** If eğer, Else değilse. Case yazılmadan önce Eşittir kullanılır, ve en son End)

- Dallarmanın çok olduđu durumlarda kullanılan yapılardır.
- Son sütun CASE'e eşitlenip kullanılabilir, böyle yapılırsa Depotip = yazmak zorunda kalmayız. Sütun ismini de selectten sonra yazmak zorunda değiliz, en son as denilerek yazılır.
- Fakat bu yazılış diğer sütunlar için içine girdiğinde sorun çıkarmakta, o yüzden tek tek yazmak daha mantıklı.
- If koşulunun fazla kullanılması gereken durumlarda kod karmaşasından kurtulmak için case kullanımı iyi bir alternatiftir.
- Programlamadaki switchcase yapısına benzemektedir.
- Sütunları getirdikten sonra, caseler yazılır.
- CASE yazıldıktan sonra hangi duruma göre case yazacaksak o yazılır. (O sütunun ismi)
- Whenden sonra istediğimiz çıktı Then yazılır.
- Whenlerden sonra En son END yazılır.
- En son AS sütun ismi diyerek, yeni sütun ismi belirlenir.

```
select Deporef, Kod, Tanim,
```

```
CASE
```

```
When Depotip = '1' Then 'Bir'
```

```
When Depotip = '2' Then 'İki'
```

```
When Depotip = '0' Then 'Sıfır'
```

```
When Depotip is NULL Then 'Boş'
```

```
END
```

```
AS Depotip
```



## İs not Null Kullanımı :

Where koşuluna yazılır.

### Syntax:

```
SELECT kod, tanim, UlkeAdi, OperasyonelBolgeKod,  
AcilisTarihi  
FROM WareHouse..Dim_Depo WHERE AcilisTarihi IS NOT NULL
```

## Case When İle Kritik Durum Belirleme

- Son sütun CASE'e eşitlenip kullanılabilir, böyle yapılırsa Depotip = yazmak zorunda kalmayız. Sütun ismini de selectten sonra yazmak zorunda değiliz, en son as denilerek yazılır.
- Fakat bu yazılış diğer sütunlar için içine girdiğinde sorun çıkarmakta, o yüzden tek tek yazmak daha mantıklı.
- Aralık değerlerilir, End ' den sonra as sütun ismi tanımlanır.

```
select Deporef, Kod, Tanim,
```

```
CASE
```

```
When DepoTip>=0 AND DepoTip < 1 Then 'Kritik Seviye'
```

```
When DepoTip>=1 AND DepoTip < 2 Then 'Normal Seviye'
```

```
When DepoTip>=2 AND DepoTip < 3 Then 'Yüksek Seviye'
```

```
When DepoTip>=10 Then 'Yeterli'
```

```
END
```

```
As Depotip
```

```
from warehouse..Dim_Depo
```

**While Döngüsü:** Declare ile kullanılır. Tekrar gerektiren durumlarda kullanılır.

- Toplam kar oranı %20 ye ulaşana kadar ürün satışı gerçekleşmeye devam etsin gibi.
  - Stok sayısı 10'u geçene kadar her ürüne 1 tane ürün eklesin gibi.
  - Declare ile kullanılır önce veri türü belirlenir.
  - Sonra set ile değer ve ya değerler atanır.
  - Set = başlayacağı değer belirlenir.
  - While @değişkenin sınırı belirlenir.
- While sınırları ifadenin kaç kere tekrar edeceğini belirler
- Begin komutu yazılır
  - Print (her gerçekleştiğinde istenen çıktı)
  - Set =Döngü kuralı.
  - End ile bitirilir.

**Not:**

```
set @sayac = @sayac + 1
```

ifadesi ile,

```
set @sayac +=1
```

ifadesi birbirine eşittir.

Kendisi artı 1 demektir.

```
Declare @sayac int
```

```
set @sayac = 1
```

```
while @sayac <=5  
  
Begin  
  
Print 'Sayıya bir ekledim'  
  
set @sayac = @sayac + 1  
  
end
```

- Print komutunun yeri değiştirilirse sonuçta değişecektir.

```
declare @x int  
  
set @x = 1  
  
while @x <=10  
  
begin  
  
print @x  
  
set @x +=1  
  
end
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

- Önce yazdır sonra ekle şeklinde yazılırsa, önce 1. Sayıyı yazdırır sonra ekleyerek devam eder.

```
declare @x int
```

```
set @x = 1

while @x <=10

begin

set @x +=1

print @x

end
```

2  
3  
4  
5  
6  
7  
8  
9  
10  
11

- Önce ekle sonra print et şeklinde yazılırsa yazılan ilk sayı eklenmiş olan 2 olur ve 11 e kadar yazar.

### Örnek:

```
Declare @toplam int, @sayac int

Set @sayac = 1

Set @toplam =0

while (@sayac <= 10)

begin

set @toplam = @toplam + @sayac

set @sayac +=1
```

End

Print '1 ile 10 arasındaki sayıların toplamı'

Print @toplam

### Örnek:

- Sütun'da bir değer, belli bir değerin altına düştüğünde döngü başlatarak değere ekleme yapma.

```
While ( select AVG (deporef) from warehouse..Dim_Depo) <= 2800
```

Begin

```
Update warehouse..Dim_Depo set Deporef+= Deporef*10/100
```

End

```
Select top 10 deporef from warehouse..Dim_Depo
```

- Değişken tanımlamadan while, Begin , Update, End komutları kullanılarak yazılabilir.

**WaitFor Delay:** Beklemek için anlamında, sorgumu şu zaman göre çalıştır anlamında.

- Diğer komutlarla da kullanılır, update, insert into gibi.
- Zaman aralığı saat vs de eklenebilir.

```
WAITFOR DELAY '00:00:07'
```

```
Select top 10 deporef from warehouse..Dim_Depo
```

### For XML Path Kullanımı:

- Tablo şeklinde oluşturulmuş verilerden karmaşık XML belgeleri oluşturulmak isteniyorsa kullanılır.

```
SELECT p.FirstName,  
p.LastName,  
s.Bonus,  
s.SalesYTD  
FROM Person.Person p  
JOIN Sales.SalesPerson s  
ON p.BusinessEntityID = s.BusinessEntityID  
FOR XML PATH
```

```
<row>  
  <firstname>Stephen</firstname>  
  <lastname>Jiang</lastname>  
  <bonus>0.0000</bonus>  
  <salesytd>559697.5639</salesytd>  
</row>  
<row>  
  <firstname>Michael</firstname>  
  <lastname>Blythe</lastname>  
  <bonus>4100.0000</bonus>  
  <salesytd>3763178.1787</salesytd>  
</row>  
<row>  
  <firstname>Linda</firstname>  
  <lastname>Mitchell</lastname>  
  <bonus>2000.0000</bonus>  
  <salesytd>4251368.5497</salesytd>  
</row>
```

Şeklinde sonuç elde edilir.

Eğer parametrelili olarak değeri atarsak, şekildeki gibi XML dökümanı elde edilir.

C# ile SQL  
Visual Studio

#### Var Olan Projeyi Açma:

- Takım > Bağlantıları Yönet > Tekrar Bağlantıları Yönet > Bir Projeye Bağlan
  - 2008 Imported = Sepet Büyüklüğü – ROWA
  - Tabular 2016 = SAR , Bağlan Dedikten Sonra

Dosya> Kaynak Denetimi > Gelişmiş > Sunucudan Aç > Tabular  
2016 Projects Tfs > Mağazacılık > SAR 2017 sln. (Yerel Yol =  
C:\Project\SAR2017) > Çözüm Gezgini ( Sağ Altta)

- Değişiklik yapacaksak sağ tık, son sürümü al.
- Check in deploy işlemi yapacaksak , model bim e sağ tık  
Düzenleme için kullanıma Al demeliyiz.
- İşlemleri yaptıktan sonra iade et alanı çıkıyor, sağ tıkladıktan  
sonra iade etmeliyiz.

**Partitions Alanı:** Aşağıdan alan seçilir > Table > Partitions 'tan  
ulaşılır.

- Örneğin SAR cube'te depo bilgilerinde neler eksik neler  
filtreleniyor buradan ulaşılır.

- Where koşulunda neler yazılmışsa onlar

**Not:** AnaGrupRef=1 Açık Mağazalar demek.

Kapanmış mağaza = Pasif Mağaza %Psf%

Hedef Mağaza = Hedef Mağaza %Hdf%

Bu mağazalar ROWA'da görünebiliyorlar. Fakat SAR'da  
almıyoruz.

Bu sorguda mağazalar geliyorsa, SAR cube'te mağazalar  
geliyordur. Yoksa yer verilmiyordur.

- Table Kısmından tüm alanları görebiliriz. En önemlisi Satış – Stok
- Yaşadığımız hafta günlük 1'de geçen hafta günlük 2'de yer  
alıyor. Hafta hafta geriye gitmekte.
- Sorgular partitionlarda bölünmüş şekildeler.
- Burdaki partlar içerisindeki sorgunun ana sorgudan farkı sıra  
No'lar.
- Örn: Günlükler, MerchYılHaftadan, YılAygüne iniyo. Haftalıklar  
sadece MerchYılHafta'dan.
- Günlüklerin datamartı gündepoklasman.



- Haftalıkların datamartı haftadepoklasman.,
- Where'den sonra Exits (içerir( komutuyla, gün hafta12 'deki merchylhafta ile depoklasmandaki merchylhafta bağlanmış.
- Temp tabloların isimlerinin hepsi farklı.

**SP Bulma:** Data Base Engine > TemaOltp > DataBases > Aranılan Database(Ozet) > Programmability > Stored Procedures

- SP sonunda sıra varsa ona göre çalıştırır.
- Sağ Tık modify dersek SP içerisindeki sorgu açılır.
- SP'yi çalıştırmak için Declare'den öncesi yorumla alınır.
- Parametrelili olan kısma direk değer girilir.

### **Dax Bulma:**

- Aşağıdan alan seçilir. Üzerine gelindiğinde formülde gözükür.
- SP'nin içerisine gidilir. Ordan control find ile bulunur. Fakat kısıt verilerek (betweenler arasına) SP çalıştırılır.
- Sevcan Hanım dax açıklamalarını az çok biliyor.

### **Sorguda Değişiklik ve Check-In Deploy**

- Değişiklik yapılsa dahi sorgu önce sql de çalıştırılır.
- Partitions içerisine yapıştırılır.
- Validate'e (doğrulama) tıklanır.
- Son sürüme alınır.
- İade edilir. (Kod Gelir)
- Deploy edilir.
- En son bu işlem aynı gün gözükmesi için p3'ten process edilmeli daha sonra bitabular1'e senkronize edilmeli.

- Sepet Büyüklüğü ve ROWA için deploy talebi jiradan Ahmet Bey'lerin ekibine açılır.
- Sorgudaki eksikleri yanlışları kontrol ederler. Acil bir şey var ise Ahmet Bey'lere çok az değişiklik yapıldığını belirt.
- Processini de onlar yapıyor.
- Sadece SAR'da process ve deploy yapabiliyoruz. Diğerlerinde sadece Check – In işlemi yapabiliyoruz.
- Sepet Büyüklüğü-ROWA'da iade ettikten sonra çözüm gezgininde Check'ın Numarası geliyor. Daha sonra Ahmet Bey'lere jiradan talep açıyoruz.

## Important Data Types:

### String

- Char: Hem sayı – hem karakter (0-100 arası karakter)
- Varchar: Hem sayı – hem karakter (0-250 arası karakter)  
Bu değişkenler Unicode diye adlandırdığımız karakterleri desteklemezler.  
Latin alfabesi verileri için kullanılır.
- NChar: Hem sayı – hem karakter (0-100 arası karakter) (Sabit Uzunluklu Veriler İçin)
- NVarchar: Hem sayı – hem karakter, Metin (0-250 arası karakter) (Değişken Uzunluklu Veriler İçin)

Latin alfabesi dışında veri eklenileceği zaman kullanılır.

- Image: Fotograf

Bu değişkenlerin içinde Unicode karakterleri barındırabiliriz.

Dolayısıyla farklı dillerdeki verileri saklamaya imkan sağlar.

Nvarchar tıpkı Varchar gibi tanımlama boyutuna göre değil içinde bulunan değerın uzunluğuna göre bellekte yer kaplamaktadır.

- + Numeric

- Bit: 1'den 64'e kadar sayı

- SmallInt: -32.000 arasında

- Integer: Yalnızca karakter

- Float: Sonuna bir ekleme yapmazsak tam sayıdır, fakat virgülden sonrası için ekleme yaparsak, float değişkeni 3.4E +/- 38 (7 basamak) arasında değer depolayabilmektedir.

- Money: Para birimi

- + Date And Time Data Types

- Datetime: YYYY-MM-DD - 1753/9999

- SmallDateTime: 1900/2079

- + Gruplandırılmamış:

Geography: Konum

Hierarchyid: Kategori

**Not:** Sadece bu data tipleriyle çoğu sütun tanımlanabilir, boyut büyüdükçe data tipleri daha spesifik seçilmelidir.







### Where:

**WHERE** StogaEtkisi<>1 AND UrunKlasmanRef NOT IN (9000,8600,-1) AND  
dd.AnaGrupRef=1 AND dd.DepoTip=3 and UlkeRef <> 48  
and g.MerchYilAy between 201701 and 201806

Tüm filtreler burada atılır. Poşetlerin çıkarılması filtresi burada eklenir, NOT IN (9000, 8600, -1). Türkiye mağazalarını almamak için <> 48 denilir.

**Group by:** Öncelikle sıralamak istediğin sıralamaya göre sütunlar yazılır.



**Order By:** Selectten sonraki kısmın, aynısı. Tekrar olarak en sona yazılır.

### Önemli Not For Sql:

SQL de tablo içeriklerine bakma (Sütünları satır şeklinde tablo olarak)

Sadece sütunlara bakarsın.

- Hangi veritabanındaysa onu seçiyoruz genelde **lcwarehouse** için **warehouse** u seçiyoruz.
- Veya yukardan **tempdb** seçiyoruz.
- Sadece veri tabanlarını çalıştırır ve tek tek çalıştırır, sql de fromdan sonra gelen veritabanlarını. Warehouse gibi.
- Ardından **ALT+F1 e** tıkladığımızda aşağıdaki ekran karşımıza gelir.
- İlgili tabloda yer alan değerleri buradan görebiliriz.

Name	Owner	Type	Created_datetime
1 StokSatis_GunDepoUrun	dbo	view	2015-05-05 17:04:00.513

Column_name	Type	Computed	Length	Prec	Scale	Nullable	Trim TrailingBlanks	FixedLenNullInSource	Collation
1 YilAyGun	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2 DepoRef	int	no	4	10	0	no	(n/a)	(n/a)	NULL
3 UrunRef	int	no	4	10	0	no	(n/a)	(n/a)	NULL
4 SatisAdet	int	no	4	10	0	yes	(n/a)	(n/a)	NULL
5 OrjinalDovizTipiRef	int	no	4	10	0	yes	(n/a)	(n/a)	NULL
6 IskontoluKdvDahilNetSatisTutar_TL	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
7 GenellskontoTutar_TL	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
8 SatiriskontoTutar_TL	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
9 KDV_Tutar_TL	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
10 USDCevimeKuru	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
11 EuroCevimeKuru	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
12 OrjinalDovizCevimeKuru	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL
13 KdvDahilNetSatisTutar_TL	decimal	no	9	19	8	yes	(n/a)	(n/a)	NULL

### Sql Not:

```
--SET NOCOUNT ON;  
--SET FMTONLY OFF;  
-- DECLARE @SQL NVARCHAR(MAX)  
  
--SET @SQL=( '  
--      '  
  
--EXEC SP_EXECUTESQL
```

```
--SELECT  
--@SQL;
```

```
--SET NOCOUNT ON;
```

- Sorgunun daha hızlı çalışmasını sağlıyor.
- Sql'le yapıştırırken yoruma alınır.
- Genelde çoğu raporun içerisinde yer alır.
- **Sql Server**'da her sorgu çalıştırdığımızda, sorgu sonucu, etkilenen satır sayısı ile birlikte, sorguyu çalıştıran uygulamaya geri gönderilir.
- Bazı durumlarda bu bilgi işimize yarasa bile, genellikle kullanmayız.
- **Sql Server**'ın bu bilgiyi hesaplamasını ve uygulamaya geri göndermesini engelleyerek, çok ufakta olsa kazanç sağlayabiliriz.
- Yapmamız gereken, sorgudan önce aşağıdaki komutu çalıştırmak olacaktır;

#### Sql Not:

```
SET NOCOUNT ON;  
SET FMTONLY OFF;  
SET ANSI_WARNINGS OFF;
```

```
SET FMTONLY OFF;
```

- Büyük tablolarınızı bir procedure içinde çağırıp kullanmadan önce **SQL** sorgusunun o tablonuz hatasız çalıştığını test etmenize olanak sağlar.
- Sorgunun hata almamasını önleyici kodlar.
- Genelde çoğu raporun içerisinde yer alır.

```
SET ANSI_WARNINGS OFF;
```

- ON olarak ayarlandığında SUM, AVG, MAX, MIN, STDSAPMA, STDSAPMAS, VAR, VARP, ya da COUNT toplama işlemleri boş değer verdiğinde bir uyarı mesajı oluşur eğer OFF olarak ayarlanmıyorsa bir hata verilmez.

## Temp Tablo Oluşturma

- Son Selectten sonraki ilk froma temp tablo yapmalıyız.
  - Union, Intersect, Except varsa **UNIONDAN ÖNCEKİ** ilk fromdan önce INTO yazmalıyız.
  - Çıktıda görmek istediğin tabloyu temp a
  - Sorgu çalıştırırken parametrelili kısımlara tarih gireceksek. DECLARE KISMINA değil, CREATE table kısmına tarihleri (değişkenleri girmeliyiz. BETWEEN varsa direk eşitleye de biliriz.
- 
- Son select ile, sorgunun aşağısında subquery varsa onu karıştırma. Subquery varsa extra 1 select daha olur.
  - Subquery varsa, fromdan sonra parantez vardır, o fromun gideceği tabloyu belirleyen bir sorgu.
  - Subquery de, Fromda parantez kapandıktan sonraki AS x **SELECT \* FROM x demek.** X tablosunu subqueryyle içeri yazmak demek.
  - Bunun amacı çok büyük tablolar için tabloyu daraltmak, örneğin perakende satış tablosu buna bir örnek (Bu çok büyük bir tablo,

satış datalarının fiş bazında, mağaza bazında, gün bazında tutulduğu.

- Bu nedenle bu oluşturduğumuz tabloda, yılaygününü, deposunu, işlemini kısıtlamış.

```
FROM Warehouse..Perakende_Satis ps (NOLOCK)
INNER JOIN #MerchYilHafta mh ON mh.yilaygun = ps.yilaygun
INNER JOIN #Depo d ON d.deporef = ps.deporef
INNER JOIN Warehouse.dbo.Dim_Islem (NOLOCK) i ON
i.IslemRef=ps.Islem
INNER JOIN #Urun u ON u.urunref = ps.urunref
WHERE i.lademiMi=0 and ps.yilaygun between @minyilaygun
and @maxyilaygun
GROUP BY mh.Merchyilhafta,d.DepoRef,SatisID
) AS x
GROUP BY x.Merchyilhafta, x.DepoRef
```

- Bizim burdaki çıktımız group by daki sütunlar olacak.
- Fromdan önce into atıp tabloya basıp tablonun içerisini görebiliriz.
- Tablolara isim verirken o tablolar sadece o sql sayfasını ilgilendireceği için kendi ismini verebilirsin.
- Tablo gelmezse bağlantıyı koparıp tekrar bağlamak sorunu çözebilir.

### Global Temp Tablo:

- Çift ## (diyez) koyarsak, global tablo olur. Yan sayfaya da geçtiğinde de çalışır.
- Tek # (diyez) sadece bu sayfada kullanabiliriz demek oluyor. İkisini birbirini vurmaz olur.

## İki Tabloyu Temp Tabloyu Sql'de Aynı Anda

### Çağırma

```
SELECT * FROM #burak WHERE deporef = 2055 AND Merchyilhafta = 202020
```

```
SELECT * FROM ##mayıs WHERE deporef = 2055 AND Merchyilhafta = 202020
```

### Not Sql:

- Temp tablo çalıştırmadan önce INTO'lu kısmı çalıştırıp içerisine basmalısın.
- Temp Tablonun Başında Select \* From varsa, sadece F5 ile tümünü seçip çalıştırman yeterli.

```
SELECT * FROM #burak WHERE deporef = 2055 AND Merchyilhafta = 202020
```

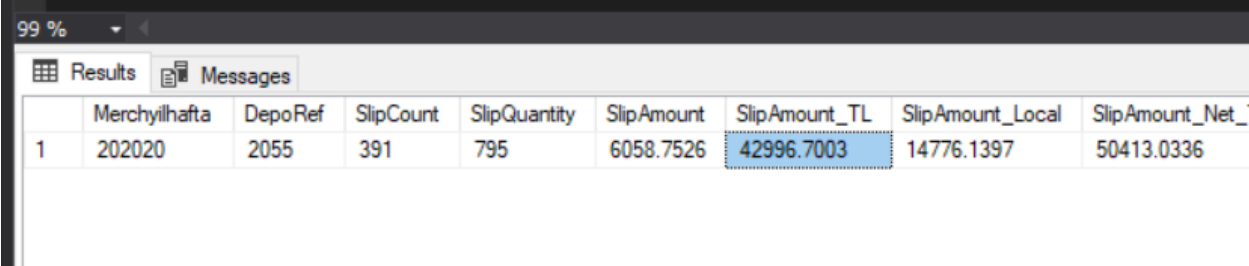
- # Temp Tablonun Başında Select \* From yoksa, Tablonun üstüne gelip, Control + 3 (ssf demek) veya Control + 4 ile çalıştırmalısın.

```
#burak
```

```
#merchyilhafta
```

### Not For Sql:

- Result simgesinin altındaki boşluğa sağ klikle tıklanıldığında, Copy with Headers ile excele yapıştırabiliriz.



	Merchyilhafta	DepoRef	SlipCount	SlipQuantity	SlipAmount	SlipAmount_TL	SlipAmount_Local	SlipAmount_Net_
1	202020	2055	391	795	6058.7526	42996.7003	14776.1397	50413.0336

