

SQL Komutları

SQL SELECT Kullanımı :SQL SELECT deyimi, veritabanı tablosundan verileri tablo şeklinde almak için kullanılır.

SELECT sutun1, sutun2, ..., sutunN

FROM tablo_adi

```
SELECT ProductName, UnitPrice, UnitsInStock FROM Products
```

sutun1, sutun2, ... değerleri istediğimiz tablonun alanlarını göstermektedir. Tüm satırları çekmek için yazılması gereken SQL SELECT komutu ise şu şekilde olmalıdır.

SELECT * FROM tablo_adi

```
SELECT * FROM Categories
```

SQL SELECT WHERE Kullanımı

SELECT WHERE deyimi, belirtilen şartla göre kayıtları süzmek ve yalnızca gerekli kayıtları almak kullanılır.

SELECT DISTINCT sutun1, sutun2,, sutunN
FROM tablo_adi
WHERE sart_ifadesi

WHERE ifadesi ile kullanılabilen operatörler şunlardır:

=	esittir
>	büyük
<	küçük
>=	büyük veya esittir
<=	küçük veya esittir
!=	eşit değildir

```
SELECT CustomerID, CompanyName, ContactName, City  
From Customers WHERE City = 'Madrid'
```

SQL LIKE Kullanımı

Sql sorgularında kullanılan **LIKE** komutu **WHERE** ifadesiyle beraber kullanılan ve veritabanındaki kayıtlarda belirli bir deseni aramak için sıkça kullanılan bir ifadedir. Desen içinde tek karakter için _ (alt tire), bir ve birden fazla karakter için ise %(yüzde) sembolleri kullanılır.

```
SELECT sutun1, ....sutunN  
FROM tablo_adi  
WHERE sutun_adi LIKE {desen}
```

```
SELECT * From Products  
WHERE ProductName LIKE 'C%'
```

Yukarıdaki sorguda ProductName alanında “C” ile başlayan kayıtlar geldi.

İçinde “C” geçen ProductName alanı için ise aşağıdaki şekilde bir sorgu yazılmalıdır.

```
SELECT * From Products  
WHERE ProductName LIKE '%C%'
```

SQL AND / OR Kullanımı

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

```
SELECT CustomerID, CompanyName, ContactName, City From Customers  
WHERE City = 'Madrid' OR City = 'London'
```

Yukarıdaki sorguda şehir alanı Madrid ve ya London olan kayıtlar listelenmiştir.

```
SELECT * FROM Employees  
WHERE HireDate >= '1993-01-01' AND HireDate < '1994-01-01'
```

Yukarıdaki sorgu ise HireDate tarihi 1993 ten büyük ve 1994 ten küçük olanları (yan, sadece 1993 te işe başlayanları) sıralamaktadır. İki şartında gerçekleşmesi istendiğinde AND ifadesi kullanılmalıdır.

SQL BETWEEN VE IN KULLANIMI

BETWEEN deyimi SELECT ve WHERE deyimleri ile birlikte kullanılan bir deyimdir. BETWEEN deyimi bir veri listesindeki bir alana ait verilerin belli bir alan aralığında veya belli bir değer aralığında olanlarının gösterilmesi için kullanabiliriz.

```
SELECT * FROM Products  
WHERE UnitPrice BETWEEN 75 AND 100
```

Yukarıdaki sorguda fiyatı 75 ve 100 arasında olan ürünler listelenmiştir.

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

```
SELECT * FROM Products WHERE CategoryID IN(1,4,8)  
ORDER BY CategoryID
```

Yukarıdaki sorgu CategoryId alanı 1,4, ve 8 olan kayıtları listelemektedir.

SQL ORDER BY Kullanımı

SELECT ORDER BY İfadesi ile kayıtları belirtilen sütuna göre artan yada azalan sırada getirmek için kullanılır. Order by ile bir yada daha fazla sütuna göre sıralama yapmak mümkündür.

```
SELECT sutun1, sutun2, sutunN  
FROM tablo_adi  
ORDER BY sutunA ASC, sutunB DESC
```

```
SELECT ProductName, UnitPrice, UnitsInStock FROM Products ORDER BY UnitPrice DESC
```

ORDER BY ifadesinin en sonunda kullanılan ASC ifadesi artan DESC ifadesi ise azalan şekilde sıralama yapmamızı sağlar. Yukarıdaki sorguda UnitPrice alanı azalan şekilde sıralanmaktadır.

SQL Is Null ve Is Not Null Kullanımı

NULL içerisinde herhangi bir değer bulundurmayan sütunlardır. Yani kayıt işlemi sırasında bir sütuna değer girilmezse o sütunun değeri **NULL** olarak adlandırılır. Sütuna boşluk girilirse sütun boş görülse dahi o sütun **NULL** olmaktan çıkacaktır. SQL de **NULL** değer içeren kayıtları sorgulamada karşılaştırma operatörü kullanılmaz. Eğer bir tabloda **NULL** değer içeren kayıtlar bulunmak isteniyorsa sorguda **IS NULL** ifadesi kullanılır. Aynı şekilde **NULL** değer içermeyen kayıtlar listelenmek isteniyorsa da **IS NOT NULL** ifadesi kullanılır.

```
SELECT FirstName, LastName FROM Person.Person  
WHERE Title IS NULL
```

```
SELECT FirstName, LastName, Title FROM Person.Person  
WHERE Title IS NOT NULL
```

İlk sorgu Title alanı NULL olanları, ikinci sorgu ise Title alanları NULL olmayan kayıtları getirecektir.

SQL'de AS Kullanımı

Bazı durumlarda sorgulama yaparken sütunun ismi kullanıcı tarafından anlaşılması zor olabilir bu yüzden daha anlamlı sütun başlıklarını kullanmak isteyebiliriz bu durumda Alias (AS) takma isimler kullanırız.

AS yardımcı kelimesini kullanarak sütunları farklı adlarla görüntüleyebiliriz.

Uzun tablo isimlerini de kullanımı daha kolay olacak şekilde değiştirebiliriz.

```
SELECT CategoryID AS [KATEGORİ NO], CategoryName AS [KATEGORİ ADI], FROM Categories
```

SQL Case ve When/Then Kullanımı:

Sql cümlelerimiz içinde belirli durumlara göre farklı işlemler yapmak istiyorsak Case-When yapısını kullanabiliriz. Kullanımı;

```
CASE  
WHEN durum1 THEN yapılacaklar  
WHEN durum2 THEN yapılacaklar  
ELSE yapılacaklar  
END
```

şeklindedir.

```
Select ProductName, UnitPrice,  
CASE UnitsInStock WHEN 0 THEN 'Stokta Yok'  
ELSE CONVERT(varchar, UnitsInStock) END  
FROM Products  
WHERE UnitPrice > 50
```

Yukarıdaki sorguda fiyatı 50 nin altında olan ürünlerden stok adeti 0 olanlarda ‘STOKTA YOK’, stok adeti 0 olmayanların ise stok adeti yazdırılmıştır.

SQL INNER JOIN Kullanımı:

INNER JOIN en çok kullanılan tablo birleştirme yöntemidir. İki tablo arasında birleştirme yaparken, tabloların her ikisinde de yer alan değerler seçilir, tablolardan sadece birinde yer alıp diğerinde ilişkili değere rastlanılmayan satırlar seçilmez.



SQL LEFT JOIN Kullanımı

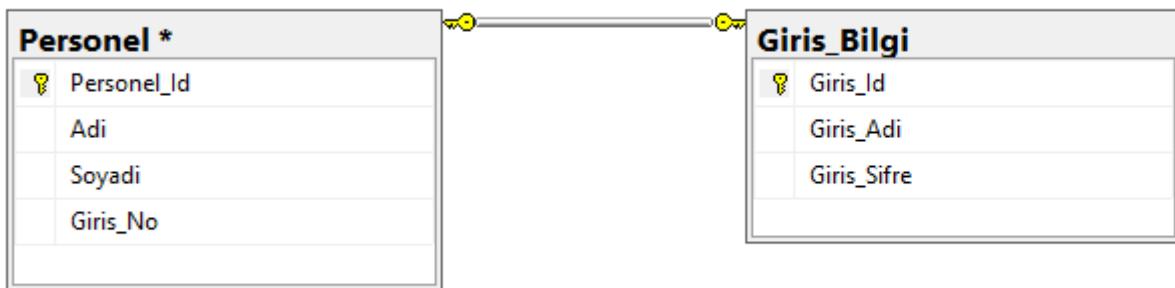
LEFT JOIN ile iki adet tablomuzdaki kayıtları belli bir kritere göre birleştirebiliriz. Burada asıl olan birinci tablodaki kayıtlardır. İkinci tablodan sadece birinci tabloda olan kayıtlar alınır. İkinci tabloda olup da birinci tabloda olmayan alanların değeri boş (NULL) olarak gelecektir

Primary Key (Birincil Anahtar)

- Bir sütundan oluşabileceği gibi birden fazla sütundan da Primary key oluşturulabilir. Bir tabloda tek bir Primary key bulunabilir.
- Tablolarda yer alan kayıtları birbirinden ayırt etmek için kullanılır.
- Birincil anahtar olan sütun NULL(boş) değerleri ve birbirinin aynı verileri içeremez ve benzersiz(UNIQUE) olmalıdır.

(T.C. Kimlik No, Öğrenci Numarası vs.)

Foreign Key (Yabancı Anahtar)



- Yabancı anahtar bir sütun veya birden fazla sütunun birleşiminden oluşabilir.
- Yabancı anahtar sütunu aynı tablo veya başka bir tablodaki birincil anahtar olan bir sütunla ilişkilendirilebilir.
- Birbiri arasında ilişki bulunan iki tablodan birisinden diğer tablodaki birincil anahtara başvuran sütun yabancı anahtar olarak adlandırılır.
- İkincil Anahtar olarak da bilinir.

TARİHSEL FONKSİYONLAR

GETDATE() FONKSİYONU

```
SELECT GETDATE() // İÇİNDE OLUNAN TARİHİ VERİR
```

DATEADD() FONKSİYONU

DATEADD fonksiyonu ile bir tarih üzerine ekleme yapabiliz. Bu fonksiyon 3 parametreye ihtiyaç duymaktadır. Tarihin hangi kısmına ekleme yapılacağı (yani ay mı, gün mü, yıl mı bilgisi), eklenecek değer ve geçerli bir tarih verisi.

```
DECLARE @t DATETIME  
SET @t = '2019-03-15 10:00:00.000'  
SELECT DATEADD(DAY, 1, @t)
```

Yukarıdaki sorgu @t isimli tarih değişkenine 1 gün ekleyecektir. Gün eklenebildiği gibi dakika, saat, ay ve yıl DATEADD fonksiyonu ile eklenebilir.

DATEDIFF() FONKSİYONU

DATEDIFF fonksiyonu ile SQL'de iki tarih arasındaki farkı buluruz. Bu fonksiyon 3 parametreye ihtiyaç duyar. İlk parametre tarihin hangi kısmı, yani ay, gün, yıl bilgisi. İkinci kısım iki tarih arasında önce gelen tarih, sonraki parametre ise sonra gelen tarih olur.

```
DECLARE @t1 DATETIME  
DECLARE @t2 DATETIME  
SET @t1 = '2019-03-15 10:00:00.000'  
SET @t2 = '2019-03-19 10:30:00.000'  
SELECT DATEDIFF(DAY, @t1, @t2)
```

Yukarıdaki sorgu @t1 adlı tarih değişkeni ile @t2 adlı tarih değişkeni arasındaki farkı gün cinsinden verecektir. DATEIFF sorgusunda fark hesaplanırken dakika,gün,ay ve yıl kullanılabilmektedir.

DATEPART() FONKSİYONU

DATEPART fonksiyonu seçilen bir tarihin sadece gün, ay, yıl şeklinde parçalanıp alınmasını sağlar.

```
DECLARE @t1 DATETIME  
SET @t1 = '2019-03-15 10:00:00.000'  
SELECT DATEPART(DAY, @t1)
```

Yukarıdaki sorgu @t1 tarih değişkeni içinden yalnızca gün bilgisini bize geri döner.

DAY() FONKSİYONU

DAY fonksiyonu seçilen bir tarihin sadece gün bilgisinin alınmasını sağlar.

MONTH() FONKSİYONU

MONTH fonksiyonu seçilen bir tarihin sadece ay bilgisinin alınmasını sağlar.

YEAR() FONKSİYONU

YEAR fonksiyonu seçilen bir tarihin sadece yıl bilgisinin alınmasını sağlar.

```
DECLARE @t1 DATETIME  
SET @t1 = '2019-03-15 10:00:00.000'  
SELECT YEAR(@t1) AS YIL, MONTH(@t1) AS AY, DAY(@t1) AS GÜN
```

DAY() MONTH() ve YEAR() fonksiyonlarının kullanımı yukarıda görülmektedir.

STRING FONKSİYONLAR

LEFT() Fonksiyonu

Verilen metinin soldan belirtilen sayıda karakterini alır

```
DECLARE @text varchar(50)  
SET @text = 'İstanbul Gelişim Üniversitesi'  
  
SELECT LEFT(@text,7)
```

RIGHT() Fonksiyonu

Verilen metinin sağdan belirtilen sayıda karakterini alır

```
DECLARE @text varchar(50)  
SET @text = 'İstanbul Gelişim Üniversitesi'  
  
SELECT RIGHT(@text,7)
```

LEN() Fonksiyonu

Verilen metinin karakter sayısını verir.

```
DECLARE @text varchar(50)
SET @text = 'İstanbul Gelişim Üniversitesi '
SELECT LEN(@text)
```

TRIM() Fonksiyonu

Verilen metinin başındaki ve sonundaki boşlukları siler

```
DECLARE @text varchar(50)
SET @text = 'İstanbul Gelişim Üniversitesi '
SELECT LTRIM(@text)
Soldaki boşlukları siler
SELECT RTRIM(@text)
Soldaki boşlukları siler
```

CHARINDEX() Fonksiyonu

Verilen metinin içerisinde herhangi başka bir metin aramayı sağlar.

```
DECLARE @text varchar(50)
SET @text = 'İstanbul Gelişim Üniversitesi'

SELECT CHARINDEX('gelişim',@text,1)
```

İlk önce aranacak ifade verilir, daha sonra içinde arama yapılacak değişken ve aramaya bu değişkenin hangi indexinden başlanacağı belirtilir..

REPLACE() Fonksiyonu

Verilen metin içinde istenilen karakterlerin başka bir karakterle değiştirilmesini sağlar

```
DECLARE @text varchar(50)
SET @text = 'İstanbul Gelişim Üniversitesi'
SELECT REPLACE(@text, 'a', 'e')
```

İlk olarak hangi değişkende kullanılacağı ikinci olarak değişecek ifade ve son olarak değişen ifadenin yerine gelecek ifade seçilir

STR() Fonksiyonu

Verilen karakteri istenilen karakter uzunlığında yazar.

```
DECLARE @text varchar(50)
SET @text = 'İstanbul Gelişim Üniversitesi'
SELECT STR(5,8)
```

beş sayısı 8 basamak halinde yazılır

SPACE() Fonksiyonu

Kaç karakterlik boşluk üretileceğini söyler

```
SELECT SPACE(10)
```

SEÇİMSEL FONKSİONLAR

IIF() Fonksiyonu

CASE WHEN kontrolü yapılarak formatlandırılmış sorgularda daha pratik ve kullanışlı çalışma sergileyebilmemiz için SQL Server 2012 sürümüyle birlikte IIF fonksiyonu gelmiş bulunmaktadır.

```
SELECT ProductName, UnitPrice, IIF(UnitPrice > 10, 'Pahalı', 'Ucuz') AS [STATUS]  
FROM Products
```

Bu sorgu Products tablosundan ürünün fiyatı 10 liranın üstündeyse pahalı değilse ucuz şeklinde yazdırarak sonuçları getirir.

CHOOSE() Fonksiyonu

SQL 2012 ile birlikte gelen yeni bir fonksiyondur. Bu fonksiyon ile birlikte bir diziden istenilen sıradaki yani istenilen indeks numarasındaki verinin geri döndürülmesi sağlanır. Kullanımı;

```
SELECT CHOOSE(1, FirstName, LastName, CONCAT(FirstName, ' ', LastName))  
AS PERSON FROM Employees
```

DÖNÜŞÜM FONKSİYONLARI

CONVERT() Fonksiyonu

```
SELECT CONVERT(hedef_veri_tipi, alan_adi, gosterim_formatı)  
FROM tablo_ad
```

Şeklinde kullanılır.

```
SELECT CONVERT(varchar, 5) + 'elma'
```

PARSE() Fonksiyonu

```
SELECT PARSE('5000' AS int)
```

şeklinde kullanılır.

SQL GROUP BY KULLANIMI

COUNT(*) Fonksiyonu

Toplam ürün sayısını verir

```
SELECT COUNT(*) FROM Products
```

MAX() Fonksiyonu

Verilen tablodaki en yüksek değeri getirir

```
SELECT MAX(UnitPrice) FROM Products
```

MIN() Fonksiyonu

Verilen tablodaki en düşük değeri getirir

```
SELECT MIN(UnitPrice) FROM Products
```

COUNT, MAX ve MIN AGGREGATE fonksiyonlardır.

GROUP BY ifadesi ise gruplama yapar. Yani sonuç kümesini bir veya birden fazla kolona göre grupper.

```
SELECT CategoryId, COUNT(0) AS [COUNT] FROM Products
GROUP BY CategoryID
```

SUM() ve AVG() KULLANIMI

```
SELECT SUM(ListPrice) FROM Production.Product
WHERE ListPrice > 0
```

Bu sorgu, SUM() fonksiyonu sayesinde Production.Product Tablosunda fiyatı 0'dan büyük olan ürünlerin toplam fiyatını verir.

AVG() Sorgusu ise kendisine verilen değerin ortalamasını alır

```
SELECT AVG(ListPrice) FROM Production.Product
WHERE ListPrice > 0
```

HAVING KULANIMI

HAVING yapısı temelde WHERE ile aynı görevi yapmaktadır. GROUP BY ile kullanılır. WHERE ifadesi ile belirtilen kriter GROUP BY uygulanmadan önce geçerli olurken, HAVING ifadesi ile belirtilen kriter ise GROUP BY uygulandıktan sonra ortaya çıkan verileri filtrelemek için kullanılır. Ayrıca WHERE ifadesinden sonra SUM, AVG gibi fonksiyonlar kullanılamazken, HAVING ile kullanılabilir.

```
SELECT p.ProductID, Name, SUM(p.ListPrice * i.Quantity)
AS Summary From Production.Product p
INNER JOIN Production.ProductInventory i On p.ProductID = i.ProductID
GROUP BY p.Name,p.ProductID
```

Örneğin yukarıdaki sorgu için bir WHERE ifadesi kullanamayız. Ancak filtrelemeyi HAVING ile yapabiliriz.

```
SELECT p.ProductID, Name, SUM(p.ListPrice * i.Quantity)
AS Summary From Production.Product p
INNER JOIN Production.ProductInventory i On p.ProductID = i.ProductID
GROUP BY p.Name,p.ProductID
HAVING SUM(p.ListPrice * i.Quantity) > 5000
```

DISTINCT KULLANIMI

Bazen bir tablonun bazı kolonlarında tekrarlanan değerler (veriler) olabilir. Örneğin aşağıdaki tabloda şehir alanına bakarsanız *İstanbul* verisinin iki kez bulunduğu görüşünüz. Elbette bu gayet normal bir durumdur. Ancak tekrarlanan verileri eleyerek her farklı veriden yalnız bir adet bulunmasını istiyorsak DISTINCT anahtar sözcüğünü kullanırız.

ID	Soyad	Ad	Şehir
1	Mustafa	Can	İstanbul
2	Hasan	Demir	İstanbul
3	Ali	Kara	Ankara

Eğer yukarıdaki tabloda yalnızca farklı şehirleri seçmek istiyorsak:

```
SELECT DISTINCT Şehir FROM Kişiler
```

ifadesini kullanmalıyız.

```
USE Northwind
SELECT Country From Employees
```

Northwind veritabanında yapılan sorguda çalışanların ülkeleri listelenmiş ve 2 ülke 9 kolonda yazılmıştır. Burada tekrar eden verileri silmek için DISTINCT ifadesi kullanılmalıdır.

```
USE Northwind
SELECT DISTINCT Country From Employees
```

Verilerin Dikey Birleştirilmesi: UNION

```
SELECT 'Hasan' AS NAME
SELECT 'Ali'
SELECT 'Veli'
SELECT 'Mehmet'
```

Bu verileri UNION ile birleştirmek için

```
SELECT 'Hasan' AS NAME
UNION
SELECT 'Ali'
UNION
SELECT 'Veli'
UNION
SELECT 'Mehmet'
```

Gerçek veriler üzerinden, Northwind veri tabanında bir UNION işlemi yapılacak olursa;

```
Use Northwind
SELECT FirstName, LastName, HomePhone From Employees
UNION
SELECT ContactName, Phone FROM Customers
```

Yukarıdaki sorgu bize hata verir. UNION işleminde birleştirme yapılan kolon sayıları aynı olmak zorundadır. Bu durumda bu sorgu şu şekilde yazılabilir.

```
SELECT FirstName + ' ' + LastName, HomePhone FROM Employees  
UNION  
SELECT ContactName, Phone FROM Customers
```

SINIRLI SAYIDA VERİ LİSTELENMESİ TOP() KULLANIMI

TOP() ifadesi döndürülecek kayıt sayısını belirtmek için kullanılır. Bildiğiniz üzere SELECT ifadesi yalnız başına kullanıldığında veritabanında ki tüm verileri seçiyordu. Doğal olarak performans açısından milyonlarca verinin olduğu veritabanlarında sıkıntı çıkaracaktır. Bu yüzden SELECT TOP ile belli bir veri miktarının seçilmesini sağlayabiliyoruz.

```
SELECT * FROM Orders
```

ifadesi ile yapılan sorguda görüldüğü üzere 830 satır oluşmuştur.

TOP() ifadesi ile ise istediğimiz sayıda sonuç elde edilebilir.

```
SELECT TOP 10 * FROM Orders
```

```
SELECT TOP 20 PERCENT * FROM Products
```

şeklinde bir ifade ise bize kesin sayı değil toplam sonuç üzerinden yüzde değeri şeklinde bir sonuç getirecektir.

CTE (Common Table Expression)

SQL Server 2005 ile birlikte sunulan CTE (Common Table Expression – Ortak Tablo İfadeleri), bir sorgunun yürütülmesi anında elde edilmiş geçici sonuçların bir veya daha fazla kullanılmasını sağlayan ifadelerdir. CTE bir table veya View olmayıp sadece bir sorgu ifadesidir. Yani geçici ve kalıcı tablolar gibi herhangi bir veri içermezler. Bir CTE'nin basit yazım biçimi aşağıdaki gibidir;

```
WITH expression_name [ ( column_name [,...n] ) ]  
AS  
( CTE_query_definition )
```

Northwind üzerinde bir CTE kullanımı.

```
;WITH Urunler(ProductID, Summary) AS  
SELECT ProductID, SUM(Quantity) AS Summary FROM [Order Details]  
GROUP BY ProductID  
)  
  
SELECT p.ProductName, u.Summary FROM Products p  
INNER JOIN Urunler u ON u.ProductID = p.ProductID  
ORDER BY u.Summary DESC
```

INTERSECT EXCEPT (KESİŞİM VE FARK KÜMELERİ) KULLANIMI

Except komutu iki data setimizden ilk datasetinde olup ikinci datasetinde olmayan yeni bir dataset verir bize.

Örnek olarak A tablosunda 1,2,3 kayıtları olsun. B tablosunda ise 3,4 kayıtları olsun.

Aşağıdaki gibi kodumuzu yazdığımızda yeni oluşacak dataset üzerinde sadece 1,2 kayıtları olacaktır.

```
Select Id From A  
Except  
Select Id From B
```

Intersect komutu ise iki tablo arasındaki kesişim değerlerini almaktadır. A ve B tablosu için düşünürsek geri dönen kayıt sadece 3 değeri olacaktır.

```
Select Id From A  
Intersect  
Select Id From B
```

CRUD İŞLEMLERİ

--C-> Create -> INSERT
--R-> READ -> SELECT
--U-> UPDATE -> UPDATE
--D-> DELETE-> DELETE

SQL'de veritabanına kayıt eklemek için **INSERT INTO** deyimini kullanırız.

INSERT INTO deyiminin formatı şu şekildedir:

INSERT INTO [tablo adı](alan adları) VALUES (veriler)

Bu formata göre INSERT INTO deyiminden sonra kayıt eklemek istediğimiz tablonun adını yazıyoruz. Tablo adından sonra parantez içerisinde tabloda veri eklemek istediğimiz alanları belirtiyoruz. Son olarak sıralı bir şekilde alanlara karşılık gelen verileri yazıyoruz. Burada tablo adından sonra alanları belirtmezsek, tabloda bulunan bütün alanlara veri gireceğimiz anlamına gelir.

Örneğin Northwind Veri tabanında OrderDetails tablosuna veri eklemek için

```
INSERT INTO [Order Details] VALUES(10253,3,11,10,0)
```

komutu kullanılır.

Bir diğer kullanımı veri girilecek alanların belirlilerek kullanılmasıdır.

```
INSERT INTO Employees (FirstName, LastName,Country, HireDate, BirthDate)  
VALUES('Eren','Efe','Turkey','1999-01-15','1981-01-01')
```

Özetlersek; INSERT INTO tablolara veri eklemek için kullanılıyor. İki şekilde kullanımı mevcut. Birinde alan adı belirtilmeksızın bütün alanlara veri giriliyor. İkincisinde ise sadece belirli alanlara veri giriliyor.

```
INSERT INTO Employees (FirstName, LastName, Country, HireDate, BirthDate)  
VALUES('Eren', 'Efe', 'Turkey', '1999-01-15', '1981-01-01')
```

UPDATE

UPDATE ifadesi tablomuzda bulunan kayıtları güncellemek yani değiştirmek için kullanılır.

UPDATE ifadesinden sonra işlem yapılacak tablo belirtiler daha sonra SET ifadesi kullanılır ve Update etmek istenilen kolon adı yazılır. Ancak Update işlemlerinde filtre kullanılmazsa bütün tablo bu işlemden etkileneceği için oldukça dikkatli olunmalıdır.

```
UPDATE Products SET UnitPrice = 20 WHERE ProductID = 5
```

Yukarıdaki sorgu Northwind veritabanında Products tablosundaki 5 id'li ürününün fiyatını 20 olarak güncelleyecektir.

DELETE İŞLEMİ

DELETE ifadesinden sonra işlem yapılacak tablo yazılır ve daha sonra silinmek istenen kayıt seçilir. Aynı UPDATE işleminde olduğu gibi filtre kullanılmazsa tablo içindeki bütün kayıtlar silineceği için oldukça dikkatli olunmalıdır.

```
DELETE FROM Categories WHERE CategoryID = 1
```

Yukarıdaki örnekte eğer ilişkili bir kayıt yoksa yani belirtilen CategoryId başka bir tablodaki bir kayıt tarafından kullanmıyorsa 1 nolu Category silinecektir.

SQL GÜVENLİK NOTLARI

Bir veritabanı sistemine **girebilmek** için ihtiyacımız olanlar;

1. Fiziksel bağlantı (Network)
2. Açık port (SQL SERVER'ın default portu 1443'tür.)
3. Kullanıcı adı
4. Şifre

BRUTE FORCE ATTACK

Tam çevirisi kaba kuvvet saldırısıdır. Aslında bu sadece SQL server sistemine değil birçok sisteme saldırı yapmak için kullanılmaktadır. Amaç belli şifre kombinasyonlarını deneyerek SQL SERVER'a bağlanmaktadır.

Brute Force, bir parolayı ele geçirmek için yapılan bir çeşit dijital ve kriptografi saldırısıdır. Brute-Force tekniğinde elde herhangi bir bilgi bulunmuyor olmasına karşın belli şifreler denenerek doğru şifreye ulaşmaya çalışılır.

SQL sistemine Brute Force saldırısı yapmak için nelere ihtiyaç var

--Sunucunun IP adresinin bilinmesi

--Bağlantı portunun açık olması(1433)

--SQL Server kullanıcı adının bilinmesi

Brute-Force yönteminde başarıya ulaşabilme şansını etkileyen birçok önemli faktör vardır. Saldırıyı gerçekleştiren kişilerin ellerinde ne kadar kuvvetli ve kapsamlı bir şifre listesinin olduğu, kullanıcının ne zorlukta bir şifte kullandığı ve hatta sistemin bu yöntemi denemeye ne olsaklarda açık olduğu gibi birçok önemli detay Brute-Force saldırısı için önemli birer maddedir.

Kullanıcı Tanımlı Fonksiyonlar(User Defined Function)

Kullanıcı Tanımlı Fonksiyonlar, parametreli parametresiz olarak birçok dilde aslında mevcuttur. Kullanıcı Tanımlı Fonksiyonlar, SQL fonksiyonları dışında kullanıcılar tarafından oluşturulan ilişkisel veri tabanı nesneleridir. SQL Server 2000 ile birlikte gelen Kullanıcı Tanımlı Fonksiyon oluşturma işlemi, tek bir değer veya tablo döndürmek için kullanılabilir.

Sayısal Değerli Fonksiyonlar (Scalar-Valued Functions):

Tek değer döndüren fonksiyon çeşididir. Bazı durumlarda kullanıcıların tercih ettiği bir tiptir. Tek değer döndürmek ile anlatılmak istenilene GETDATE() sistem fonksiyonu örnek olabilir. GETDATE() fonksiyonu çalıştırıldığında sisteminin zaman ve saatini döndüren yani tek değer döndüren bir sayısal değerli sistem fonksiyonudur.

Yazımı:

```
CREATE FUNCTION fonksiyonAdi (parametre)
RETURNS int
AS
Begin
    Sorgu
    RETURN int
End
Go
```

Tablo Döndüren Fonksiyonlar(Table-Valued Functions)

Sayısal değerli fonksiyondan tek farkı döndürdüğü verinin tablo olmasıdır. Viewlerle büyük benzerlikler içerir ancak farklı olarak dışarıdan parametre alabilirler.

Yazımı:

```
CREATE FUNCTION fonksiyonAdi (parametre)
RETURNS Tablo
AS
    Sorgu
GO
```

SQL SERVER AUTHENTICATIONS

Sql Server ile kullanıcılar arasında iki çeşit kimlik doğrulama yöntemi vardır. Windows Authentication ve Sql Server Authentication.

Windows Authentication bağlantı türünde, SQL SERVER'a erişmek için geçerli bir Windows kullanıcısı gerekmektedir.

Sql Server Authentication bağlantı türünde ise SQL SERVER'a erişebilmek için geçerli bir SQL SERVER kullanıcısı ve şifresi gerekmektedir.

SQL SERVER SUNUCU TÜRLERİ

Database Engine: Verileri depolamak, işlemek ve güvence altına almak için kullanılır.

Analysis Services: Çevrimiçi Analitik İşleme ve veri madenciliği işlevleri için kullanılır.

Reporting Services: İlişkisel, çok boyutlu veya XML tabanlı veri kaynaklarından etkileşimli, tablo, grafik veya serbest formlu raporlar oluşturmak için kullanılır.

Integration Services: Çok çeşitli veri taşıma görevlerini yapmak için kullanılır. Veri entegrasyonu ve iş akışı uygulamaları için bir platformdur.

BACKUP RESTORE

Sql Server üzerinde 3 farklı backup çeşidi bulunmaktadır.

Full Backup: Adından anlaşılacağı üzere yedeklenme anı itibarıyle veritabanındaki her şey yedeğin içine kopyalanır. Elimizde bulunan bir full backup dosyası ile başka bir şeye ihtiyaç duymadan yedeğin alındığı tarihe veritabanı geri yüklenebilir.

Differential Backup: Çok fazla verinin bulunduğu büyük veri tabanlarını sürekli full backup ile yedeklemek, hem uzun işlem süresi hem de gereksiz disk kullanımı neden olur. Böyle durumlarda sadece değişen kısımların yedeğinin alınması işlemidir.

Transaction Log Backup: Transaction, ilgili veritabanında yapılan her işlemin bilgilerinin tutulduğu log dosyasıdır. Yedekleme sadece bu kayıtları kapsar.

DB MAIL

İlk olarak Microsoft SQL Server 2005'te ortaya çıkan bir özellik olan Database Mail (Veritabanı Postası), SQL Veritabanı üzerinden e-posta göndermek için kullanılan bir araçtır. SQL Server'in daha önceki sürümlerinde bulunan SQLMail yapısına göre daha gelişmiş özelliklere sahiptir. SQLMail'de e-posta gönderme protokolü olarak MAPI (Messaging Application Programming Interface - Mesaj Gönderme Uygulaması Programlama Arayüzü) kullanılır. Database Mail'de ise, SMTP (Simple Mail Transfer Protocol - Basit Posta Aktarım Protokolü) kullanılmaktadır.

SERVER AGENT

SQL serverda periyodik olarak otomatik olarak yerine getirilmesini istediğimiz (yedek alma, sistem akımları, çeşitli maillerin gönderilmesi vs) gibi işlemleri yerine getirmek için SQL üzerinde kullandığımız araçtır.

SQL VIEW

View'lar sorguları basitleştirmek, erişim izinlerini düzenlemek, farklı sunuculardaki eşdeğer verileri karşılaştırmak veya bazı durumlarda sorgu süresini kısaltmak için kullanılan, gerçekte olmayan SELECT ifadeleri ile tanımlanmış sanal tablolarıdır.

View bir tablonun sanal görüntüsüdür. Gerçekte içerisinde bir veri olmayan, ancak içeriğinde belirtilen SQL sorgu bloğunun getireceği verileri kendisine verilen kısa bir ad ile kullanıcıya tablo olarak göstermek için tasarılmıştır.

SELECT ifadelerinin uzun ve **JOIN** gibi karmaşık sorguları içерdiği durumlarda, bu sorguların kolay anlaşılabilir, kullanılabilir ve yönetilebilir olması için View'lar kullanılır.

View kullanmanın önemli nedenlerinden bir tanesi sağladığı güvenlidir. Örneğin tablolarınızın tamamının görünmesini istediğiniz zamanlarda sanal tablo kullanıp tablolarınızın tamamının görünmesini engelleyebilirsiniz

Karmaşık sorguları basitleştirmek, sorgu süresini kısaltmak ve ağ üzerindeki trafiği düşürmek, erişim izinlerini düzenlemek ve farklı sunuculardaki benzer verileri karşılaştırmak içinde kullanılır.

View yazımı

```
CREATE VIEW view_adi
```

```
AS
```

```
SELECT sütun_adları
```

View'lar üzerinde değişiklik yapmak için ALTER ifadesi kullanılır.

Örneğin aşağıda Northwind veritabanı üzerinde Employess tablosu için oluşturulmuş bir View örneği bulunmaktadır.

```
CREATE VIEW vm_Employess
AS
SELECT EmployeeID, FirstName, LastName
FROM Employees
```

Artık vm_Employess isimli View'imizi bir tablo gibi sorgulayabiliriz

The screenshot shows a SQL query window titled 'SQLQuery1.sql - (...-2HBFUUP\Eren (52))'. The query is 'SELECT * FROM vm_Employess'. The results pane displays a table with 10 rows, each containing EmployeeID and FullName. The data is as follows:

EmployeeID	FullName
1	Nancy Davolio
2	Andrew Fuller
3	Janet Leverling
4	Margaret Peacock
5	Steven Buchanan
6	Michael Suyama
7	Robert King
8	Laura Callahan
9	Anne Dodsworth
10	Bayraktar Ibrahim

Oluşturduğumuz viewlar veritabanının altında Views sekmesinin altında görüntülenebilir.

View'lar üzerinde değişiklik yapmak için ALTER ifadesi kullanılır. Yukarıda oluşturulan vm_Employess isimli View'da değişiklik yapmak için;

```
ALTER VIEW vm_Employess
AS
SELECT *
FROM Employees
```

View tanımlarken de WHERE ile filtreleme işlemleri yapabiliriz. Yine Northwind veritabanı üzerinde Employess tablosu için WHERE parametresi ile bir View tanımlamak için;

```
CREATE VIEW vm_Employess2
AS
SELECT *
FROM Employees WHERE HireDate > '1993'
```

View'ları şifreyebiliriz. Bunun için kullanılması gereken ifade **WITH ENCRYPTION**'dur. Yukarıda tanımladığımız vm_Employess2 isimli View'ı şifrelemek için;

```
ALTER VIEW vm_Employess2
WITH ENCRYPTION
AS
SELECT *
FROM Employees WHERE HireDate > '1993'
```

Şifrelemeden sonra viewlar'ın yapısının başkaları tarafından görülmesi mümkün olmayacağındır.

View'ları silmek için DROP ifadesi kullanılır. Yukarıda tanımladığımız vm_Employess2 isimli View'ı silmek için;

```
DROP VIEW vm_Employess2
```

Sql View'in İçerisinde Yapılamayacak İşlemler Nelerdir?

- View içerisinde parametre gönderilemez.
- View yapısı içerisinde dml kodları (insert into, update, delete) kullanılamaz.
- View yapısı içerisinde, sadece select ile başlayan ifadeler kullanılabilir.
- View içerisinde order by (sıralama) fonksiyonu kullanılamaz.

SQL TRIGGER

Tetikleyici (Trigger) yapısı, ilişkisel veri tabanı yönetim sistemlerinde, bir tabloda belirli olaylar meydana geldiğinde veya gelmeden önce otomatik olarak çalışan özel bir Stored Procedure türüdür.

Bir tabloda ekleme, güncelleme ve silme işlemlerinden biri gerçekleştiğinde veya gerçekleşmeden önce, aynı tabloda veya başka bir tabloda belirli işlemlerin yapılmasını istediğimizde, Trigger yapısını kullanırız.

Örneğin; bir veri eklendiğinde, aynı ya da farklı bir tabloda, başka bir sütunun değerini değiştirmek gerekebilir. Örnek verecek olursak, satış tablosunda satış işlemi gerçekleştiğinde ürünün stok miktarının eksiltilmesi, banka hesabında işlem gerçekleştirildikten sonra otomatik olarak e-mail gönderilmesi gibi.

Triggerler, genellikle loglama ve güvenlik amaçlı kullanılırlar

Trigger yapısının en önemli unsurlarından biri sözde tablolardır. Sözde tablolar, **INSERTED** ve **DELETED** olmak üzere iki adettir.

Bu tablolar RAM üzerinde olarak bulunurlar. Gerçek veri tablosuna veri eklendiğinde, eklenen kayıt **INSERTED** tablosuna da eklenir. Tablodan bir kayıt silindiğinde ise silinen kayıt **DELETED** tablosunda yer alır.

Update anında ise her iki tabloya da veri gelir, yeni eklenen veri **INSERTED**'tan, eski veri **DELETED**'dan gelir. Güncellenmeden önce ki hali **DELETED** tablosunda güncellendikten sonra ki hali ise **INSERTED** tablosunda tutulur.

Tetikleyici (Trigger) Türleri

1- DML (Date Manipulation Language) Triggres

INSERT, UPDATE ve DELETE gibi veri okuma ve işleme için kullanılan DML ifadeleridir. DML tetikleyicileri, INSERT, UPDATE ve DELETE olayları kullanılarak veriler her değiştirildiğinde tetiklenir. DML tetikleyicileri iki tip olarak sınıflandırılır.

DML tetikleyicileri iki tip olarak sınıflandırılır.

- **AFTER tetikleyicileri:** Bir tabloda INSERT, UPDATE, DELETE işlemleri yapıldıktan belli işlemlerin yapılması için kullanılmaktadır.

- **INSTEAD OF tetikleyicileri:** Veri değişimi başlamadan hemen önce çalışırlar. Bir tablo veya view yapısında INSERT, UPDATE veya DELETE işlemlerini atlayıp, bunun yerine tetikleyici içinde tanımlanan diğer ifadeleri yerine getirmektedir.

2- DDL (Data Definition Language) Triggers

CREATE, ALTER, DROP komutlarını içerir. Veritabanında nesne oluşturmak, değiştirmek ve silmek için kullanılan bu komutlar için de trigger'lar tanımlanabilir.

DDL trigger'lar, sadece **AFTER** trigger olarak tanımlanabilirler. **INSTEAD OF** türden bir DDL trigger oluşturulamaz.

3- Logon Triggers:

Logon tetikleyicisi, SQL Server sunucusuna bağlanan bir kullanıcılarla belirli kısıtlamalar uygulayabiliriz. Logon tetikleyicileri, SQL Server'de güvenlik ve kontrol amaçlı kullanılmaktadır. Örneğin kullanıcı veritabanına dışarıdan bilinmeyen bir bilgisayarın bağlanması engelleyip, sadece şirket bilgisaylarından bağlanılmasına izin verilebilir.

Trigger Oluşturma

```
CREATE TRIGGER tetikleyiciAdı
ON tabloAdı
AFTER | FOR | INSTEAD OF komut
AS
BEGIN
    --Yapılması istenen işlemler
END
END
```

Trigger Değiştirme

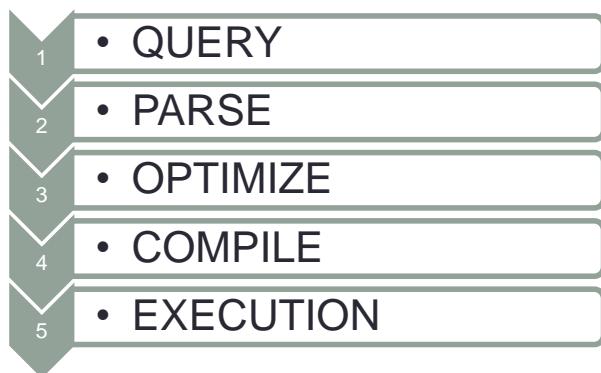
```
ALTER TRIGGER tetikleyiciAdı
ON tabloAdı
AFTER | FOR | INSTEAD OF komut
AS
BEGIN
    --Değişiklik Yapılan Trigger Kodları
END
```

STORED PROCEDURE

Stored Procedure, kod bloklarının, bir kez yazılarak derlendikten sonra SQL Server'da prosedür hafızasında tutulan, tekrar çalıştırıldıklarında ise, ilk sorgu sırasında oluşturulan sorgu ve çalışma planı hazır olduğu ve hafızada tutulduğu için daha hızlı sonuç döndüren SQL Server nesneleridir.

Stored Procedure'ler kısaca SPROC, SP ya da PR olarak isimlendirilirler

Sql Sorgularının Çalışma Sırası



Query: Bizim gönderdiğimiz sorgunun SQL Server tarafından alınması

Parse: Sorgunun doğru yazılıp yazılmadığı ve o anda bağlı olan kullanıcının bu sorguyu yazabilmek için yetkili olup olmadığını kontrol ediyor.

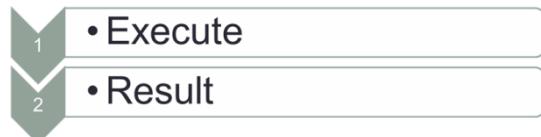
Optimize: Gönderilen sorguyu en iyi nasıl çalıştıracağına karar verir. Bir execution plan oluşturulur. hangi indeksin kullanılacağı vb

Compile: Execution Plan makine diline çevrilir. Yani kod derlenir.

Execute: Artık makine diline çevrilmiş yapı çalıştırılır.

Result: Sonuç gösterilir.

Stored Procedure Çalışma Sırası



Stored Procedure olduğu anda ilk dört adım tamamlanır ve bir Stored Procedure sonra tekrar çalıştırıldığında tekrar etmez.

İlk dört aşamayı atladığı için normal bir sql sorgusuna göre çok daha hızlı çalışır.

Stored Procedure'ların Faydaları

- Network trafiğini azaltarak performansı artırır.
- Sorgu sonuçlarını çok daha hızlı getirir.
- Stored Procedure'e bağlı işlemler için çağrılmak üzere olan işlem basamakları oluşturmakımıza sağlar.

- Birçok uygulamanın aynı Stored Procedure'u çalıştırmasıyla güvenlik ve tutarlılık daha iyi kontrol edilir.
- Bazı saldırıların etkisini azaltarak güvenlik yönüyle iyileşme sağlar.

Stored Procedure Türleri

1- Extended Stored Procedure

Eski bir programlama özelliği olduğu için genel olarak eski uygulamalarda görülebilir. Master veritabanında tutulurlar. C, C++ gibi diller ile geliştirilirler. Genellikle .dll şeklinde prosedürlerdir.

2- CLR Stored Procedure

SQL Server 2005 sonrasında CLR ortamında herhangi bir dili kullanarak kodlanan Stored Procedure'lerdir. .NET mimarisinin gücünü SQL Server içerisinde kullanmaya olanak tanıyan bu teknolojiyi

3- System Stored Procedure

Genellikle sp_ ön ekiyle başlarlar ve hepsi master veri tabanında tutulan Stored Procedure'lerdir. SQL Server içerisinde birçok özellik ve nesneler hakkında bilgi almak için kullanılır.

4- Kullanıcı Tanımlı SP

Kullanıcıların tanımladığı Stored Procedure'lerdir

STORED PROCEDURE OLUŞTURMA

```
CREATE PROCEDURE or CREATE PROC prosedür_adi
[ WITH Seçenekleri ]
AS
BEGIN
SQL ifadeleri
END
```

Sorgularınızı **AS** ifadesinden sonra **BEGIN...END** bloğu içerisinde de yazabilirsiniz.

Ancak SQL için **BEGIN...END** bloğunu kullanmak zorunlu değildir.

AS ifadesinden sonra sorgu yazılabılır.

Northwind veri tabanından Products tablosu kullanılarak oluşturulan Stored Procedure;

```
CREATE PROC pr_UrunGetir
AS
BEGIN
SELECT * FROM Products
END
```

Stored Procedure'nin kaynak kodlarına ulaşmak için;

sp_helpTEXT yazılır ve daha sonra Stored Procedure'nin adı yazılarak kaynak kodlar görülebilir.

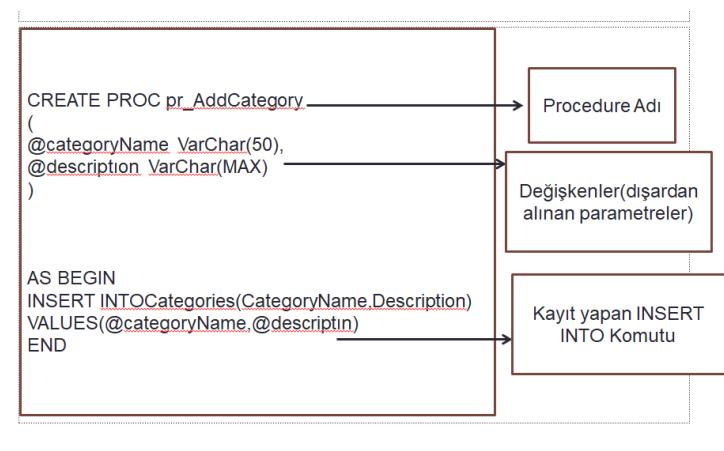
```
sp_helpTEXT 'pr_UrunGetir'
```

Oluşturduğumuz Stored Procedure'ları Database altındaki Programmability menüsünün altında Stored Procedures bölümünden görebiliriz.

Stored Procedure yapısı genelde veri çekmek için değil veri ekleme güncelleme ve silme işlemleri için kullanılmaktadır.

Insert Stored Procedure Örneği

Northwind veritabanında Categories tablosuna ver ekleyen bir Stored Procedure için;



Yukarıdaki Stored Procedure yapısında Categories tablosuna kayıt eklemek içi iki adet parametre (@categoryName ve @description) kullanılmıştır. Stored Procedure'lar dışardan parametre alabilirler.

Ayrıca INSERT INTO yapısında kayıt olacak alanların sıralaması nasıl verildiyse VALUES alanında belirtilen sıralama aynen takip edilmek zorundadır.

```
INSERT INTOCategories(CategoryName,Description)
```

```
VALUES(@categoryName,@descriptin)|
```

Yukarıda tanımladığımız Stored Procedure aşağıdaki şekilde çağrılarak kayıt işlemi gerçekleştirilecektir;

```
_pr_AddCategory 'Yeni Kategori', 'Yeni Eklenen İlk Kategori'
```

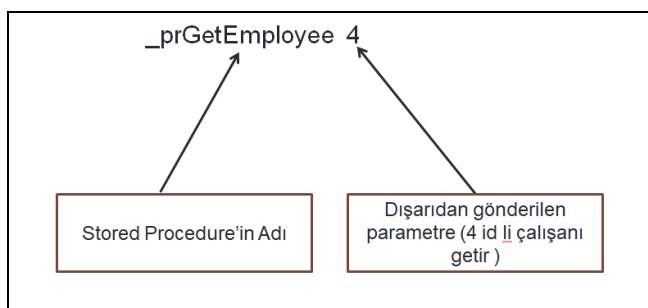
Yukarıda yazdığımız Stored Procedur'a gönderilecek parametreler

Select Stored Procedure Örneği

```
CREATE PROC_prGetEmployee(@employeeId INT)
AS
BEGIN
SELECT * FROM Employees WHERE EmployeeID=@employeeId
END
```

Yukarıdaki Stored Procedur'da Northwind veritabanından Employess tablosundan, Stored Procedur'a parametre olarak gönderilen @employeeId adlı parametre ile bir WHERE sorgusu yapılarak bu id ye sahip çalışan kaydı getirilmiştir.

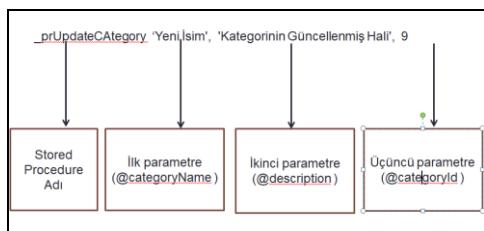
BU Stored Procedure aşağıdaki şekilde çalıştırılacaktır;



Update Stored Procedure Örneği

```
CREATE PROC _prUpdateCCategory
(
    @categoryName  VarChar(50),
    @description   VarChar(MAX),
    @categoryId    INT
)
AS
BEGIN
    UPDATE Categories SET CategoryName = @categoryName, Description = @description
    WHERE CategoryID = @categoryId
END
```

Yukarıdaki Stored Procedure aşağıdaki şekilde çalıştırılmalıdır.



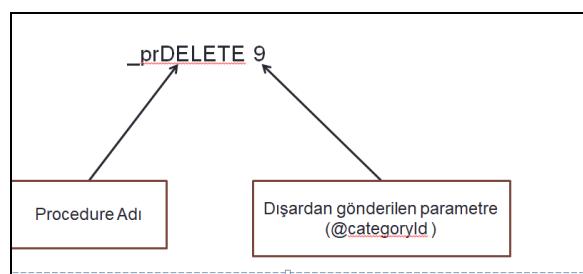
Yukarıdaki Stored Procedur'da Northwind veritabanından Categories tablosunda, Stored Procedur'a parametre olarak gönderilen @categoryName, @description ve @categoryId adlı parametreler ile ilgili kayda bir WHERE sorgusu yapılarak ulaşılmış ve yine parametre olarak gönderilen değişkenler ile ilgili alanlar UPDATE sorgusuyla güncellenmiştir.

Deleted Stored Procedure Örneği

```
CREATE PROC _prDelete(@categoryId INT)
AS
BEGIN
DELETE FROM Categories Where CategoryID =@categoryId
END
```

Yukarıdaki Stored Procedur'da Northwind veritabanından Categories tablosundan, Stored Procedur'a parametre olarak gönderilen @categoryId adlı parametre ile bir WHERE sorgusu yapılarak bu id ye sahip kategori kaydı tablodan silinmiştir

Yukarıdaki Stored Procedure aşağıdaki şekilde çalıştırılmalıdır;



Stored Procedure'da Değişiklik Yapmak

Stored Procedur'larda da diğer SQL nesnelerinde olduğu gibi ALTER anahtar kelimesi ile değişiklik yapılmaktadır. Örneğin daha önce tanımlamış olduğumuz pr_UrunGetir adlı Stored Procedure yapısını Select sorgusunda Products tablosundan sadece belli kolonları getirecek şekilde güncellemek için;

```
ALTER PROC pr_UrunGetir
AS
BEGIN
SELECT ProductID, ProductName FROM Products
END
```

Stored Procedure'u Silmek

Stored Procedur'larda da diğer SQL nesnelerinde olduğu gibi DROP anahtar kelimesi ile silinmektedir. Örneğin daha önce tanımlamış olduğumuz pr_UrunGetir adlı Stored Procedure yapısını silmek için;

```
DROP PROC _pr_UrunGetir
```

Stored Procedure'a gönderilen parametrelerin kontrolü önemli bir konudur. Bir Stored Procedure oluştururken dışarıdan parametre alması istenmişse bu Stored Procedure'a gönderilen parametreler

mutlaka Stored Procedure oluştururken belirtilen tipte olmalıdır. Bunun için Stored Procedure yapısı içinde IF ile kontrol sağlanabilir

```
ALTER PROC _prGetEmployee(@employeeId INT)
AS
IF @employeeId IS NOT NULL AND @employeeId < 50
BEGIN
SELECT EmployeeID, FirstName FROM Employees WHERE EmployeeID = @employeeId
END
ELSE
BEGIN
SELECT 'Böyle bir ürün yok'
END
```

Yukarıda daha önce oluşturduğumuz _prGetEmployee Stored Procedure'u değiştirilmiş ve bir IF kontrolü eklenmiştir. Bu Stored Procedure dışarıdan INT tipinde bir @employeeId parametresi beklemektedir. Eklenen IF kontrolünde bu @employeeId'in hem boş olmaması hem de 5'den küçük olması halinde Stored Procedure'un SELECT sorgusunun çalışması, bu koşul sağlanmadığında ise ELSE adımda 'Böyle Bir Ürün Bulunmadı' sonucunun gösterilmesi sağlanmıştır.

Burada dikkat edilmesi gereken nokta IF şartı sağlandığında çalışacak kodlar bir **BEGIN...END** içine yazılırken şartlar sağlanmadığında çalışacak kodların ise ayrı bir **BEGIN...END** bloğuna yazılmış olduğudur.

Stored Procedure Şifreleme

Stored Procedure'lar da aynı View yapılarında olduğu gibi WITH ENCRYPTION anahtar kelimesi ile şifrelenmektedir. Daha önce oluşturmuş olduğumuz _prGetEmployee adlı Stored Procedure'u şifrelemek için;

```
ALTER PROC _prGetEmployee
WITH ENCRYPTION
AS
IF @employeeId IS NOT NULL AND @employeeId < 50
BEGIN
SELECT EmployeeID, FirstName FROM Employees WHERE EmployeeID = @employeeId
END
ELSE
BEGIN
SELECT 'Böyle bir ürün yok'
```

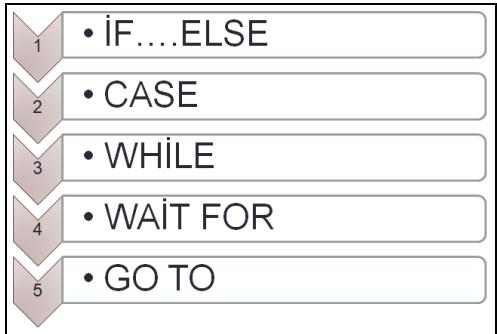
KARAR İFADELERİ VE DÖNGÜLER

Tüm programlama dillerinde bulunan kontrol ifadeleri ve döngüler, genel olarak şu işlemler için kullanılır.

- 1 • Programsal akış şartları sağlanması
- 2 • Belli görevin belli şartlarda gerçekleşmesi
- 3 • Belli görevlerin belirli defa gerçekleşmesi

T-SQL komutlarını bir koşula dayandırarak çalıştırırmak istediğimizde karar ifadelerini ve döngüleri kullanırız. Bu yapılar birçok programlama dilinde de kullanılmaktadır. SQL dilinde de benzer amaçlarla kullanılmakla birlikte söz diziminde (syntax) farklılıklar bulunmaktadır.

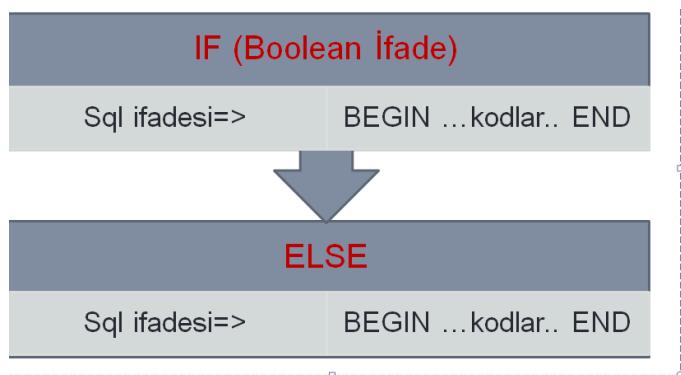
Akiş kontrolleri ve kullanılan yardımcı ifadeler;



IF ELSE YAPISI

En çok kullanılan kontrol ifadelerinden biridir. If-Else yapısı ile veri içerisinde sorguları koşula göre rahatça yapabilmekteyiz. Kisaca, SQL sorgusu belli şartlara uygun ise bu işlemi yap, değilse şu işlemi yap demektir.

IF ELSE Yapısı Söz Dizimi



Basit bir IF örneği

```
DECLARE @sayi INT;
SELECT @sayi = COUNT(ProductID) FROM Products;
IF @sayi IS NOT NULL
PRINT 'Toplam ' + CAST(@sayi AS VARCHAR) + ' kayıt bulundu';
```

Yukarıdaki örnekte Northwind veritabanından Products tablosunda bulunan kayıtlar sorgulanmıştır. @sayi isimli bir değişken oluşturularak bu değişkenin değeri Products tablosundaki ürün sayısına atanmıştır.

```
SELECT @sayi = COUNT(ProductID) FROM Products;
```

Daha sonra IF ifadesi yazılarak @sayi değişkeninin NULL olmaması durumu kontrol edilmiş;

IF @sayi IS NOT NULL

Ve bu şart sağlanırsa çalışması istenen kodlar yazılmıştır.

```
PRINT 'Toplam ' + CAST(@sayi AS VARCHAR) + ' kayıt bulundu';
```

IF yapısında, ELSE kullanılarak akış kontrolünde belirtilen şart ya da şartlar sağlanmadığı takdirde çalışacak sorgular oluşturulabilir. Northwind veritabanından bir IF ELSE örneği;

```
DECLARE @Sayi INT;
SELECT @Sayi = COUNT(*) FROM Orders;
IF @Sayi > 500
BEGIN
PRINT '500 den fazla kayıt var';
END
ELSE
BEGIN
PRINT '500 den az kayıt var';
END;
```

Örneğimizde @sayi isimli bir değişken tanımlanarak değeri Orders tablosundaki kayıt sayısına eşitlenmiştir.

IF @Sayi > 500

denilerek bu sayının 500 den fazla ise ekranda '500 den fazla kayıt var' yazılması istenmiştir.

```
BEGIN
PRINT '500 den fazla kayıt var';
END
```

Bu şartın sağlanmaması yani kayıt sayısının 500'den az olması durumunda çalışması istenen kod ise

ELSE bloğuna yazılmıştır.

```
ELSE
BEGIN
PRINT '500 den az kayıt var';
END;
```

Burada dikkat edilmesi gereken hem IF hem de ELSE bloğu için çalışması istenen SQL kodlarının BEGIN...END arasında yazılmış olduğudur.

IF ELSE yapısında koşul sayısında bir sınırlama yoktur. Kontrol edilecek koşul arttıkça IF ve ELSE arasında ELSE IF eklenerek bütün şartların kontrolü sağlanabilir.

```
IF(Şart1)
BEGIN
    İşlemlerimiz
END
ELSE IF(Şart2)
BEGIN
    İşlemlerimiz
END
ELSE IF(Şart3)
BEGIN
    İşlemlerimiz
END
ELSE
BEGIN
    İşlemlerimiz
END
```

Dikkat edilmesi gereken bir başka durum **IF ... ELSE** yapısında, hiç bir şartın sağlanmadığı durumda çalışması için kullanılan **ELSE** komutunun sorgunun en sonunda kullanılması zorunluluğudur.

```
DECLARE @sayi INT;
SET @sayi = 1;
IF @sayi = 1
PRINT 'Bir'
ELSE IF(@sayi = 2)
PRINT 'İki'
ELSE IF(@sayi = 3)
PRINT 'Üç'
ELSE IF(@sayi >= 4) OR (@sayi <= 8)
BEGIN
PRINT '4 - 8 arasında bir değer';
END
ELSE
PRINT 'Tanımlanamadı.');
```

Yukarıdaki örneğimizde birden fazla şart sorgulanmıştır.

@sayı değişkeninin değeri 1 olarak atandığında, ilk **IF** bloğu çalışacak ve **PRINT** ile ekrana 'Bir' yazacaktır.

@sayı değişkeninin değeri 2 olarak atandığında, ilk **IF** bloğu çalışacak ve **PRINT** ile ekrana 'İki' yazacaktır.

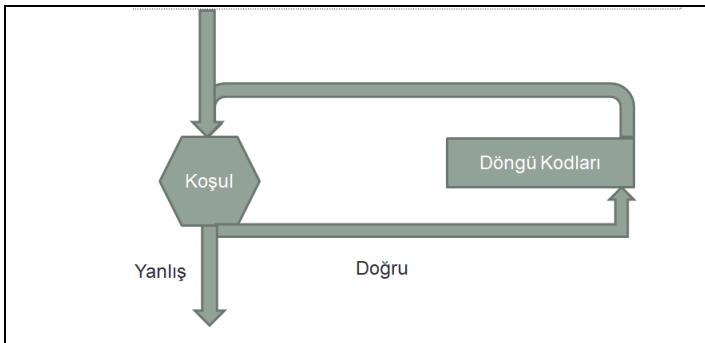
@sayı değişkeninin değeri 3 olarak atandığında, ilk **IF** bloğu çalışacak ve **PRINT** ile ekrana 'Üç' yazacaktır.

@sayı değişkeninin değeri 4 ve 8 dahil, bu sayıların arasında bir değer ise, **PRINT** ile ekrana '4 - 8 arasında bir değer' yazacaktır.

Bu şartların hiç biri sağlanmadığı takdirde, en son olarak belirtilen **ELSE** bloğu çalışır.

WHILE Döngüsü

WHILE ifadesi, tüm programlama dillerinde olduğu gibi, döngüsel olarak bir koşul deyimini test etmek için kullanılır. Sonuç **TRUE** olduğu sürece, döngünün en başına dönerek tekrar test edilir. Sonuç **FALSE** olursa döngüden çıkarılır.



WHILE Örneği

```
DECLARE @sayac INT  
SELECT @sayac = 0  
WHILE @sayac < 5  
BEGIN  
SELECT '@sayac değeri : ' + CAST(@sayac AS VARCHAR(1))  
SELECT @sayac = @sayac + 1  
END;
```

Yukarıdaki örneğimizde `@sayac` isimli bir değişken tanımlayarak bu değişkenin başlangıç değerini 0 olarak belirledik. Daha sonra WHILE döngümüzü oluşturarak şart olarak `@sayac` değişkeninin değerinin 5'ten küçük olmasını belirledik. Bu şart sağlandığında ise ekrana çalışacak kodumuz ise `@sayac` değişkeninin değerinin yazdırılması ve her döngüde `@sayac` değişkeninin değerinin 1 arttırılmasını söylemektedir. `@sayac` değişkeninin değeri her döngüde 1 aratarak koşulumuz olan 5'ten küçük olma durumunu sağlamadığında ise döngü sona ermektedir.

WHILE gibi döngü ifadeleri, genelde bir işlemi tekrarlayarak gerçekleştirmek kullanılır.

WAITFOR

Belirli bir saatte belirlenen işlemi gerçekleştirmek için kullanılır.



WAITFOR ifadesi, parametre olarak belirtilen sürenin dolmasını bekler. Süre dolduğu anda görevini gerçekleştirir.

Waitfor ifadesi delay veya **time** parametreleri ile birlikte kullanılabilir.

waitfor delay; sql sorgusunu belirttiğimiz süre boyunca geç çalışırmak için kullanılır. Yani sorgu çalışma zamanını ertelemek için **waitfor delay** kullanabiliriz.

waitfor time; ise bir saat değerini ifade eder ve ifade edilen saatte sorgunun çalıştırılmasını sağlar.

-- 5 dakika bekletir.

```
WAITFOR DELAY '00:05';
```

-- 5 saniye bekletir.

```
WAITFOR DELAY '00:00:03';
```

-- 5 saat bekletir.

```
WAITFOR DELAY '05:00:00';
```

-- Saat 08:55'de çalışır.

```
WAITFOR TIME '08:55';
```

-- Saat 12:00'de çalışır.

```
WAITFOR TIME '12:00';
```

CASE Kontrolü

CASE yapısı programlama dillerinde kullanılan switch case yapısına benzemektedir. CASE WHEN işlemlerinde sonuçların gösterimi ile alakalı işlemler yapılmaktadır

Söz Dizimi

- SELECT CASE
- WHEN koşul THEN yapılacak işlem
- WHEN koşul THEN yapılacak işlem
- ELSE koşullar sağlanmadığında yapılacak işlem
- END

Northwind veri tabanında Stok kontrollü yapan bir CASE yapısı örneği;

```
Select ProductName AS [ÜRÜN ADI], UnitPrice AS [FİYAT],  
CASE UnitsInStock WHEN 0 THEN 'Stokta Yok'  
ELSE CONVERT(varchar, UnitsInStock) END AS [STOK]  
FROM Products
```

USE ve GO komutu

Go komutu bir T-SQL ifadesi değildir, fakat TSQL kodlarında sıkça kullanılır. Go komutu öncesinde yazılmış olan tüm sql ifadelerini veya bir önceki Go ifadesine kadar olan tüm sql ifadelerini tek bir

execution plan dahilinde SQL Server'a gönderir. Go komutu TSQL ifadesi olmadığı için yeni bir satırda tek başına yazılır.

The screenshot shows the Microsoft SQL Server Management Studio interface. A query window titled 'SQLQuery1.sql - W...ORTHWND (sa (52))' contains the following T-SQL code:

```
USE NORTHWND
GO
SELECT * FROM Products
```

The results pane displays the output of the query:

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit
1	1	Chai	1	1	10 boxes x 20 bag
2	2	Chang	1	1	24 - 12 oz bottles
3	3	Aniseed Syrup	1	2	12 - 550 ml bottles
4	4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars
5	5	Chef Anton's Gumbo Mix	2	2	36 boxes
6	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars
-	-	MOLTON BRIDGE BISCUITS	-	-	40 - 14 - oz jars

Yukarıdaki örnekte ilk olarak kullanılan;

USE NORTHWND

ifadesi sorgu esnasında kullanılacak veritabanını belirtir.

Buradan sonra gelen

GO

ifadesi ise sorgunun devam ettiğini ve üstteki ifadenin altta yer alan

SELECT * FROM Products

ifadesi ile birleştirilerek birlikte çalıştırılması gerektiğini belirtmek için kullanılmıştır.

Go ifadesinin temel kullanımı bu amaçla yapılır. Ancak script tabanlı uzun ifadeler yazıldığındá kullanılır. Örneğin;

```
INSERT deneme
SELECT 'EREN', 'EFE'
GO 4
```

Yukarıdaki kod deneme isimli tablomuza 4 kere Insert işlemi yapacaktır.

PRINT komutu

Değişkenlerin değerlerini, dönen hataları, diğer ihtiyaç duyulan herhangi bir veriyi yazdırınmak için kullanılır. Genelde DEBUG işlemlerinde kullanılmaktadır.

Örneğin;

```
Print 'EREN EFE'
```

komutunu çalıştırduğumızda, bu sorgunun sonucunun messages alanında gösterildiğini görebiliyoruz;

The screenshot shows the SSMS interface with a query window titled 'SQLQuery1.sql - W...O.TESTDB (sa (52))'. The query 'Print 'EREN EFE'' is entered. Below the results grid, the 'Messages' tab is selected, displaying the output 'EREN EFE'.

SSMS içinde Print ifadesinin yanındaki herhangi bir değer messages alanının içinde gösterilir. Normalde biz sorgu sonuçlarını Results alanında görüyoruz. Örneğin Northwind veritabanında aşağıdaki sorgumuzu çalıştırduğumızda;

```
SELECT * FROM Products
```

The screenshot shows the SSMS interface with a query window titled 'SQLQuery1.sql - W...ORTHWND (sa (52))'. The query 'SELECT * FROM Products' is entered. The results are displayed in a grid under the 'Results' tab, showing product details for 77 rows. The 'Messages' tab is also visible below the results.

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18.00	39	0	10	0
2	Chang	1	1	24 - 12 oz bottles	19.00	17	40	25	0
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.00	13	70	25	0
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.00	53	0	0	0
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	0	0	0	1
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.00	120	0	25	0
7	Uncle Bob's Organic Dried Pears	4	2	42 - 14 oz jars	30.00	15	10	15	0

Şeklinde görürüz.

```
SELECT * FROM Products
```

Ifadesinin bize 2 şekilde sonuç döndüğünü görüyoruz. Birincisi sorgu yapılan tablodaki kayıtların listesi bu Results alanında görünen sonuçlardır. İkincisi ise Messages alanında bu sorgu ile ilgili Sql serverin bize gönderdiği mesaj gelmektedir. Bu sorgu için messages bölümünde 77 kayıtın bu sorgudan etkilendiğini söylemektedir;

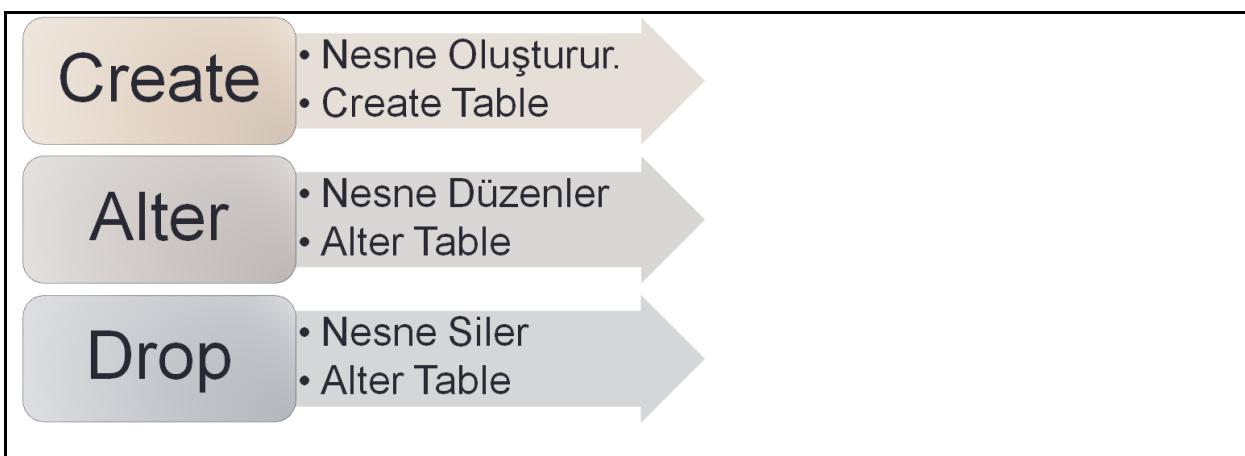
The screenshot shows a SQL query window titled "SQLQuery1.sql - W...ORTHWND (sa (52))". The query is:

```
Print 'EREN EFE'  
SELECT * FROM Products
```

The results pane shows the message "(77 rows affected)".

DATA DEFINITION LANGUAGE (VERİ TANIMLAMA DİLİ)

Veri Tanımlama Dili (**DDL**) yani **Data Definition Language**, nesnelerin tanımlanması için kullanılan komut kümesidir. Create, Alter ve Drop komutlarını içerir.

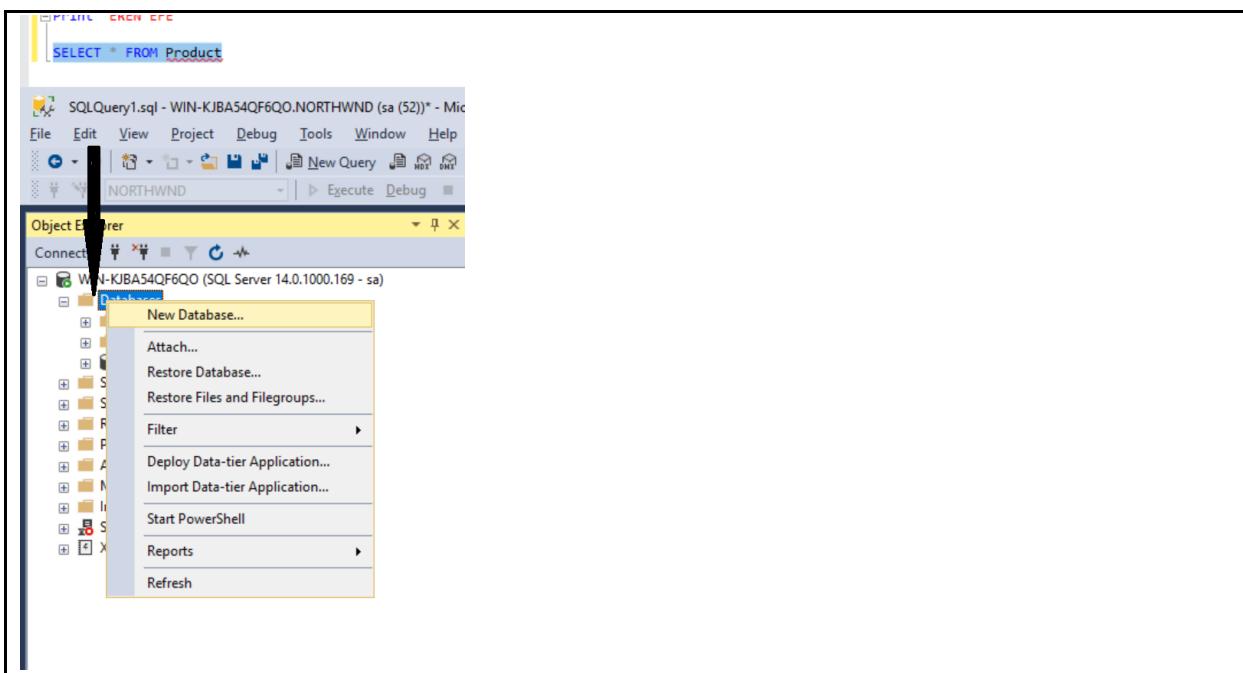


CREATE DATABASE

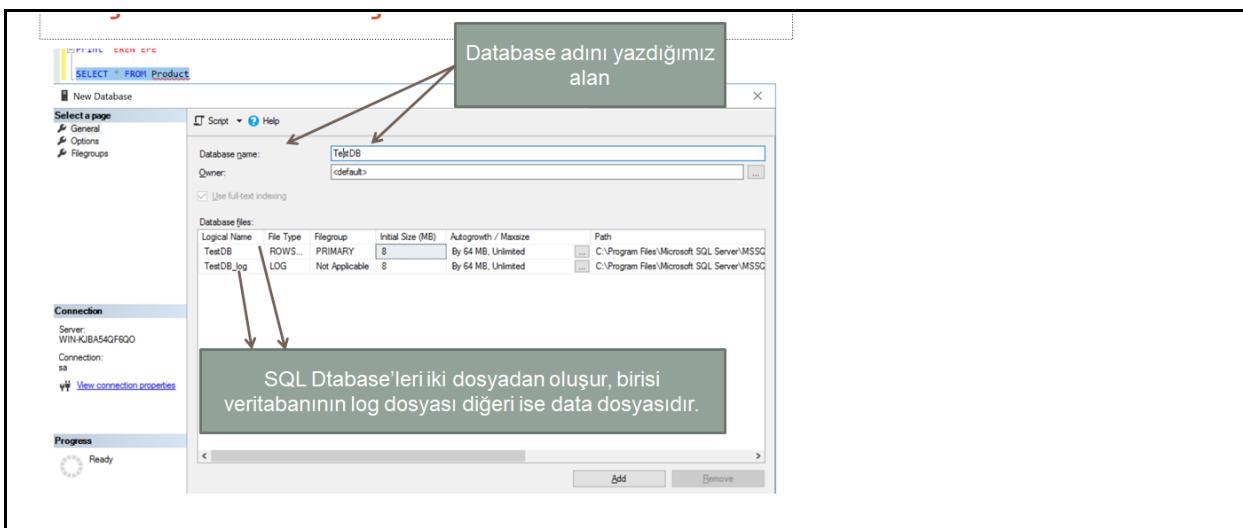
Sql Server üzerinde veritabanı oluşturmak için kullanılan komuttur. Biz şu ana kadar Sql Server Management Studio'nun arayüzüni kullanarak veritabanlarımızı oluşturduk. Bunun için Databases menüsüne sağ tıklayarak New Database dememiz yeterlidir;

Database oluşturmak için bir başka yöntem ise, Sql Server Management Studio'nun da arka planda çalıştığı Sql kodlarını kullanarak veritabanı oluşturmaktır. Bunları sırasıyla yapalım. Sql Server Management Studio'da Databases menüsüne sağ tık New Database diyerek yeni bir veritabanı oluşturalım;

Database oluşturmak için bir başka yöntem ise, Sql Server Management Studio'nun da arka planda çalıştığı Sql kodlarını kullanarak veritabanı oluşturmaktır. Bunları sırasıyla yapalım. Sql Server Management Studio'da Databases menüsüne sağ tık New Database diyerek yeni bir veritabanı oluşturalım;

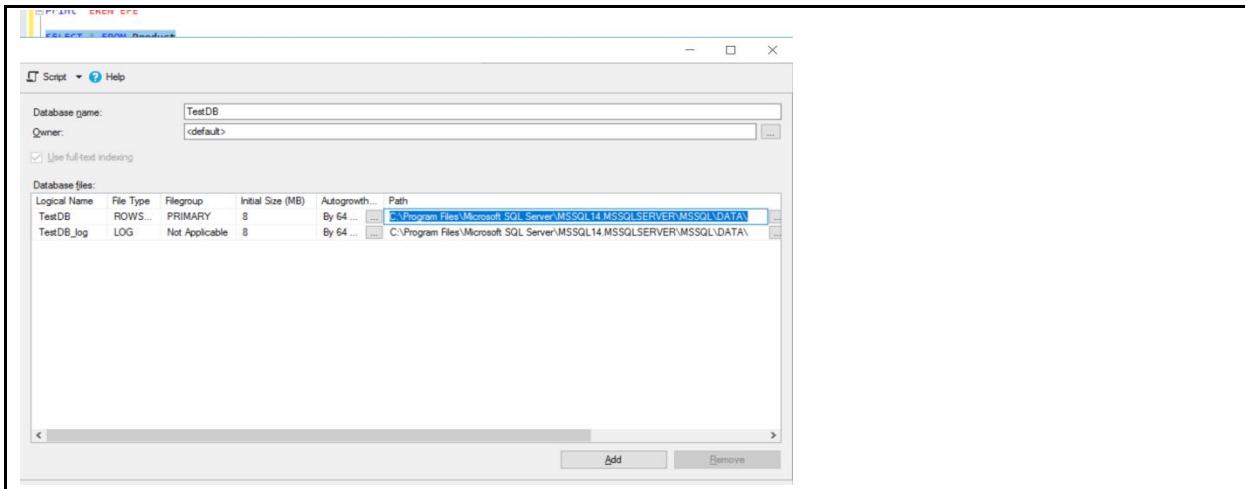


Açılan pencere;

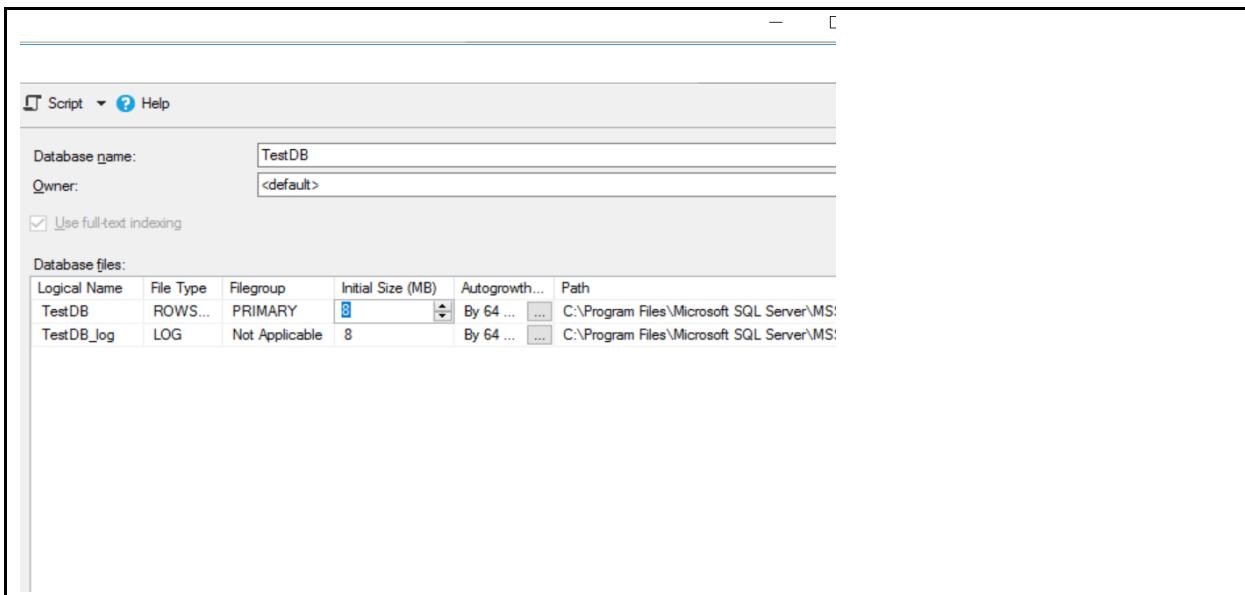


Bu alanda öncelikle veritabanı adını 'Database name' yazan alana gireriz. Bunun altında 'Database files' bölümünde ise bir sql veritabanını oluşturan iki dosya görülmektedir. Bunlardan birisi log dosyası diğeri ise data dosyasıdır. Veri dosyası uzantısı mdf yani master data file, Log dosyası uzantısı ldf yani log data file olarak görülmektedir.

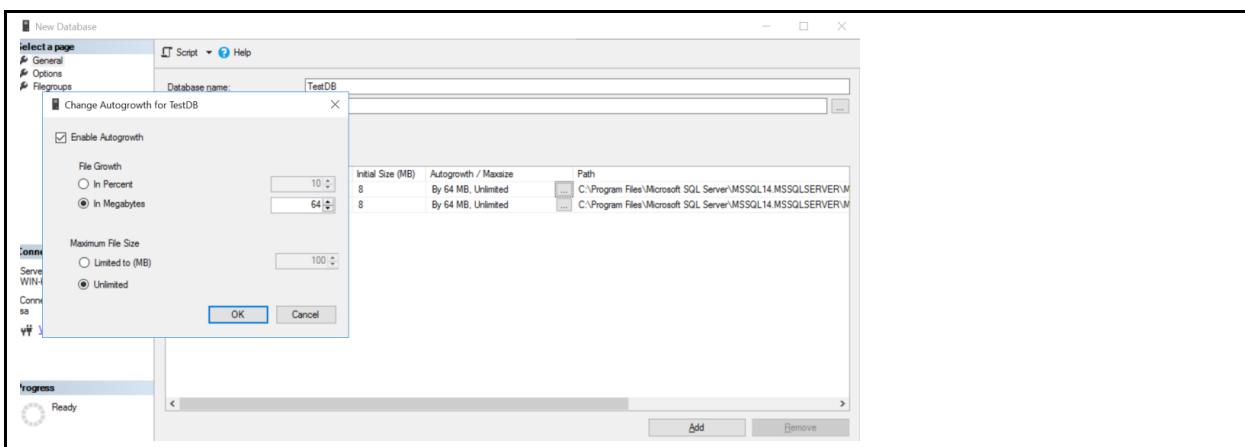
Path alanında bu dosyaların default olarak kayıt oldukları dosya görülmektedir. Eğer değişiklik yapmazsa veritabanı dosyaları Sql server altındaki data klasörüne kaydolur;



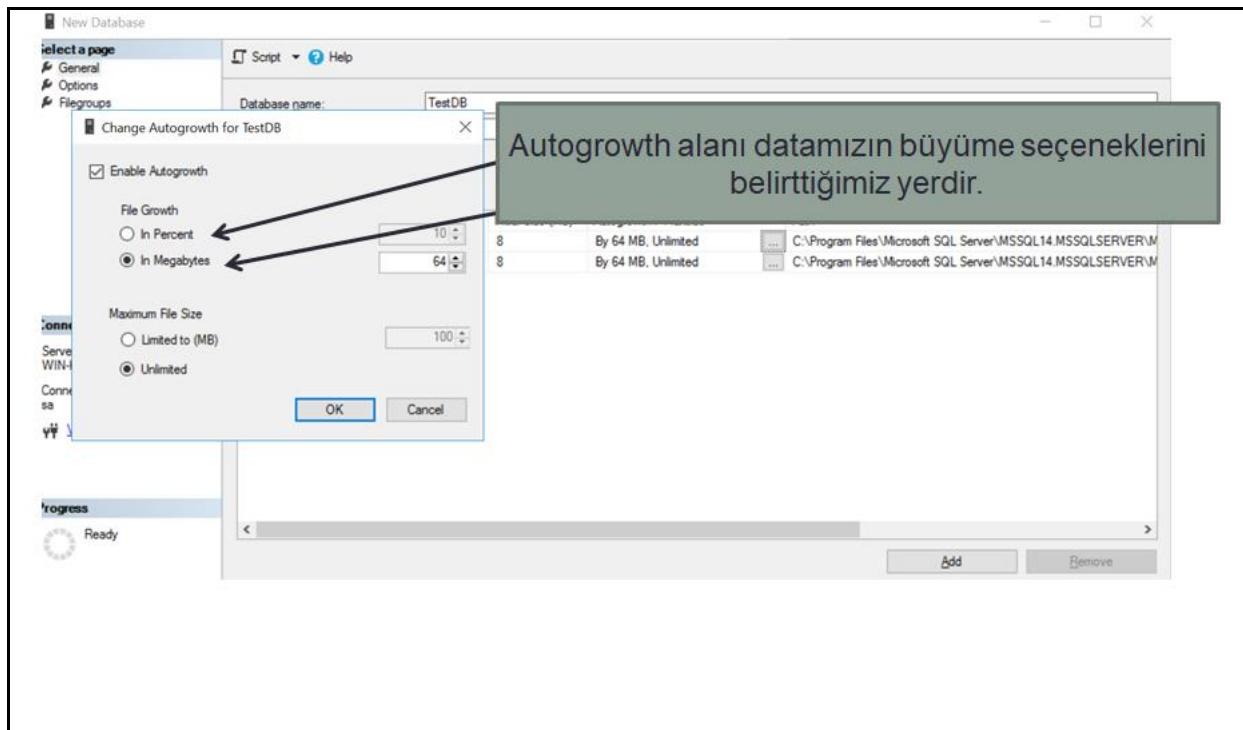
Initial Size alanında ise dosyalarımızın ilk defa oluşturulurken boyutları gösterilmektedir. Burada 8mb görünürmekte;



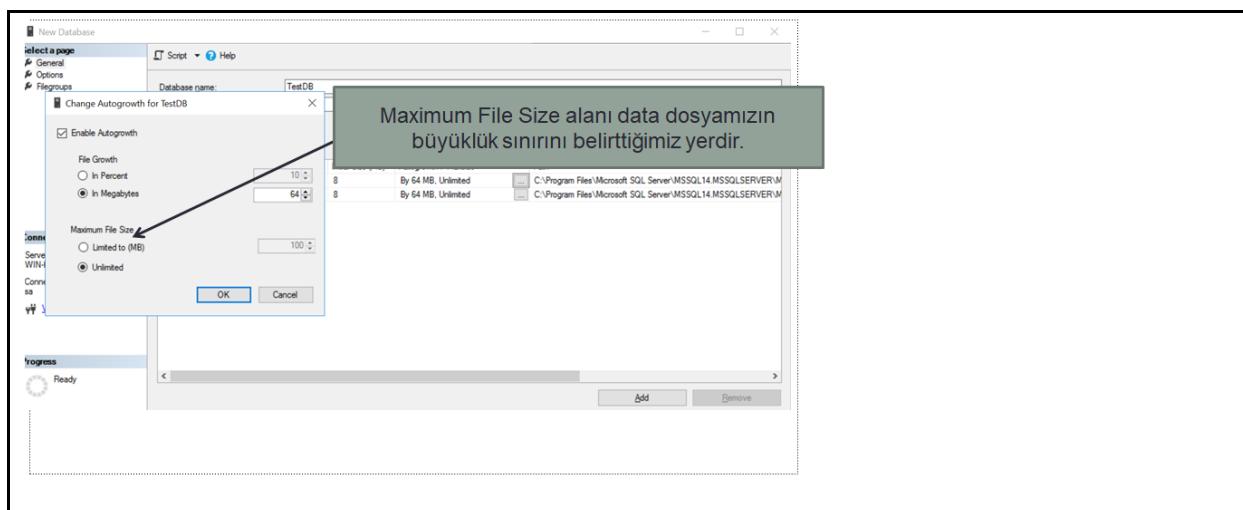
Autogrowth ve Maxsize bölümünde ise, dosyalarımızın büyümeye seçeneklerini belirleriz.



Burası data büyürken nasıl büyüsün sorusuna cevap verdigimiz alandır. In Megabytes alanı datamızın kaç Mb'lık veriler halinde büyüyeceğini belirtir. Eğer datamız hızlıca büyüyecekse 1'er Mb şeklinde büyümesi sistemin kasılması neden olur. Örneğin tek seferde 50Mb'lık bir veri eklediğimizde datanın 1'er Mb şeklinde büyümesi sistemi yavaşlatır. Bunun yerine Percent alanı seçildiğinde burada belirtilen yüzde oranı kadar data büyümesi seçilmiş olur.

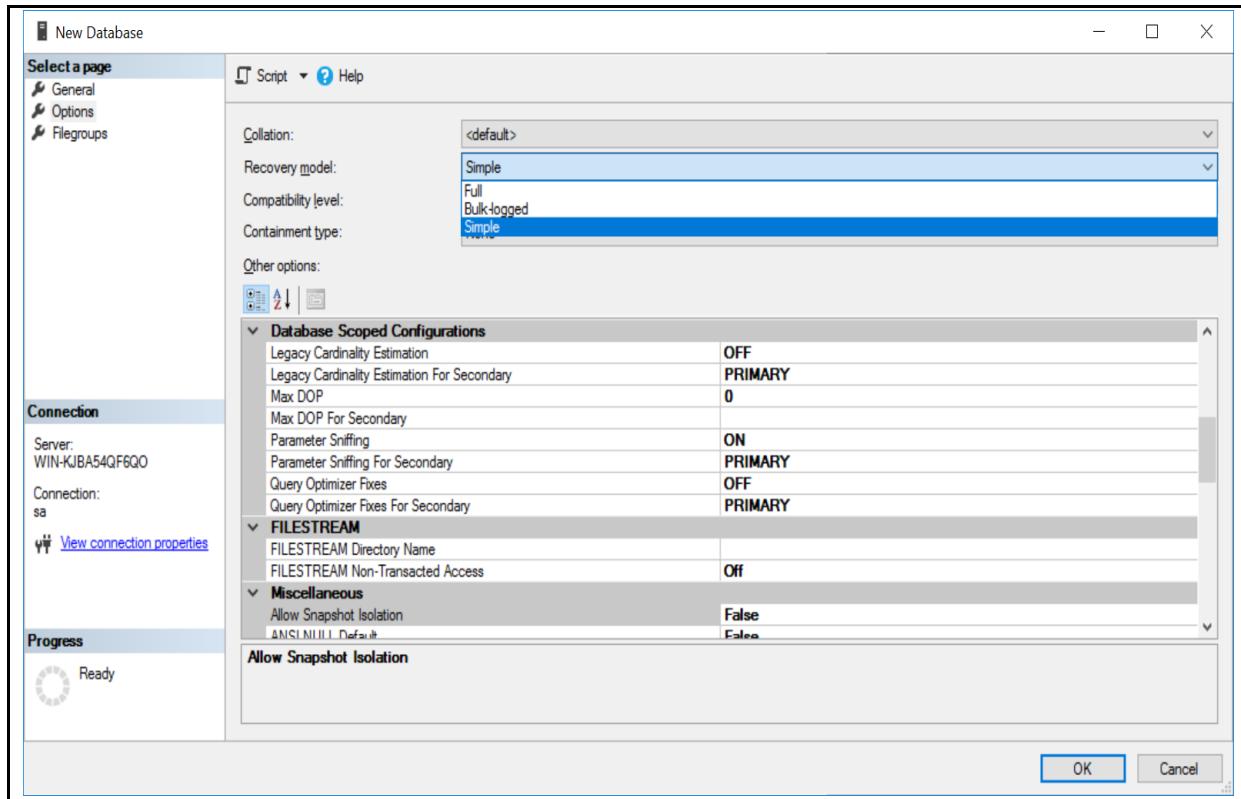


Buradaki Maximum File Size alanı ise datamızın büyüklüğünü sınırlayabileceğimiz alandır. Örneğin buraya 100Mb dersek veritabanımızdaki data büyülüğu 100Mb olduktan sonra herhangi bir kayıt ekleyemeyiz. Burada unlimited seçeneği ise herhangi bir dosya boyutu sınırı olmadan veritabanımıza kayıt ekleyebilmemizi sağlar.



Burada Options kısmında ise; Recovery model alanını görmekteyiz. Buradaki seçeneklerimiz Full, Simple ve Bulk-logged olarak karşımıza çıkar. Burası Full ise bizim Sql serverda yaptığımız her işlem

log dosyasına aynı şekilde yazılır ve biz silene kadar log dosyasında kalır. Elbette log dosyasının bu durumda çok fazla büyümesi söz konudur. Bulk-logged'de sadece toplu işlemler log dosyasına yazılır. Simple'da ise yapılan işlemler log dosyasına yazılıp daha sonra data dosyasına kaydedilir ve log dosyasından silinir.



SQL üzerinde database oluşturmanın bir diğer yolu da TSQL kodları ile yapılmalıdır. Sql kodları ile bir veritabanı oluştururken çok fazla parametre kullanılmamakta ve bir veritabanının tüm ayarları bu parametreler ile yapılmaktadır. Ancak En temel kullanımı bir data ve bir log dosyası üzere iki dosyanın oluşturulması şeklindeki kullanımdır.

Sql kodlarıyla bir database oluşturmak için

CREATE DATABASE

İfadesi kullanılarak başlanır ve daha sonra Database adı yazılır. Az önce TestDB adında oluşturduğumuz için bu sefer DenemeDB adında bir Database oluşturalım

```
CREATE DATABASE DenemeDB
```

daha sonra ON PRIMARY ifadesini kullanarak bir parantez açıyoruz. Tahmin edileceği üzere bu alan PRIMARY olan yani mdf olan dosyamızın bilgilerini gireceğiz.

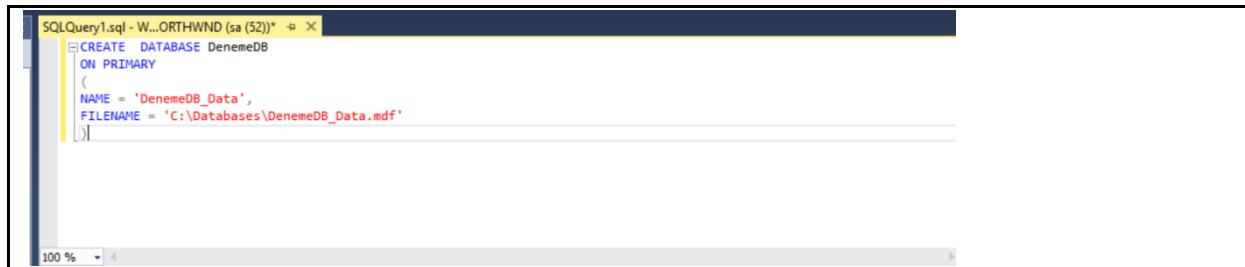
```
CREATE DATABASE DenemeDB  
ON PRIMARY  
( )
```

Daha sonra açtığımız parantez içine;

İlk olarak Name parametresine data dosyamızın adını yazıyoruz. Database adı ve araya – (tire) işaretini koyarak dosya adını verebiliriz.

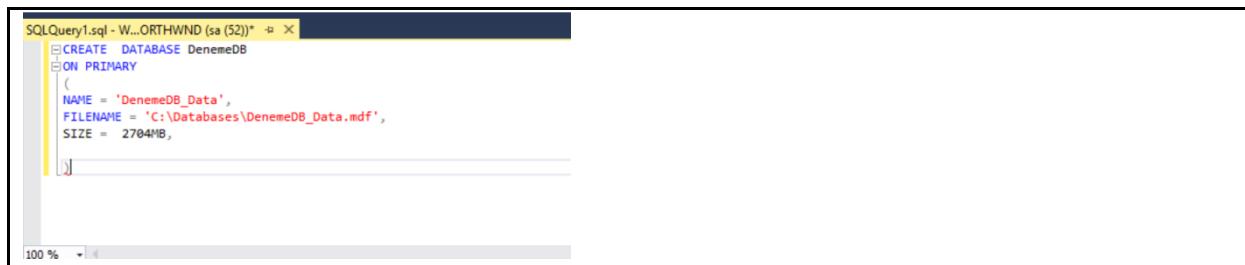
```
CREATE DATABASE DenemeDB  
ON PRIMARY  
(  
NAME = 'DenemeDB_Data'  
)
```

Daha sonra dosyamızın FILENAME'ini girmeliyiz. Bunu dosyamızın fiziksel olarak isimlendirilmesi şeklinde düşünebiliriz. Aslında dosyamızın yolunu gösterdiğimiz alandır.



```
SQLQuery1.sql - W...ORTHWND (sa (52)) *  
CREATE DATABASE DenemeDB  
ON PRIMARY  
(  
NAME = 'DenemeDB_Data',  
FILENAME = 'C:\Databases\DenemeDB_Data.mdf'  
)
```

Daha sonra SIZE alanında data dosyamızın boyutunu belirtiyoruz. Bunu kb, mb ve ya gb olarak belirtebiliriz.



```
SQLQuery1.sql - W...ORTHWND (sa (52)) *  
CREATE DATABASE DenemeDB  
ON PRIMARY  
(  
NAME = 'DenemeDB_Data',  
FILENAME = 'C:\Databases\DenemeDB_Data.mdf',  
SIZE = 2704MB,  
)
```

Daha sonra MAXSIZE alanına ise data dosyamızın büyülüklük sınırını belirliyoruz. Bunu az önce gördüğümüz gibi Unlimited olarak belirleyebiliriz.



```
SQLQuery1.sql - W...ORTHWND (sa (52)) *  
CREATE DATABASE DenemeDB  
ON PRIMARY  
(  
NAME = 'DenemeDB_Data',  
FILENAME = 'C:\Databases\DenemeDB_Data.mdf',  
SIZE = 2704MB,  
MAXSIZE = UNLIMITED  
)
```

Daha sonra FILEGROWTH alanda ise data dosyamızın ne kadarlık şekilde arttırılacağını belirttiğimiz alandır.

```
SQLQuery1.sql - W...ORTWND (sa (52))* X
CREATE DATABASE DenemeDB
ON PRIMARY
(
    NAME = 'DenemeDB_Data',
    FILENAME = 'C:\Databases\DenemeDB_Data.mdf',
    SIZE = 2704MB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 16MB
)
```

Data dosyamızın parametrelerini girdik. Şimdi ise

LOG ON () diyerek log dosyamızın bilgilerini girmeliyiz. Yukarıdaki NAME, FILENAME, SIZE ve FILEGROWTH alanlarını giriyoruz.

Elbette log dosyamızın adını ve uzantısını değiştirmeliyiz.

```
SQLQuery1.sql - W...ORTWND (sa (52))* X
CREATE DATABASE DenemeDB
ON PRIMARY
(
    NAME = 'DenemeDB_Data',
    FILENAME = 'C:\Databases\DenemeDB_Data.mdf',
    SIZE = 2704MB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 16MB
)

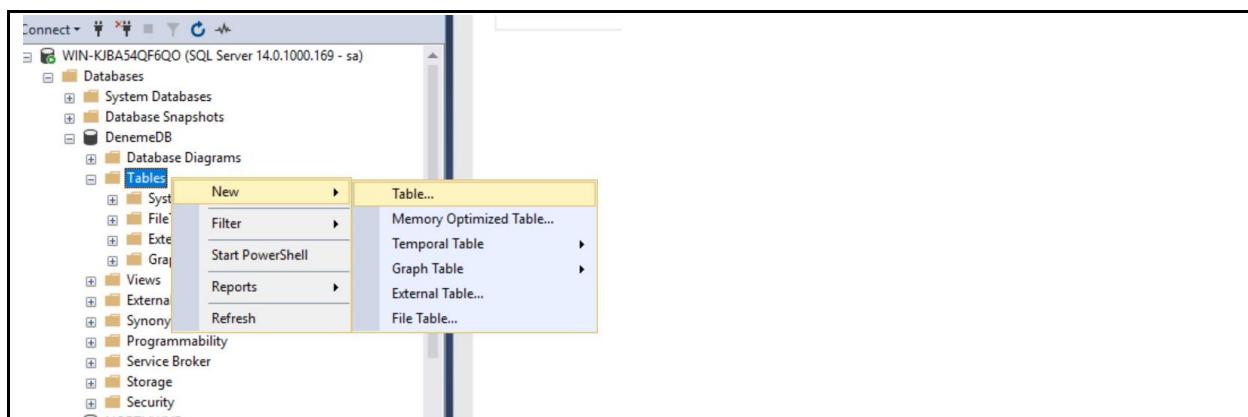
LOG ON
(
    NAME = 'DenemeDB_Log',
    FILENAME = 'C:\Databases\DenemeDB_Log.ldf',
    SIZE = 2GB,
    MAXSIZE = UNLIMITED,
    FILEGROWTH = 16MB
)
```

Burada Log dosyamızın SIZE alanını dikkat ettiğiniz gibi 2GB gibi büyük boyutta belirledik. Çünkü log dosyaları data dosyalarına log dosyaları daha çok kullanılırlar ve bütün işlemler bu dosyaya eklendiği için bu dosya sürekli büyüyecektir.

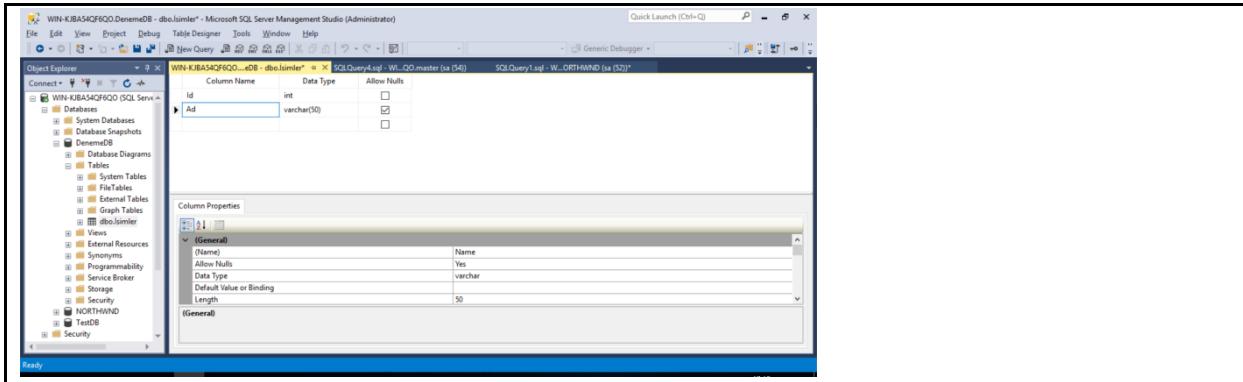
Ve en altta GO ifadesini kullanarak database'imizin oluşmasını sağlıyoruz.

CREATE TABLE

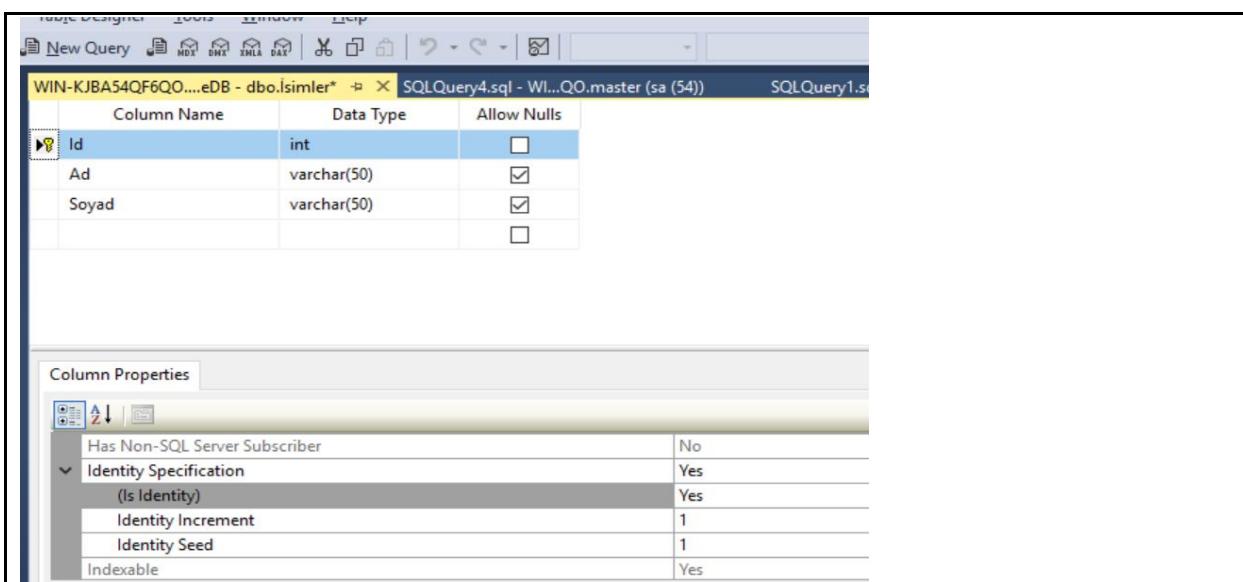
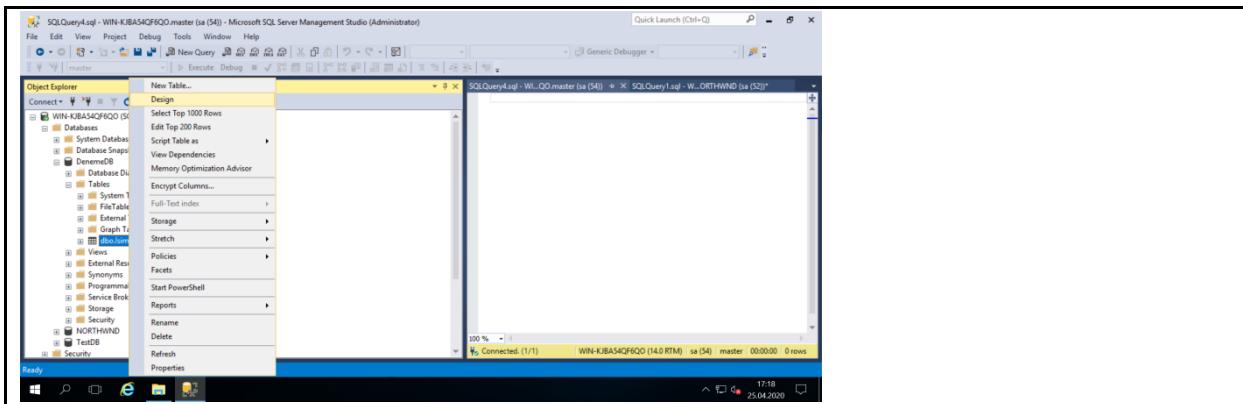
Sql Server Management Studio arayüzü kullanarak tablo oluşturmak için, tabloyu ekleyeceğimiz Database'e giderek burada Tables bölümüne sağ tık ve New Table diyebiliriz.



Açılan ekranda tablo kollarının isimlerini ve data tiplerini belirleriz. Burada ayrıca eklediğimiz kolonların NULL olup olamayacaklarını da belirleriz. DenemeDB veritabanına İsimler adında bir tablo kaydedelim;

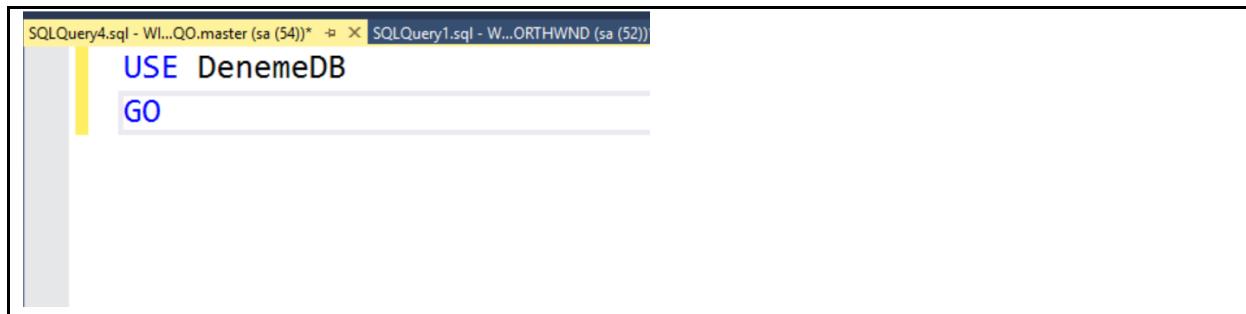


Bu tablo üzerinde değişiklik yapmak için tablo adına sağ tık ve Design dememiz gereklidir. Örneğin tablomuza soyad şeklinde bir kolon daha ekleyelim ve Id alanımızı PrimaryKey olarak işaretleyelim;



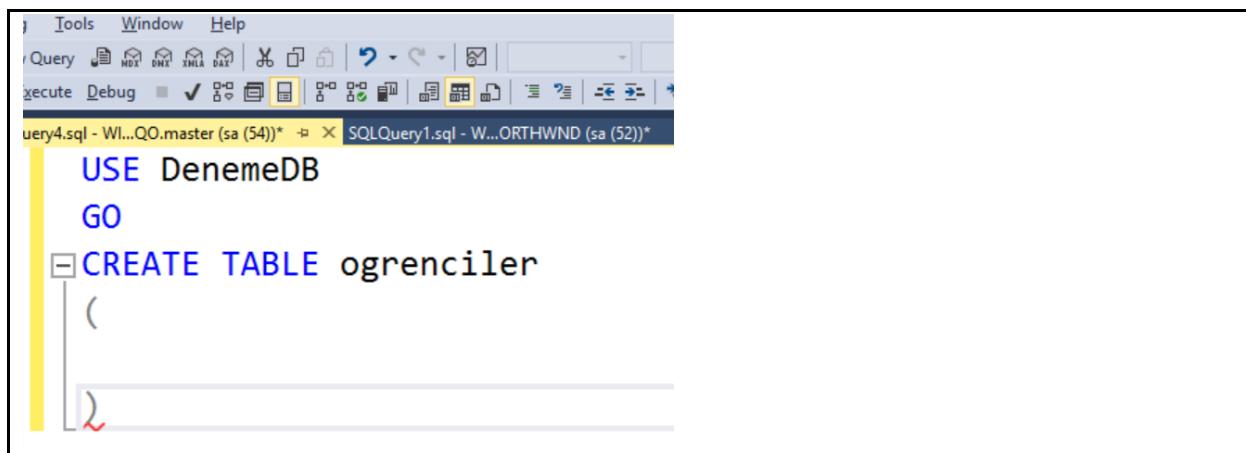
Tablomuzda istediğimiz değişiklikleri yaptıktan sonra kaydederek çıkabiliriz.

Şimdi de Sql kodları kullanarak bir tablo kaydetme işlemi gerçekleştirelim. Öncelikle USE ifadesiyle birlikte hangi Database'e bir tablo ekleyeceğimizi belirtmeliyiz. Ve daha sonra Go komutuyla komutumuza devam ediyoruz.



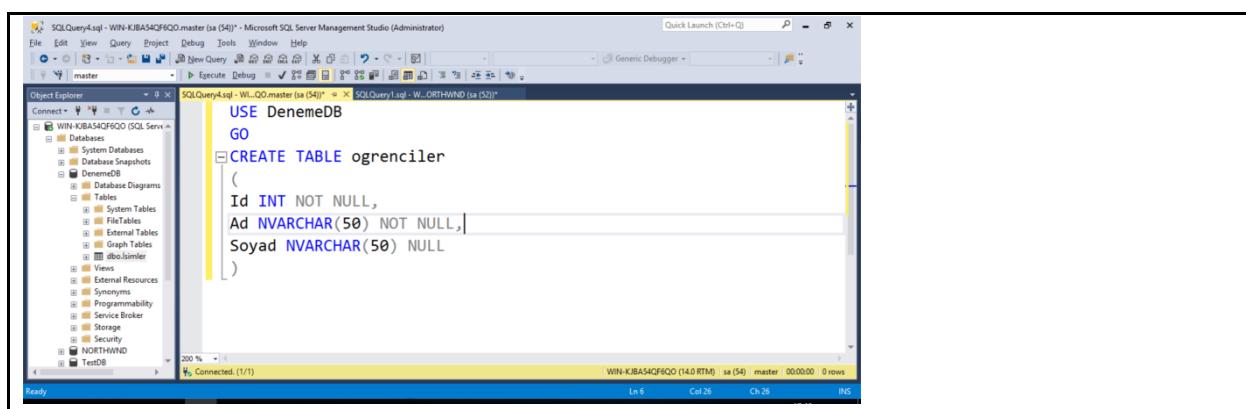
```
USE DenemeDB
GO
```

Daha sonra CREATE TABLE diyerek tablomuzun ismini yazarak parantez açıp kapatıyoruz. Bu parantez içine tablomuzdaki kolonları yazacağız. Örnekte öğrenciler simli bir tablo oluşturulmaktadır.



```
USE DenemeDB
GO
CREATE TABLE ogrenciler
```

Daha sonra kolon isimlerini ve bu kolonlara gelecek verilerin tiplerini yazmalıyız. öğrenciler tablosunda INT tipinde bir Id, NVARCHAR(50) tipinde bir Ad ve NVARCHAR(50) tipinde bir Soyad alanı olacak. Ayrıca burada bu kolonların hangilerinin NULL olup olamayacağını da belirleriz. Bu tabloda Id ve Ad alanları NULL olamaz Soyad alanı ise NULL olabilir şekilde yapılacak.



```
USE DenemeDB
GO
CREATE TABLE ogrenciler
(
    Id INT NOT NULL,
    Ad NVARCHAR(50) NOT NULL,
    Soyad NVARCHAR(50) NULL
)
```

Kodumuzu çalıştırduğımızda tablomuz oluştu.

ALTER DATABASE

Veri tabanında değişiklik yapmak için ALTER DATABASE kodu kullanılır. Daha sonra ise üzerinde değişiklik yapmak istediğimiz database adını yazmamız gerekmektedir

```
ALTER DATABASE DenemeDB
MODIFY FILE
(
    NAME= 'DenemeDB_Data',
    SIZE = 3GB
)
```

Commands completed successfully.

Kodumuzu çalıştırduğımızda DenemeDB veritabanımız güncellenecektir.

ALTER TABLE

Tablo üzerinde değişiklik yapmak için ALTER TABLE kodu kullanılır. ALTER TABLE ifadesinden sonra değişiklik yapmak istediğimiz tablonun adı yazılır. Az önce oluşturduğumuz. Ogrenciler isimli tabloma Kayıt tarihi isimli yeni bir kolon ekleyelim. Yeni bir kolon ekleneceği için ADD ifadesini kullanmalıyız;

```
ALTER TABLE ogrenciler
ADD
    Kayıt_Tarihi DATETIME NOT NULL
```

Commands completed successfully.

Kodumuzu çalıştırıp SELECT * FROM ogrenciler şeklinde tablomuzu çağrıduğımızda kolonun tabloma eklendiğini görebiliriz.

```
ALTER TABLE ogrenciler
ADD
    Kayıt_Tarihi DATETIME NOT NULL

Select * From ogrenciler
```

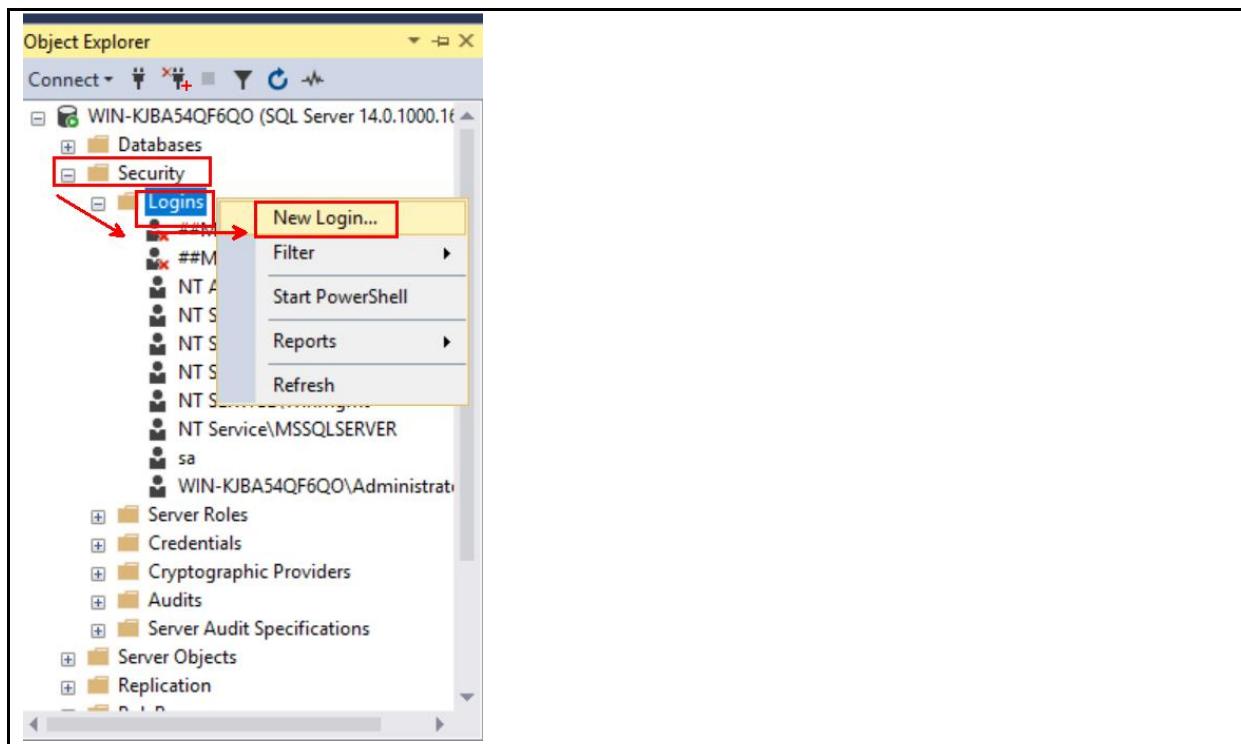
Id	Ad	Soyad	Kayıt_Tarihi

SQL KULLANICI OLUŞTURMA VE YETKİLENDİRME

Sql Server'daki kullanıcıların hesaplarına Logins adı verilmektedir. Her bir Login'e ayrı yetki verilebilir ve erişebileceği veritabanları belirlenebilir.

Sql Server'da güvenlik seviyesi iki katmandan oluşur. Bunlar server ve veritabanı seviyeleridir. Login hesabı öncelikle server seviyesinde yani instance seviyesinde oluşturulur.

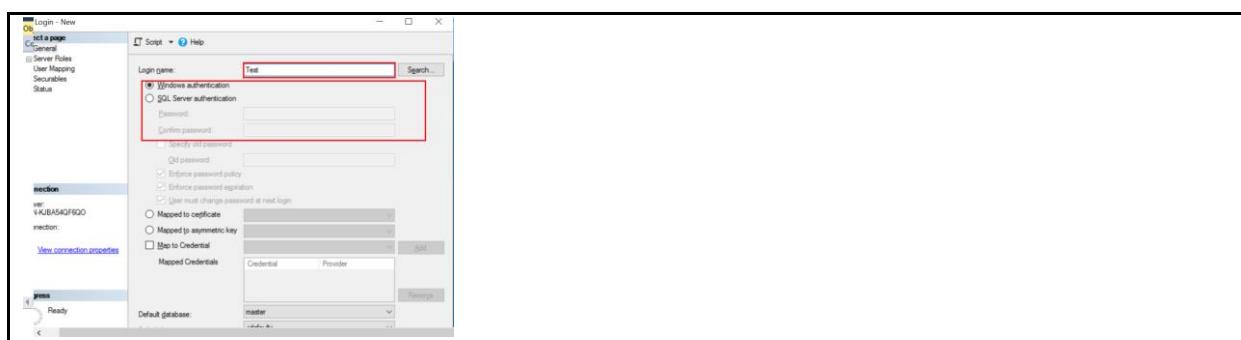
Sql server üzerinde yeni bir Login oluşturmak için Security klasörü altındaki Logins klasörüne sağ tık ve New Login dememiz gerekmektedir.



Açılan ekranda öncelikle Login name alanına Login adını giriyoruz. Bu alanın altında ise Sql Server Management Studio'ya nasıl bağlanacağımızı seçmemiz gerekiyor. Daha önce görmüş olduğumuz iki tip bağlantı seçeneği bulunmakta.

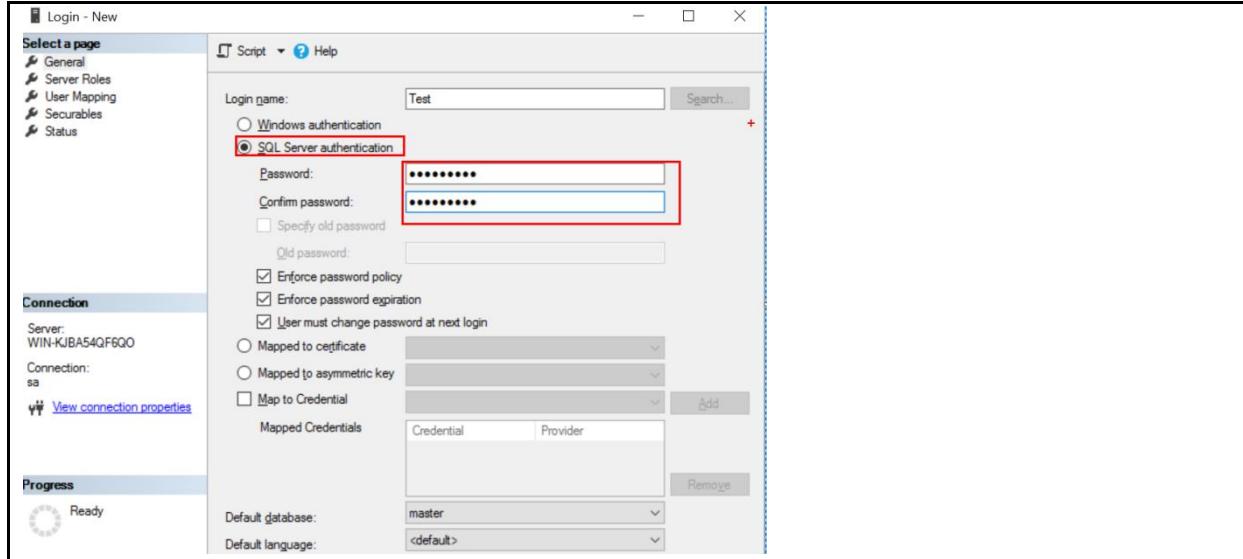
Sql Server Authentication

Windows Authentication



Windows Authentication, sadece Windows hesaplarından açılan oturumları kabul eder. Kullanıcı Windows hesabı ile bağlandığında kullanıcı adı ve parolaya gerek duymadan Sql Server'a login olur.

Sql Server Authentication, ise kullanıcının Sql Server'a login olabilmesi için kullanıcı adı ve parolaya istemektedir. Biz Server Authentication seçelim ve parolamızı girerek devam edelim.



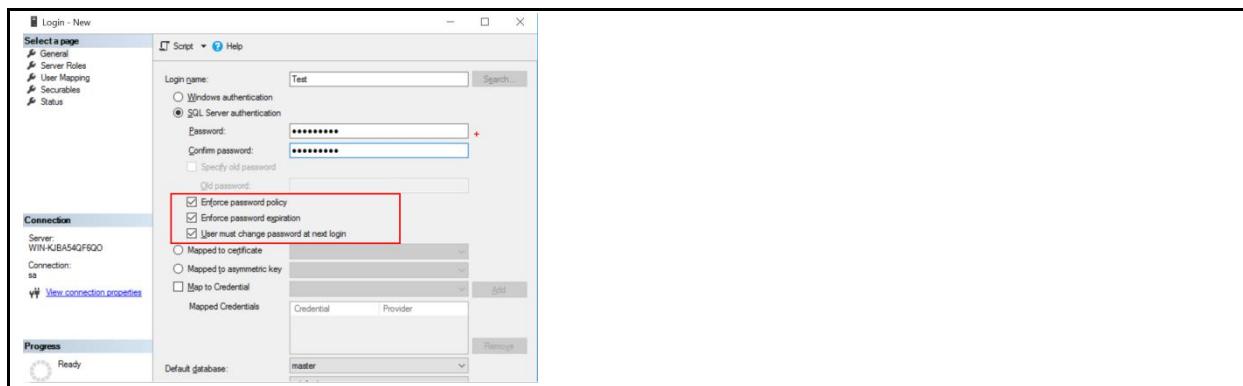
Bu bölümün hemen altında password'a ilişkin seçenekler bulunmaktadır;

Enforce Password Policy oluşturacağımız şifrenin Sql Server'da tanımlı Policy yani şifre kurallarına uygun olup olmadığını kontrol eder.

Enforce Password Expiration seçeneği zamanı geldiğinde şifre değiştirme gerekliliği oluşturur ve şifresi değiştirilmeyen Login'lerin Disable(pasif) duruma çekilmesini sağlar.

User Must Change Password At Next Login seçeneği ise oluşturan kullanıcının ilk giriş yaptığı anda şifresini değiştirmesini zorunlu tutar.

İki seçenekleri seçili olarak bırakıp devam edelim



Sonraki seçeneklerde ise Login için sql server sertifika oluşturma seçenekleri sunar. Default Database kısmında bu kullanıcı sql servera giriş yaptığıda başlangıçta hangi veritabanında karşılaşacağını belirler. Default Language kısmında da başlangıç dili belirlenir. BU kısımları değiştiriyoruz. Sol taraftan Server Roles kısmını seçerek devam edelim.



Server Roles kısmında oluşturduğumuz yeni kullanıcının Sql Server üzerinde hangi yetkilere sahip olabileceğini belirtiyoruz. Eğer hiç değiştirmeden public bırakarak devam edersek bu kullanıcı için herhangi bir yetki vermemiş oluyoruz. public bırakarak devam edelim.

Buradaki roller çeşitli işlevlere sahiptir.

bulkadmin: Toplu Insert işlemlerini yerine getirir.

dbcreator: Veritabanı oluşturma yetkisine sahiptir.

diskadmin: Disk yönetici, veritabanı dosyalarını yönetir.

processadmin: İşlem yönetici Sql Server'daki process'leri yönetir.

public: Sql Server'da standart giriş yapan herkesin atandığı roldür. Bu kural ile tim kullanıcıların kısıtlı hakları vardır.

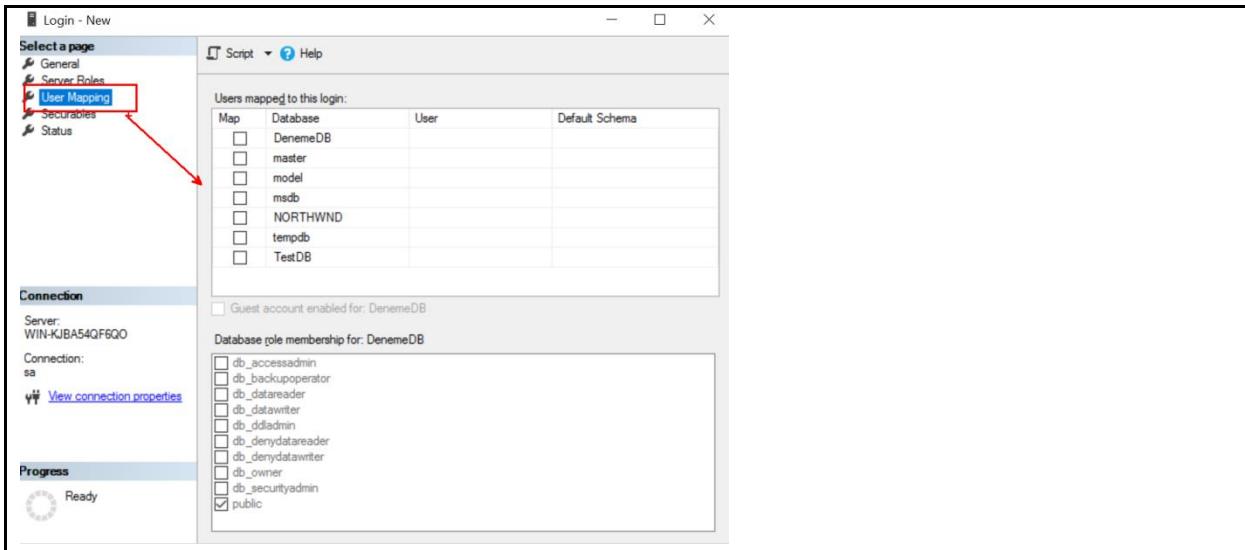
securityadmin: Güvenlikten sorumludur.

serveradmin: Sunucu yönetici

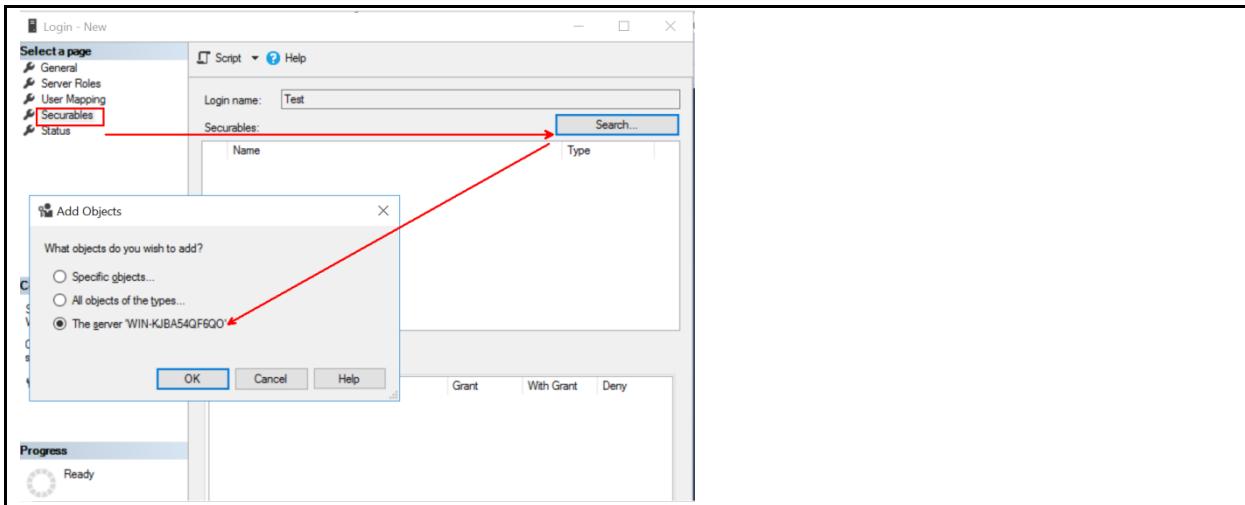
setupadmin: Kurulum yönetici

sysadmin: Sistem admini. Server'daki hemen hemen her işi yapabilir.

Sol taraftan User Mapping kısmı ise bu Login'in hangi tablolar üzerinde yetkili olacağı belirlendiği alandır. Burada da herhangi bir değişiklik yapmadan devam edelim.



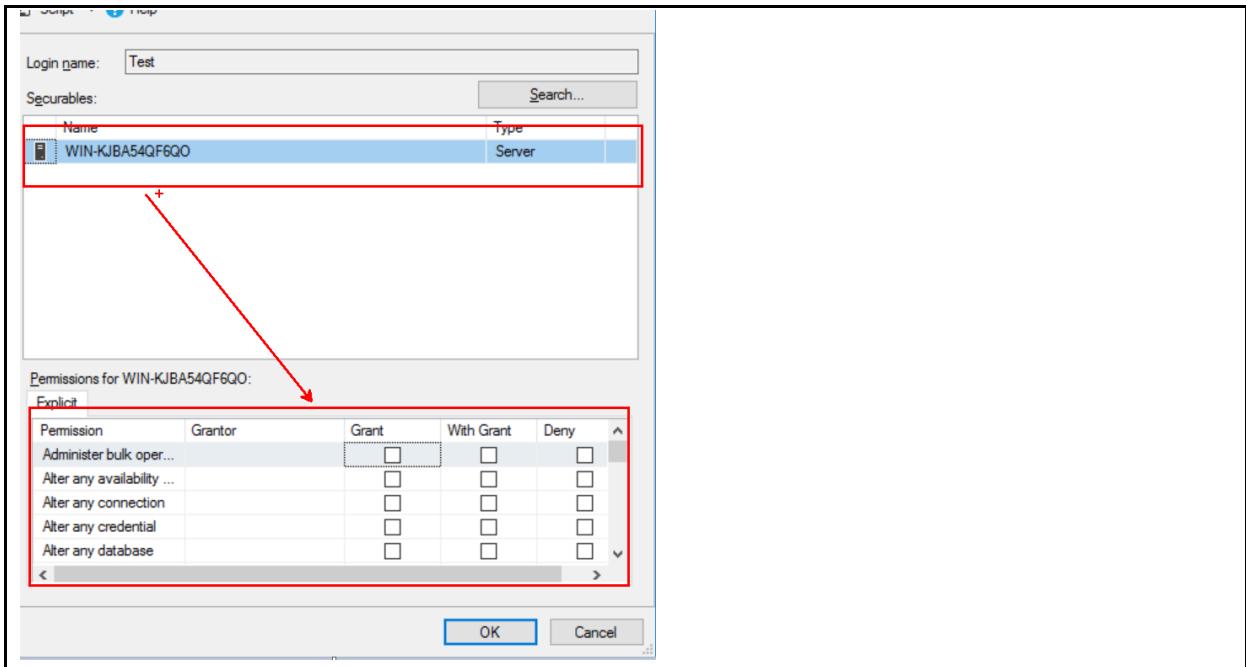
Securables tabı ise oluşturan bu Login'in server düzeyinde hangi yetkilerinin olacağını belirlediğimiz alandır. Burada Search diyerek sunucumuzu seçeriz;



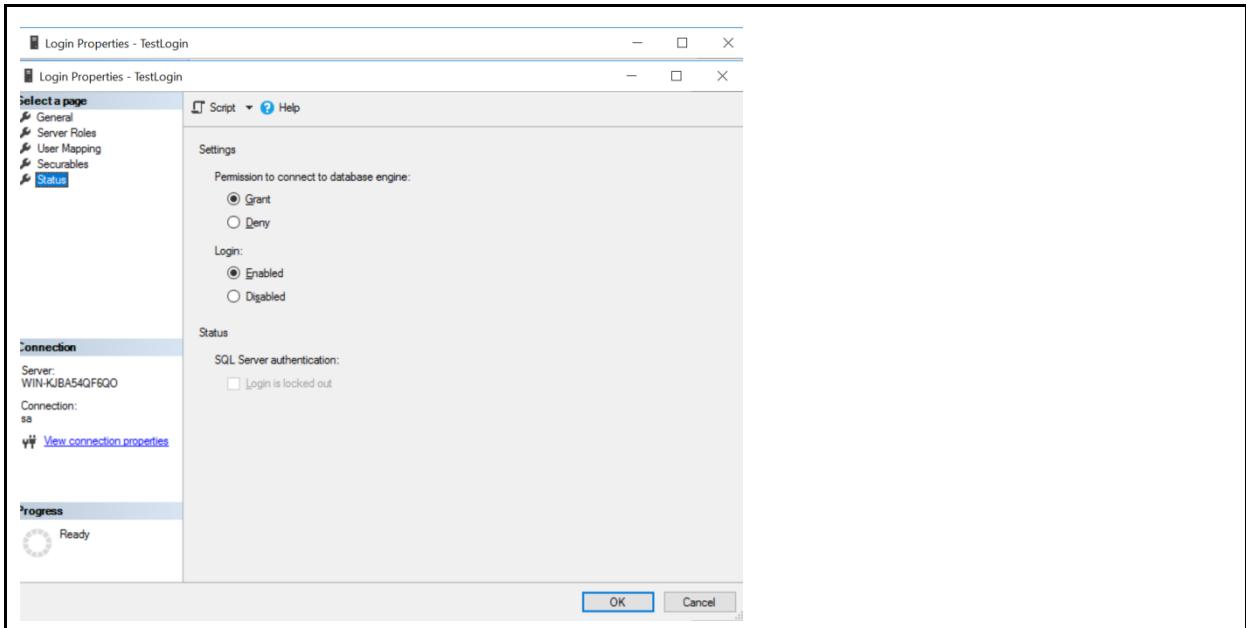
Açılan ekranda Server bazında verebileceğimiz yetkilerin neler olduğunu görebiliriz. Burada da herhangi bir yetki vermeden devam edelim.

Burada örneğin Test kullanıcısı için Sql Connect iznini yani Sql server'a bağlanma iznini verip vermediğimizi belirleyebiliriz. Burada Grant alkanındaki işaretü kaldırıp bu yetkiyi alırsak TestLogin oturumuna bağlı olanlar bu veritabanı sunucusuna bağlanamazlar.

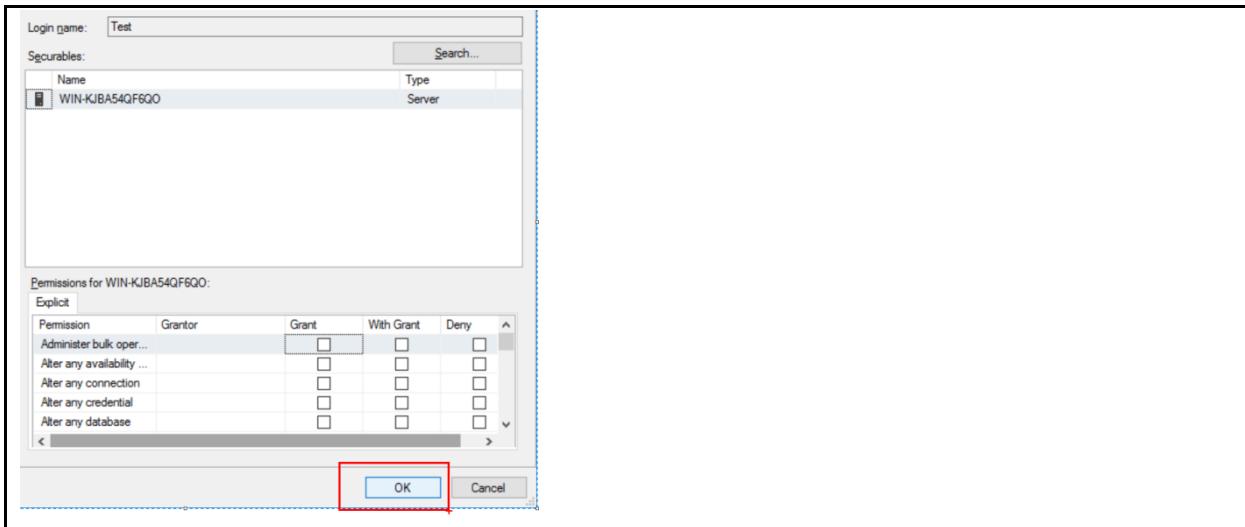
Yine bu alanda Create Any Database alanındaki izin ise bir database oluşturulup oluşturamayacağı ile ilgili izni belirler. Burada birçok konuya ilgili izniler düzenlenlenebilir.



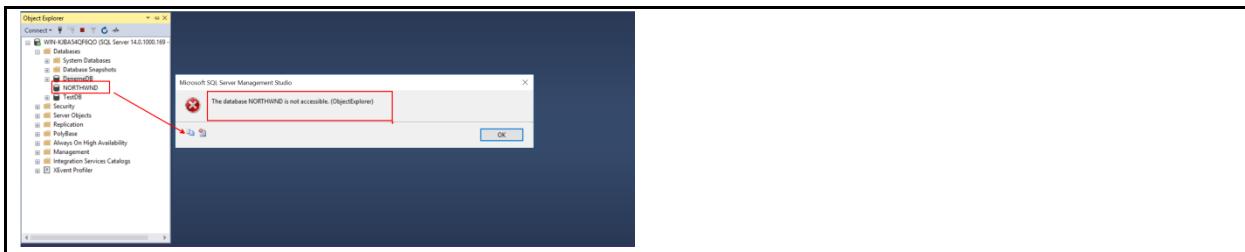
Satus kısmında ise Permission Connect to Database Engine alanı olduğunu görebiliyoruz. Yani bu TestLogin Database veritabanı motoruna bağlanma yetkisi olsun mu olmasın mı şeklinde bir belirleme yapıyoruz. Default ayarda Grand olduğunu yani bu yetkisinin olduğunu görebiliyoruz. Buradaki Login alanı ise yine erişim yetkisinin belirlendiği alandır.



Ok diyerek Login'ınınmizin olmasını sağlıyoruz;

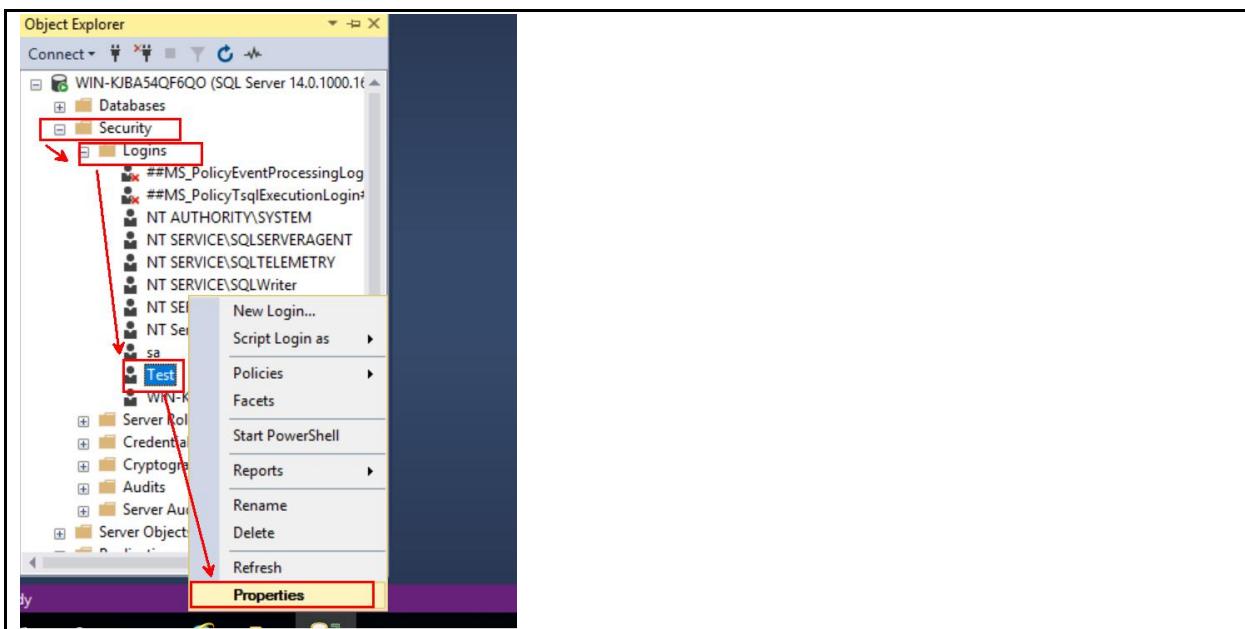


Oluşturduğumuz login hesabı ile Sql Server Management Studio'ya bağlanalım. Şimdi burada herhangi bir veritabanına örneğin Northwind veritabanına tıklayarak veritabanındaki dosyaları, tabloları görüntülemeye çalışalım.

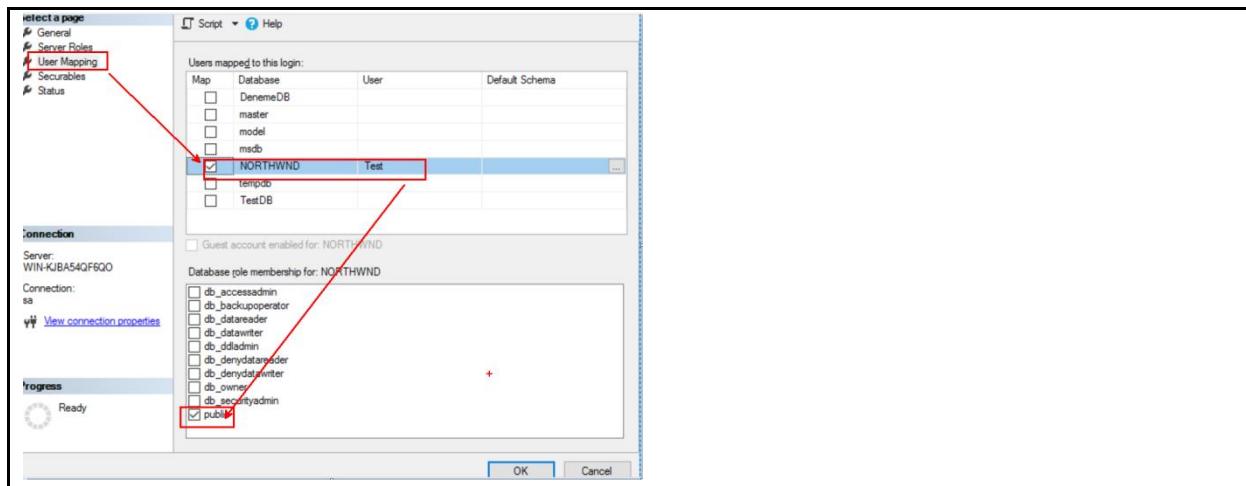


Ancak gördüğünüz gibi, Sql Server Management Studio Northwind veritabanını görüntüleme izin vermedi. Bunun sebebi Login'i oluştururken herhangi bir veritabanında yetki vermemiş olmamızdır. Şimdi tekrar sa kullanıcısıyla Sql Server Management Studio'ya login olarak Northwind veritabanı için bir yetkilendirme yapalım.

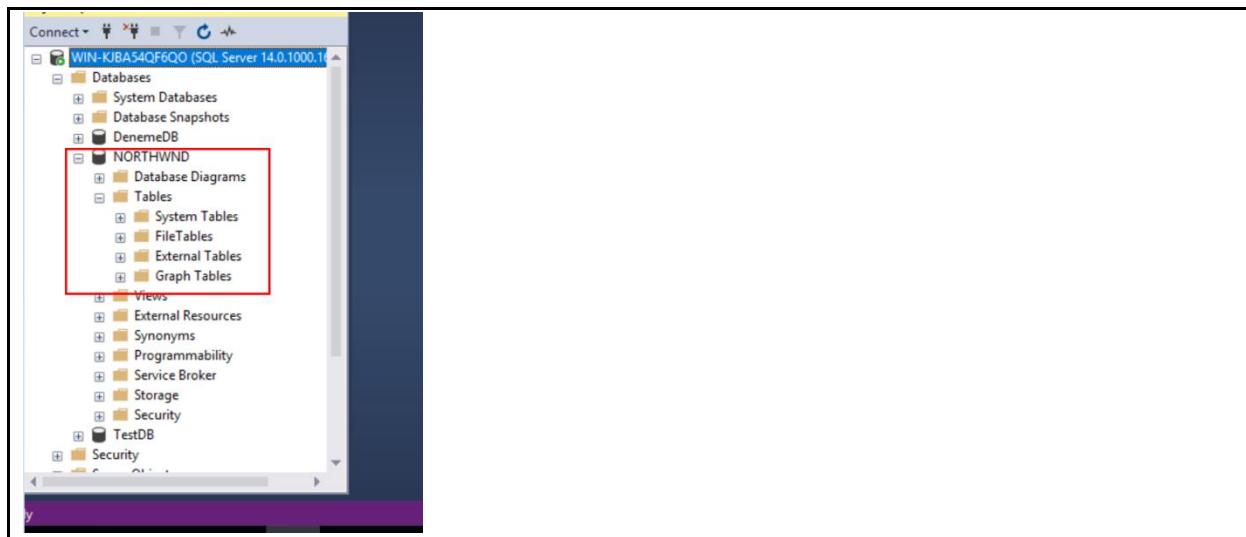
Login üzerine sağ tıklayarak Properties seçeneğini seçiyoruz



Burada User Mapping altında veritabanlarını görüyoruz. Burada Northwind veritabanını seçerek public yetkisini vererek OK diyelim.

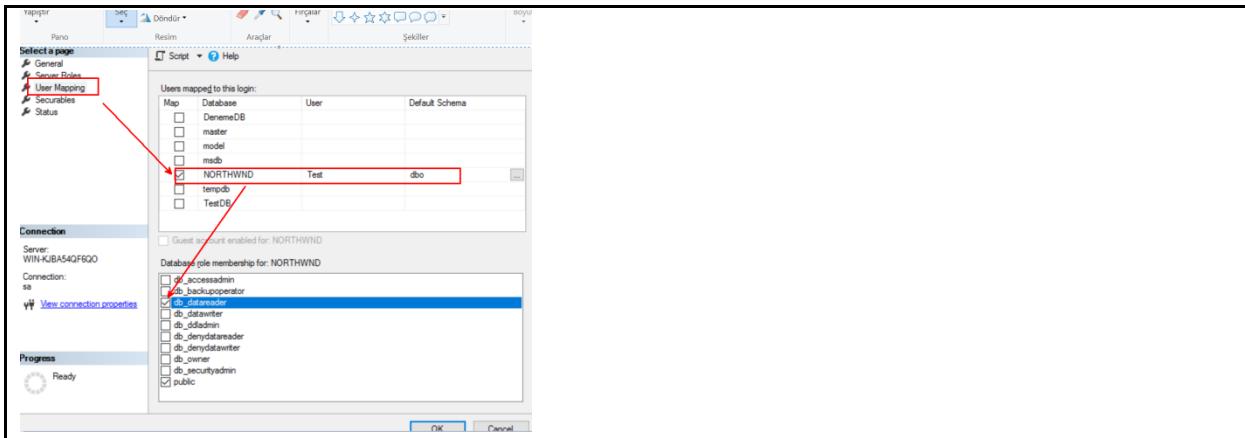


Şimdi Tekrar Test Login'i ile Sql Server Management Studio'ya bağlanarak Northwind veritabanını görüntüleyelim.

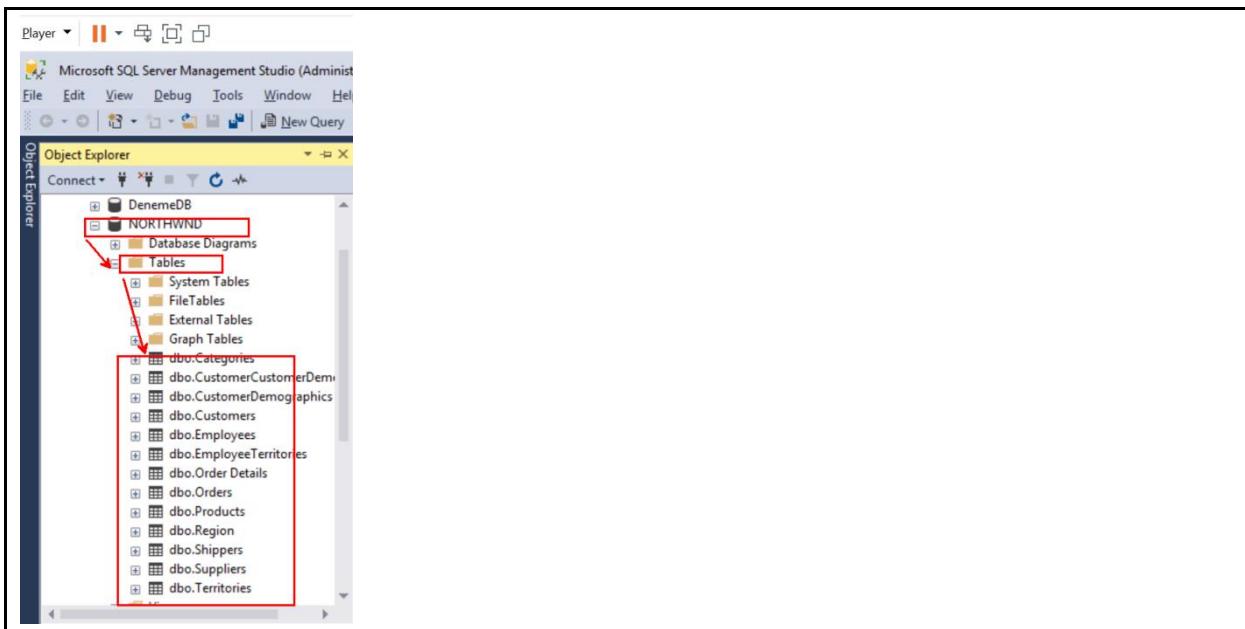


Bu sefer veritabanının dosyalarını görüntüleyebildik. Ancak Tables sekmesini açtığımızda veritabanının içindeki tabloları hala göremiyoruz. Tablo için yetki vermemiz gerekiyor. Tekrar sa kullanıcısı ile Sql Server Management Studio'ya bağlanarak yetkilendirmemizi yapalım

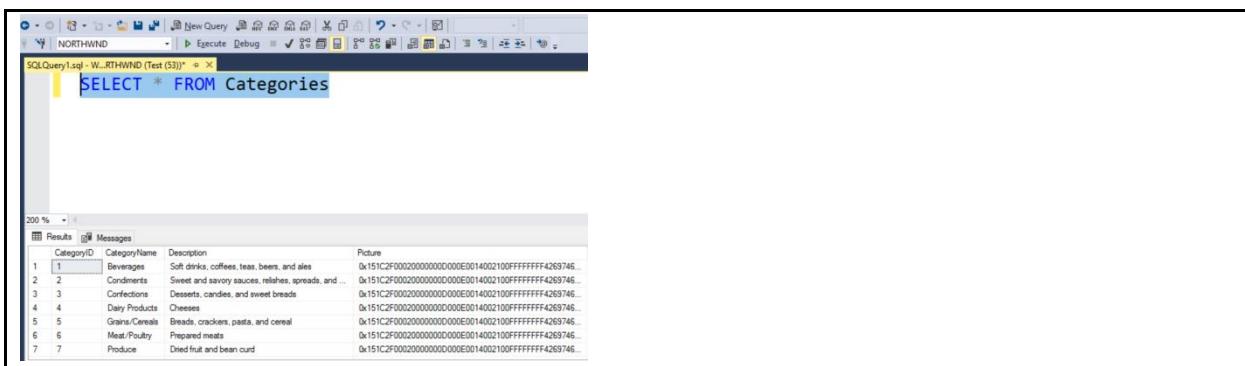
Login üzerine sağ tıklayarak Properties seçeneğini seçiyoruz. Simdi bir tablo üzerinde yetkilendirme işlemi yapalım. Burada User Mapping bölümünde Data Reader yetkisi vererek ve tablolar üzerinde okuma yapabilmesine imkan tanıyalım;



Şimdi Tekrar Test Login'i ile Sql Server Management Studio'ya bağlanarak Northwind veritabanını görüntüleyelim. Artık veritabanının içindeki tabloları görebiliyoruz.



Burada artık bu kullanıcı ile tablolar üzerinde select sorgusu da yapabiliriz.



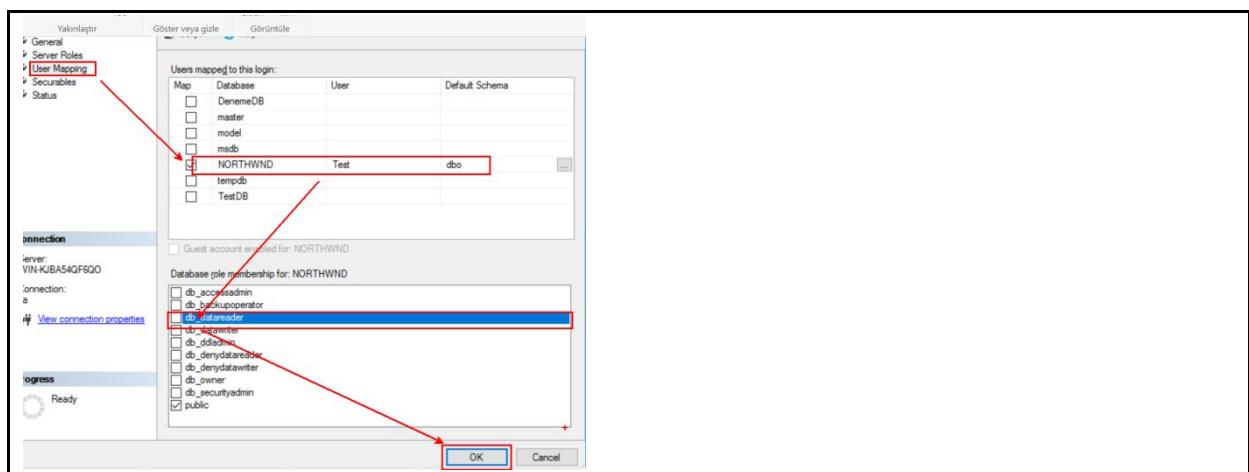
Peki biz bu kullanıcının bütün tabloları görmesini istemiyorsak sadece bazı tablolar, hatta bazı kolonları görebilmesini ve onlar üzerinde işlem yapmasını istiyorsak ne yapmalıyız? Bu yetkileri düzenlemek için tekrar sa kullanıcısı ile Sql Server Management Studio'ya bağlanalım. Öncelikle

kullanıcıımıza server düzeyinde verdiğimiz Data Reader yetkisini iptal etmemiz gereklidir. Bu yetkiye sahip bir kullanıcı veritabanındaki tüm tabloları okuyabilir.

Tekrar sa kullanıcısı ile Sql Server Management Studio'ya bağlanarak yetkilendirmemizi yapalım. Login üzerine sağ tıklayarak Properties seçeneğini seçiyoruz.



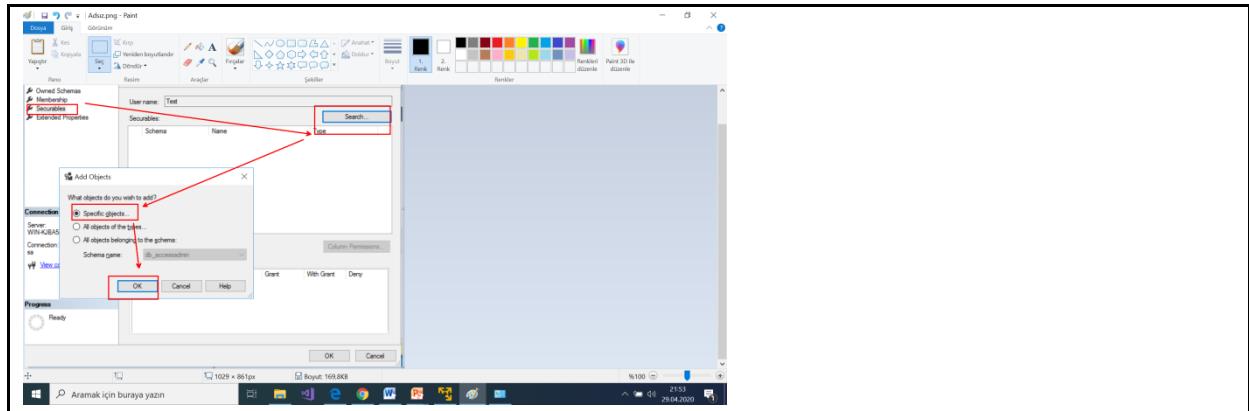
Daha sonra User Mapping tabını seçerek kullanıcıımıza Northwind veritabanı için verdiğimiz Data Reader yetkisini iptal ederek yani buradaki check işaretini kaldırarak Ok diyoruz.



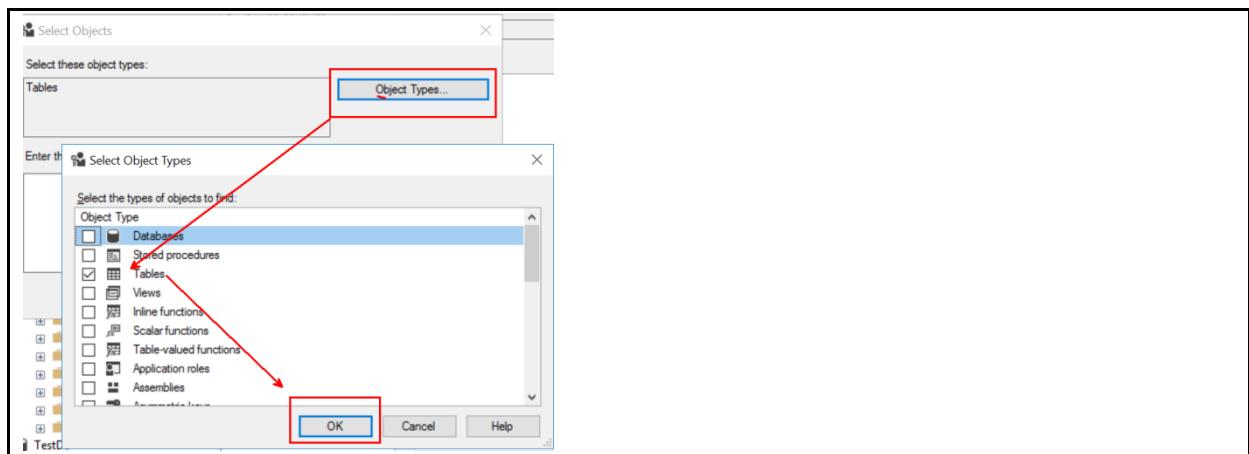
Artık Northwind veritabanının Security klasörü altındaki Users klasörüne giderek burada kullanıcıımıza sağ tık ve Properties sekmesini seçiyoruz. Bu sefer yetkilendirme işlemi, veritabanı bazlı olduğu için işlemi ilgili veritabanı yapacağız.



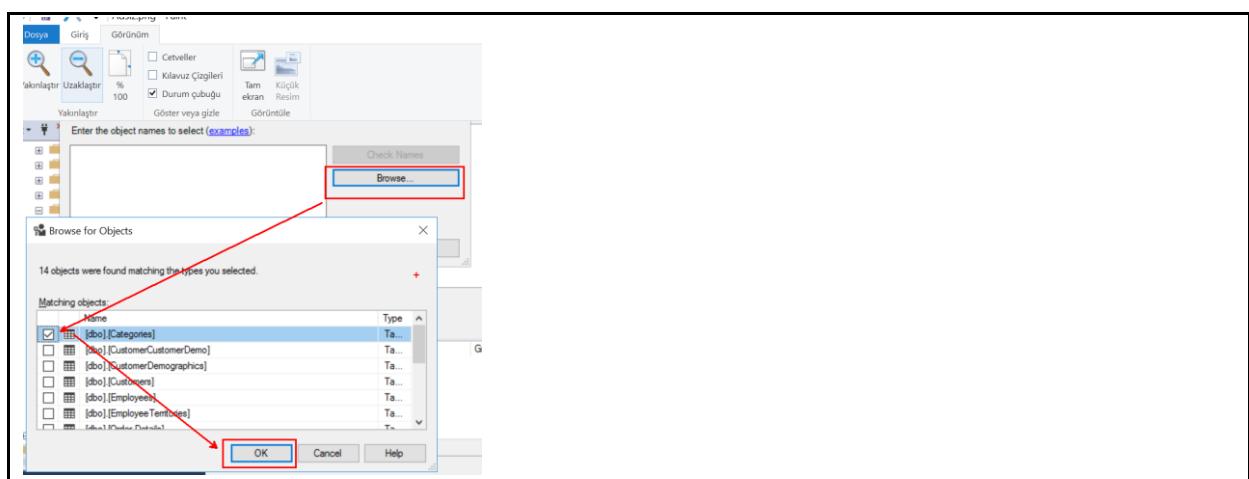
Açılan ekranda Securables tabına tıklayarak buradan Search diyoruz. Karşımıza gelen pencereden Spesific Objects' i seçerek OK diyoruz



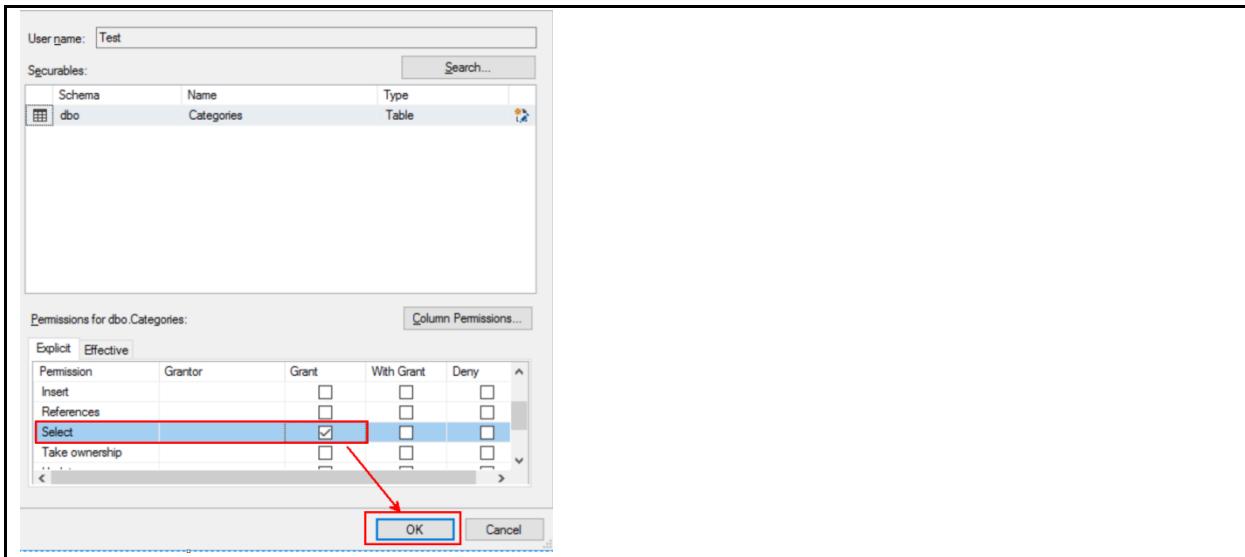
Daha sonra açılan pencerede Object Types diyerek gelen ekranın Tables seçeneğini işaretliyoruz.



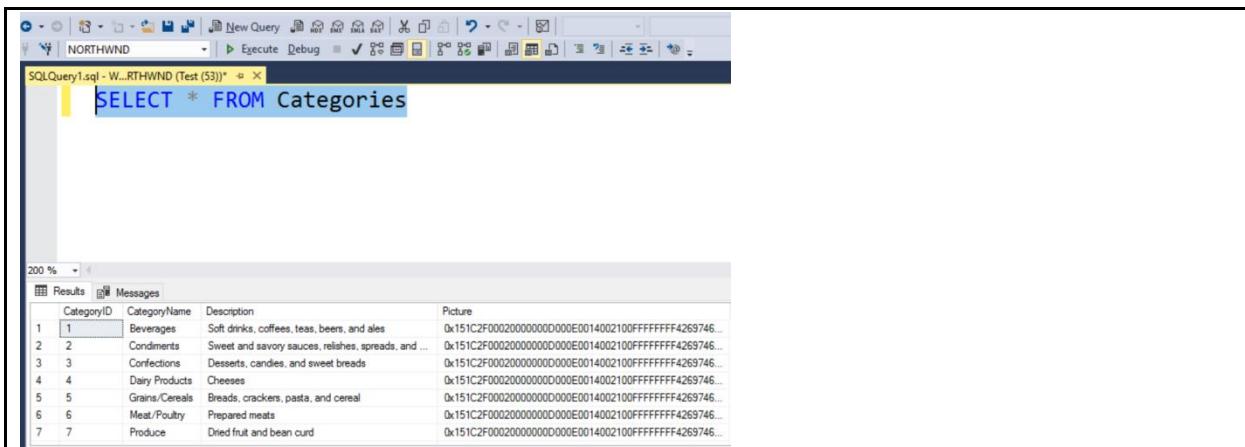
Daha sonra Browse seçeneğine tıkladığımızda Northwind veritabanı üzerindeki tablolarımız görüntülenecektir. Biz burada Categories tablosunu seçerek kullanıcımızın yetkilerini bu tablo üzerinden şekillendirelim;



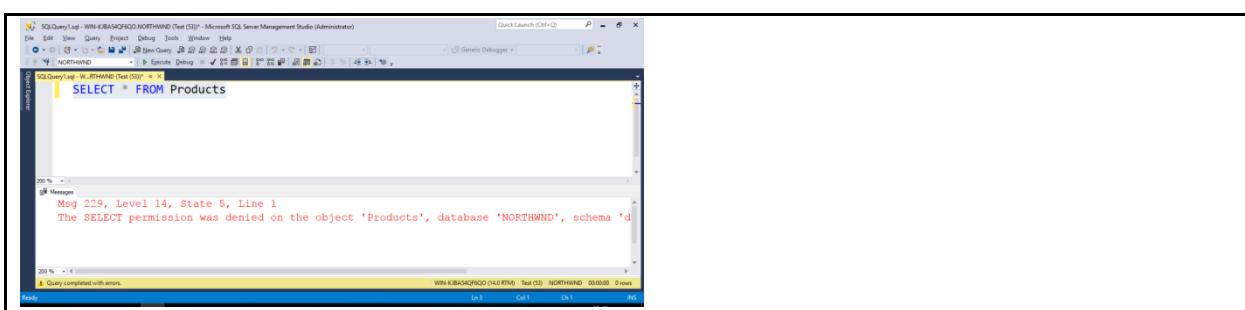
Artık açılan ekranda bu kullanıcın Categories tablosunda ne yapıp ne yapamayacağını belirteceğiz. Örneğin burada SELECT yetkisi vermek için alt tarafta Select seçeneğini bularak GIANT (yani izin ver) check'ini işaretleyelim ve OK diyerek yetki işleminiz bitirelim;



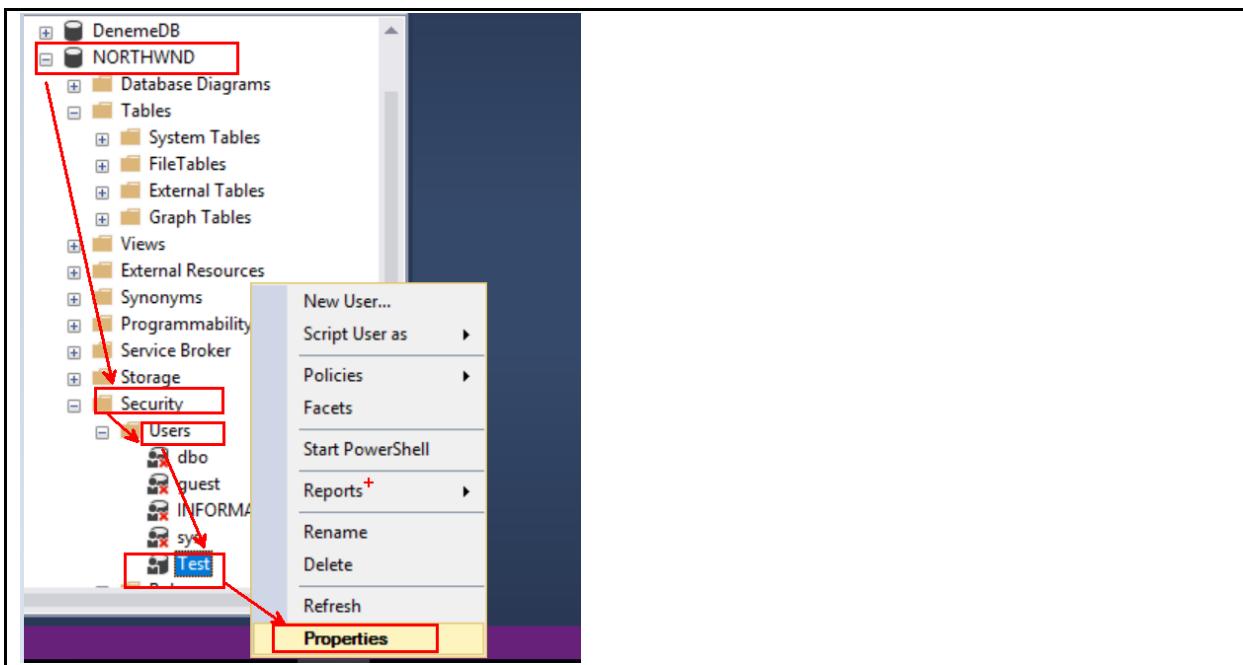
Şimdi tekrar Test kullanıcımızla Sql Server Management Studio'ya bağlanarak Categories tablosu üzerinde bir SELECT sorgusu yazalım



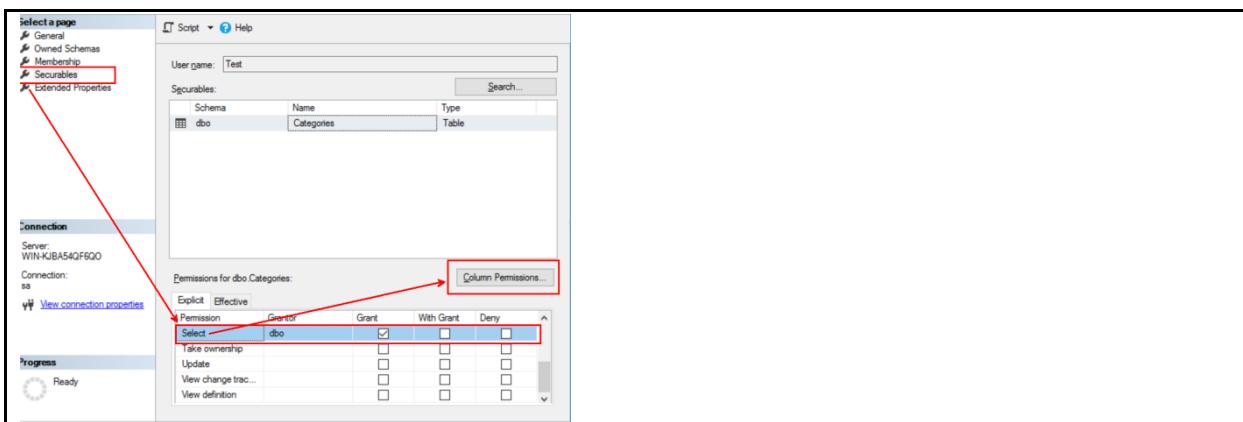
Herhangi bir sorun olmadan sorgumuzu çalıştırıldı. Şimdi de Products tablosu üzerinde bir SELECT sorgusu yazalım.



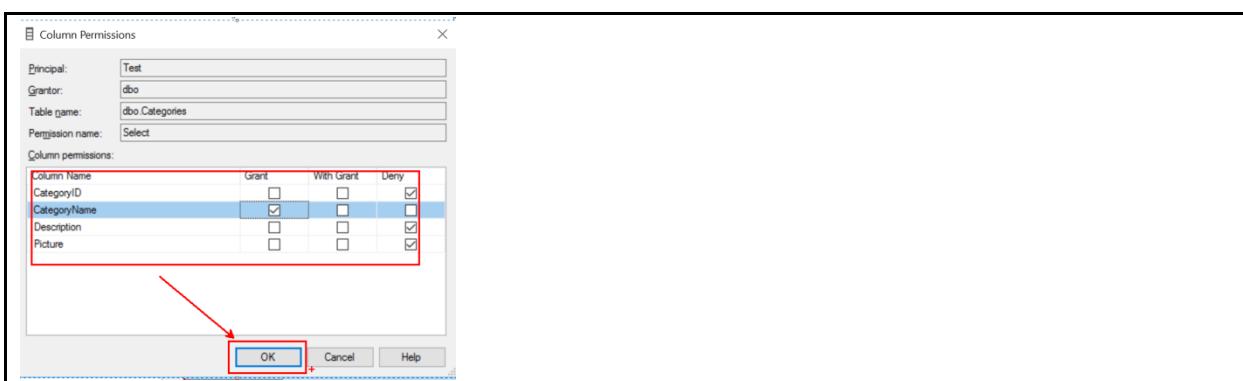
Gördüğü şekilde Products tablosu üzerinde bir SELECT sorgusu yazmamıza izin verilmedi ve yetkiniz yok şeklinde bir hata aldık. Kullanıcıya yalnızca Categories tablosunda yetki verdik. Bu tablo üzerinde verdığımız yetkiyi de kısıtlayabiliriz ve kolon bazında da yetkilendirme yapabiliriz. Şimdi tekrar sa kullanıcısıyla bağlanarak kolon bazında bir yetkilendirme yapalım. Sa kullanıcısıyla login olalım ve;



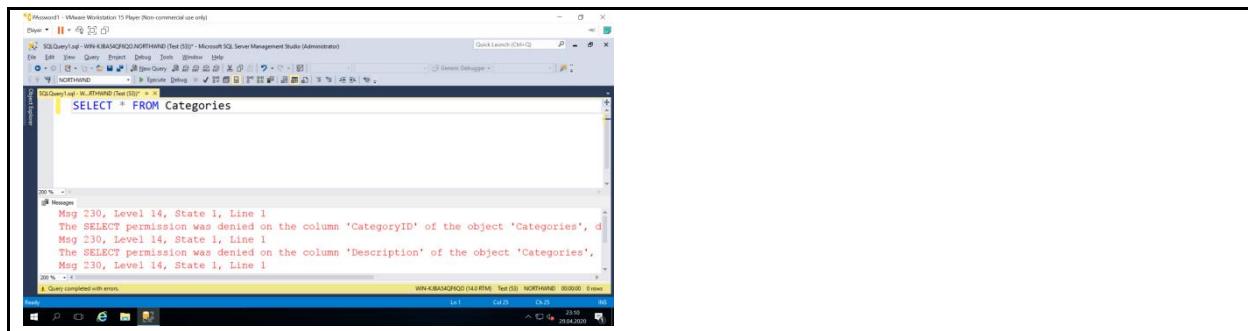
Açılan ekranda Securables menüsünü seçiyoruz ve izinler bölümünde önce Select'e daha sonra ve Column Properties seçeneğine tıklıyoruz.



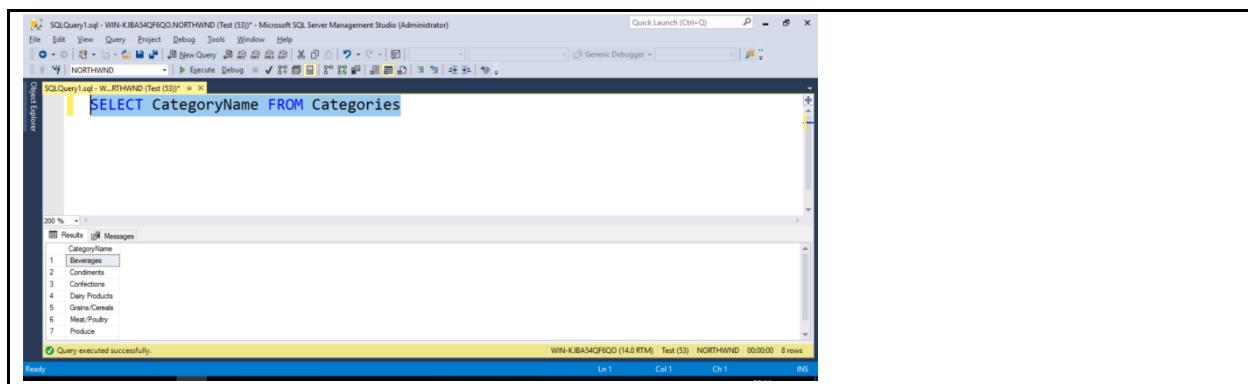
Gördüğü üzere Categories tablomuzda yer alan kolon isimleri listelendi. Burada kullanıcının görmesini istediğimiz ve istemediğimiz kolonları seçebiliriz. Örneğin biz sadece kategori ismini görebilsin başka hiçbir kolonu görmesin istiyoruz. Bu durumda sadece CategoryName alanını GRANT (yani izinli) olarak işaretlememiz lazım. Diğer kolonları için ise DENY (yani izin yok) seçeneğini işaretleyelim;



Şimdi tekrar Test kullanıcımızla Sql Server Management Studio'ya bağlanarak Categories tablosu üzerinde bir SELECT sorgusu yazalım



Göründüğü üzere SELECT * FROM Categories şeklinde yazılan sorgu sonucunda yetkiniz yok yanıtını aldık. Sorgumuzu SELECT CategoryName FROM Categories olarak değiştirirsek;



Bu şekilde sorgu yapmamıza izin verildi. Çünkü biz sadece CategoryName kolonu için verdik

DATA CONTROL LANGUAGE –VERİ KONTROLLERİ-

Veri Kontrol Dili (**DCL**) yani **Data Control Language**, veritabanında bir kullanıcıyı, rolünü ve izinlerini düzenlenmesini sağlayan ifadelerdir.

DCL, bir veritabanı ile ilişkili kullanıcıları ve rollerin izinlerini değiştirmek için yani verilere erişim yetkilerini düzenlemek amacıyla kullanılır.

GRANT Kullanıcıların verileri kullanmasına ve T-SQL komutlarını çalıştırmasına izin verir. Örneğin bir veritabanı üzerinde CREATE TABLE izini vermek için GRANT kullanılması gereklidir. **DENY** Kullanıcıların verileri kullanmasını kısıtlar.

Örneğin herhangi bir kullanıcının CREATE PROCEDURE konutunu çalıştırmasını istemiyorsak bunun için DENY kullanılmalıdır.

REVOKE Daha önce yapılan tüm kısıtlama ve izinleri iptal eder.

Login Oluşturma

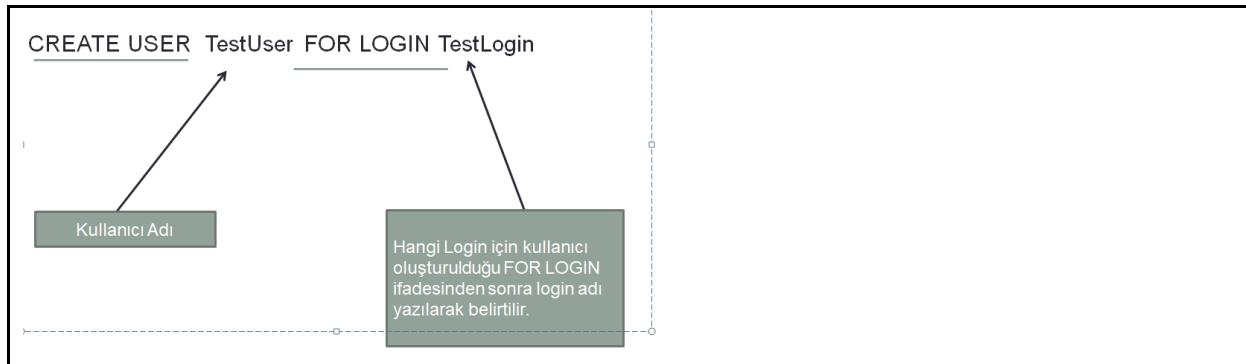
DCL komutlarını test edebilmek için öncelikle bir oturum oluşturmalı ve bunun üzerine yetkili kullanıcılar oluşturarak bunların izinlerini yönetebiliriz;

İlk olarak sunucumuza dışarıdan erişebilmesi için bir login oluşturalım;

```
CREATE LOGIN TestLogin WITH PASSWORD = 'Test123'
```

Şimdi oluşturduğumuz TestLogin isimli Login ile sunucuya erişebilmesi için bir kullanıcı oluşturalım. TestLogin aslında bizim için bir havuz oluşturur. Biz bir oturum için birden fazla kullanıcı atayabiliriz. Aslında bir tür gruplama yaparak Server yönetimini daha kolay şekilde organize etmek amacıyla yapılır.

Kullanıcı oluşturmak için

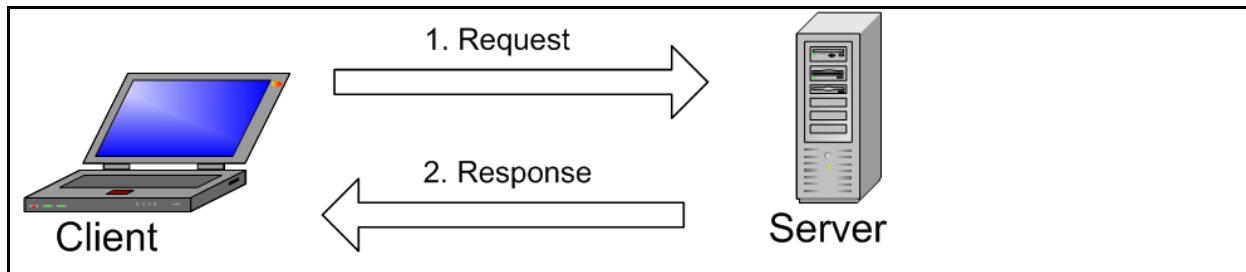


GRANT İle Yetki Vermek

Kullanıcının verileri kullanmasına ve SQL komutlarını çalıştırmasına izin veren komut Grand'dır. Söz dizimi aşağıdaki şekildedir.

```
GRANT <ALL veya izinler>
ON <izin_verilenler>
TO <hesaplar>
```

SQL INJECTION



Veritabanı programcıları veri ekleme, güncelleme, düzenleme ve silme gibi işlemler için istemci ve sunucu tarafında özel yazılımlar geliştirirler.

Saldırganlar, SQL kodları arasına kendi istedikleri ve gerçek sorgunun amacından farklı olarak çalışmasını sağlayacak komutlar enjekte (ekler) edebilirler.

SQL Injection, veritabanı sunucusuna gönderilen sorgu bloğuna, yeni bir sorgu sıkıştırma yöntemi ile gerçekleşir.

Sql Injection özet olarak querystringler ile sorgulara müdahalede bulunup veritabanına erişerek veritabanındaki bilgileri görüntülemek şeklinde açıklanabilir.

Yani biz bu saldırı yöntemi ile; zararlı olmayan sql cümlelerinin arasına zararlı olan sql cümleleri yerleştiriyoruz.

Sql Injection saldırının mantığı oldukça basittir.

Diyelim SQL kullanan bir siteye Admin kullanıcı adı ve 12345 şifresiyle giriş yapıyorsunuz. Giriş butonuna tıkladığınız anda sitenin yazılımı veritabanına aşağıdaki gibi bir sorgu gönderecektir

```
SELECT * FROM Users WHERE Name = 'Admin' AND Password = '12345'
```

Eğer şifre doğru ise, yani bu sorguda bir sonuç bulunursa sistem girişinize izin verecek, bulunamazsa izin vermeyecektir.

Ancak biz şu şekilde bir sorgu gönderirsek;

```
SELECT * FROM Users WHERE Name = 'Admin' AND Password = '12345' OR '1' = '1'
```

Şimdi, bu ikinci sorgumuzu inceleyelim;

Bu sorguda 3 şart bulunmaktadır. 2 si AND (VE) ile birbirine bağlanmış ve 1 de OR(VEYA) ile bağlanan 3. şart var.

Eğer bir OR şartı varsa bu AND ile bağlanan şartları ezer.

'OR 1=1' SQL dilinde her zaman true döndürür (olumlu sonuç döndürür). Bu yüzden SQL programı bütün kayıtları listeleyeceğinden, yazılım doğru giriş yapıldı kabul edecektir.

'OR 1=1' ifadesi, önceki koşul gerçekleşmese bile, 1=1 gerçekleşiyorsa true döndür anlamına gelmektedir.

Bu bir güvenlik açığıdır ve biz bu açığı kullanarak çok basit şekilde sisteme girebiliriz.

Northwind veritabanında basit bir sorgu çalışıralım;

```
SELECT * FROM Products WHERE ProductID = 1
```

Yukarıdaki sorgu Products tablosundan ProductID değeri 1 olan ürünü getirecektir.

The screenshot shows a SQL query window with the following content:

```
SELECT * FROM Products WHERE ProductID = 1
```

Below the query, the results are displayed in a table:

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18.00	39	0	10	0

Ancak sorgumuzu aşağıdaki şekilde değiştirerek çalıştırırsak;

```
SELECT * FROM Products WHERE ProductID = 1 OR 1 = 1
```

Aşağıdaki şekilde bir sonuç elde ederiz;

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
7	Uncle Bob's Organic Dried Pears	3	7	12 - 18 pkgs	30.00	0	10	0	0
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40.00	6	0	0	0
9	Mehi Kobe Niku	4	6	18 - 500 g pkgs	97.00	29	0	0	1
10	Kula	4	8	12 - 200 ml jars	31.00	31	0	0	0
11	Queso Cabriles	5	4	1 kg pkgs	21.00	22	30	30	0
12	Queso Manchego La Pastora	5	4	10 - 500 g pkgs	38.00	96	0	0	0
13	Tofu	6	8	2 kg cartons	0.50	24	0	5	0
14	Tofu	6	7	40 - 100 g pkgs	23.25	35	0	0	0
15	Genen Shoushu	6	2	24 - 250 ml bottles	15.50	39	0	5	0
16	Pavlova	7	3	32 - 500 g boxes	17.45	29	0	10	0
17	Alice Mutton	7	6	20 - 1 kg tins	39.00	0	0	0	1

Sorguya dışarıdan müdahale ettiğimiz için tüm kayıtlar listelendi ve 77 adet kayıt getirildi

SQL Injection ile bir SELECT sorgusu gerçekleştirken, sorgu arasında bir veri eklemesi gerçekleştirilebilir. Öncelikle

```
SELECT * FROM Categories
```

diyerek kaç adet kategori kaydı olduğunu görelim

Gördüğü üzere Categories tablosunda 8 adet kategori kaydı bulunmaktadır.

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x151C2F0002000000D000E0014002100FFFFFF4269746...
2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x151C2F0002000000D000E0014002100FFFFFF4269746...
3	Confections	Desserts, candies, and sweet breads	0x151C2F0002000000D000E0014002100FFFFFF4269746...
4	Dairy Products	Cheeses	0x151C2F0002000000D000E0014002100FFFFFF4269746...
5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x151C2F0002000000D000E0014002100FFFFFF4269746...
6	Meat/Poultry	Prepared meats	0x151C2F0002000000D000E0014002100FFFFFF4269746...
7	Produce	Dried fruit and bean curd	0x151C2F0002000000D000E0014002100FFFFFF4269746...
8	Seafood	Deneme	0x151C2F0002000000D000E0014002100FFFFFF4269746...

Aşağıdaki kod ise Sql Injection yöntemiyle Categories tablosuna bir kayıt eklemektedir.

```
SELECT * FROM Categories  
WHERE CategoryID = 1;  
INSERT INTO Categories(CategoryName)  
VALUES ('Bilgisayar');
```

Gördüğü sorgu basit bir Select sorgusu gibi başlamakta CategoryID değeri 1 olan kaydı istemektedir. Ancak sonrasında sorguya bir INSERT INTO eklenmiştir. Sorgumuzu çalıştırıldığımızda;

Gördüğü üzere sorgu sonucunda INSERT INTO komutuyla ilgili herhangi bir sonuç gösterilmemekte, sadece Select sorgusunun sonucu olarak CategoryID değeri 1 olan ürün listelenmektedir.

```
SELECT * FROM Categories
WHERE CategoryID = 1;
INSERT INTO Categories(CategoryName)
VALUES ('Bilgisayar');
```

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x151C2F0002000000D000E0014002100FFFFFF4269746...

Ancak Categories tablosunu

```
SELECT * FROM Categories
```

diyerek tekrar görüntülediğimizde tabloya yeni bir kayıt eklendiği görülmektedir;

```
SELECT * FROM Categories
```

CategoryID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees, teas, beers, and ales	0x151C2F0002000000D000E0014002100FFFFFF4269746...
2	Condiments	Sweet and savory sauces, relishes, spreads, and ...	0x151C2F0002000000D000E0014002100FFFFFF4269746...
3	Confections	Desserts, candies, and sweet breads	0x151C2F0002000000D000E0014002100FFFFFF4269746...
4	Dairy Products	Cheeses	0x151C2F0002000000D000E0014002100FFFFFF4269746...
5	Grains/Cereals	Breads, crackers, pasta, and cereal	0x151C2F0002000000D000E0014002100FFFFFF4269746...
6	Meat/Poultry	Prepared meats	0x151C2F0002000000D000E0014002100FFFFFF4269746...
7	Produce	Dried fruit and bean curd	0x151C2F0002000000D000E0014002100FFFFFF4269746...
8	Seafood	Deneme	0x151C2F0002000000D000E0014002100FFFFFF4269746...
9	Bilgisayar	NULL	NULL

Basit bir sorgu ile veritabanı içinden bir tablo silinebilir. Test yapmak için Server üzerinde yeni bir veritabanı ekleyelim. Ben eklediğim veritabanına SecurityDB ismini veriyorum;

```
CREATE DATABASE SecurityDB
```

Commands completed successfully.

Şimdi de Products adlı bir tablo ekleyelim. Kolonlarımız Id (PrimaryKey olacak), ProductName (VarChar(50) olacak), Price(INT olacak) şeklinde düzenlenecek.

WIN-KJBA54QF6QO\yDB - dbo.Table_1*		
Column Name	Data Type	Allow Nulls
Id	int	<input type="checkbox"/>
ProductName	varchar(50)	<input checked="" type="checkbox"/>
Price	int	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Şimdi Products tablomuza bir miktar kayıt ekleyelim;

Id	ProductName	Price
1	Bilgisayar	1000
2	Telefon	500
3	Tablet	750
NULL	NULL	NULL

SELECT * FROM Products şeklinde sorguladığımızda ürünlerimizin geldiğini görebiliriz

SELECT * FROM Products		
 Results  Messages		
Id	ProductName	Price
1	Bilgisayar	1000
2	Telefon	500
3	Tablet	750

Basit bir sorgu ile veritabanı içinden bir tablo silinebilir. Aşağıdaki şekilde sorgumuzu yazalım;

```
SELECT * FROM Users
WHERE Name = '\'; DROP TABLE Products;
```

Sorgumuz basitçe Users tablosundan Name alanı '/' olan kullanıcıyı getirmek için yazılmıştır. Ancak sorgu sonuna bir DROP ifadesi eklenerek Sql Injection saldırısı yapılmaktadır. Sorgumuzu çalıştırduğumızda;

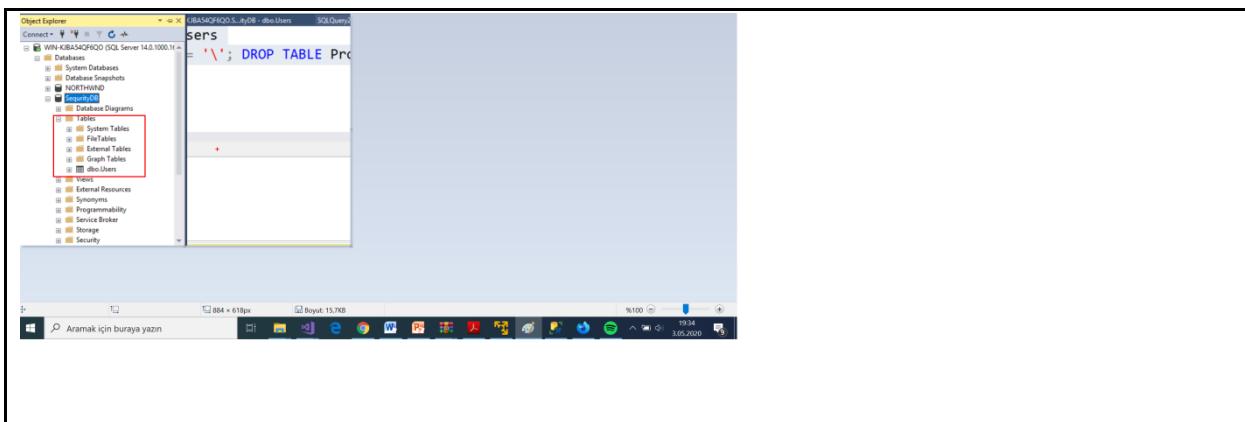
```

SELECT * FROM Users
WHERE Name = '\'; DROP TABLE Products;

```

Gördüğü üzere Users tablosunda Name alanı '/' olan hiçbir kullanıcı olmadığı için herhangi bir kayıt listelenmemiştir.

Ancak sorgunun sonuna eklenen `DROP TABLE Products;` ifadesi nedeniyle Products tablonu silinmiştir. Database'imizi refresh yaparsak Products tablosunun artık olmadığı görülecekti;



Sql Injection ile yapılabilecek en önemli saldırılarından biri, veritabanı sunucusunu kapatmaktadır. Bir e-ticaret sitesinin veritabanı sunucusunu kapatmak oldukça zarar verebilecek bir iştir. Aşağıdaki sorgumuzu çalıştıralım.

```

SELECT UserName FROM Users WHERE UserName = '';
SHUTDOWN WITH NOWAIT;

```

Sorgumuz basitçe Users tablosundan Name alanı '/' olan kullanıcıyı getirmek için yazılmıştır. Ancak sorgu sonuna bir `SHUTDOWN WITH NOWAIT` ifadesi eklenerek Sql Injection saldırısı yapılmakta ve sunucu kapatılmaktadır. Sorgumuzu çalıştırduğumızda;

Gördüğü üzere Users tablosunda Name alanı “” olan hiçbir kullanıcı olmadığı için herhangi bir sonuç gösterilmemiştir.

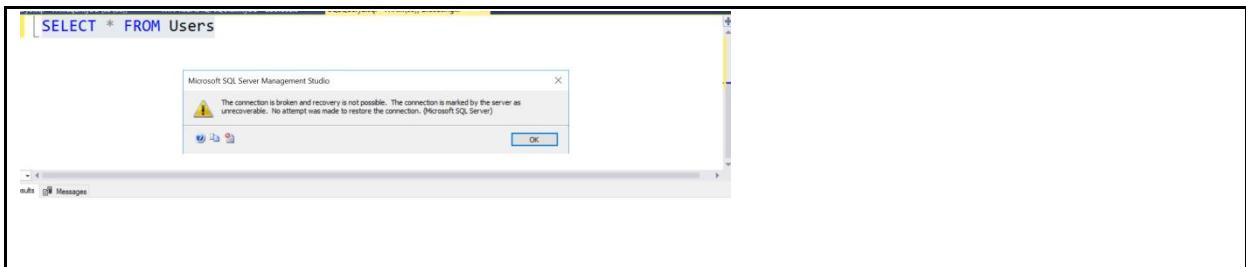
```
SELECT UserName FROM Users WHERE UserName = '';  
SHUTDOWN WITH NOWAIT;
```

Results Messages
UserName

Ancak sorgunun sonuna eklenen SHUTDOWN WITH NOWAIT; ifadesi nedeniyle Sql Server kapatılmıştır. Örneğin Users tablosu için bir sorgu yazarak bunu görebiliriz.

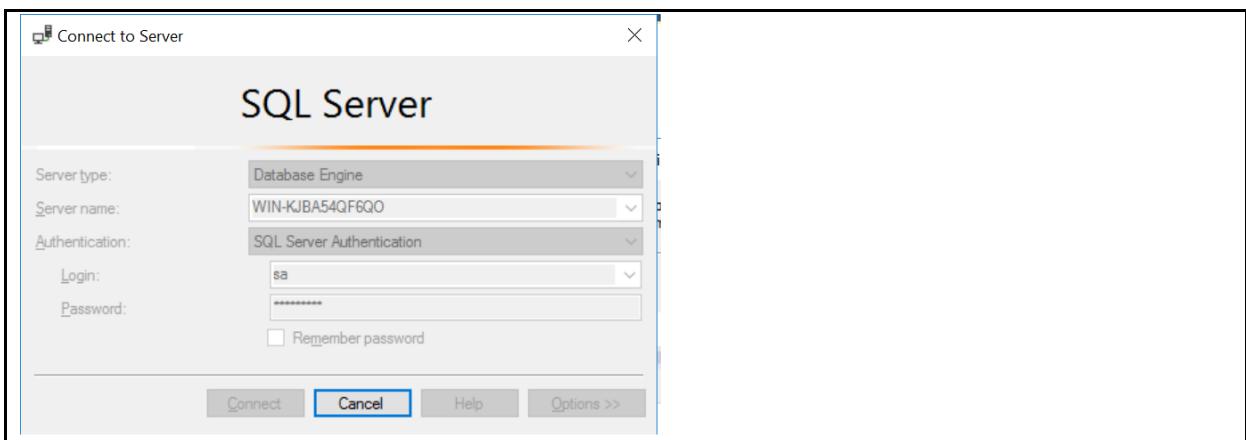
```
SELECT * FROM Users
```

Sorguyu çalıştığımızda aşağıdaki ekran çıkacaktır



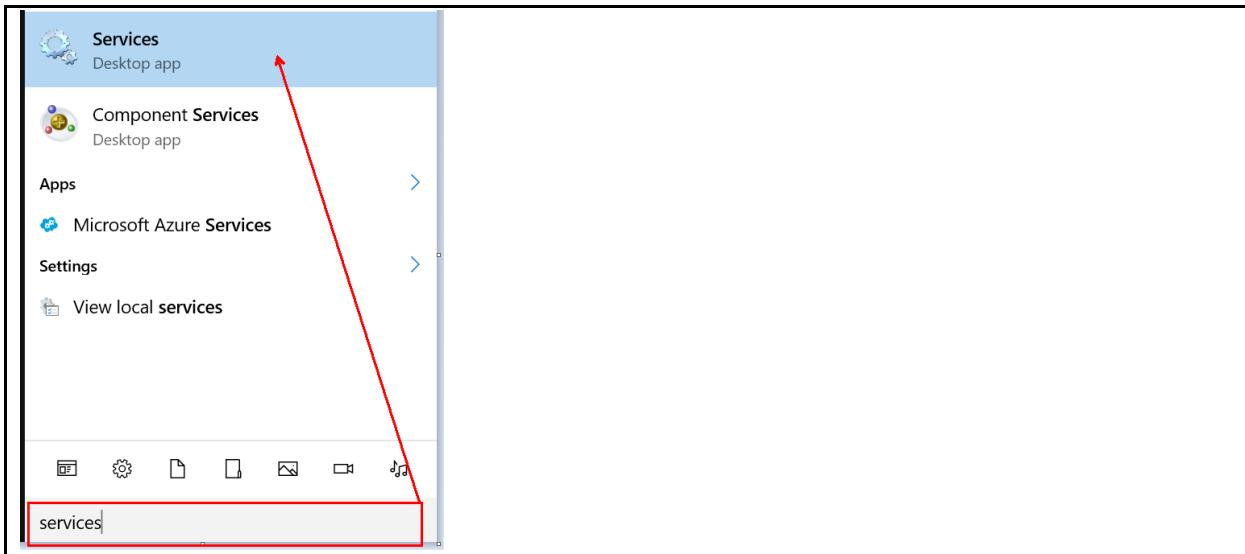
Sql Server'ımız kapalı olduğu için sorgumuz çalıştırılamamaktadır.

Üstelik tekrar Login olmaya çalıştığımızda da buna izin vermemektedir. Çünkü Sql Server bütünüyle kapatılmıştır.

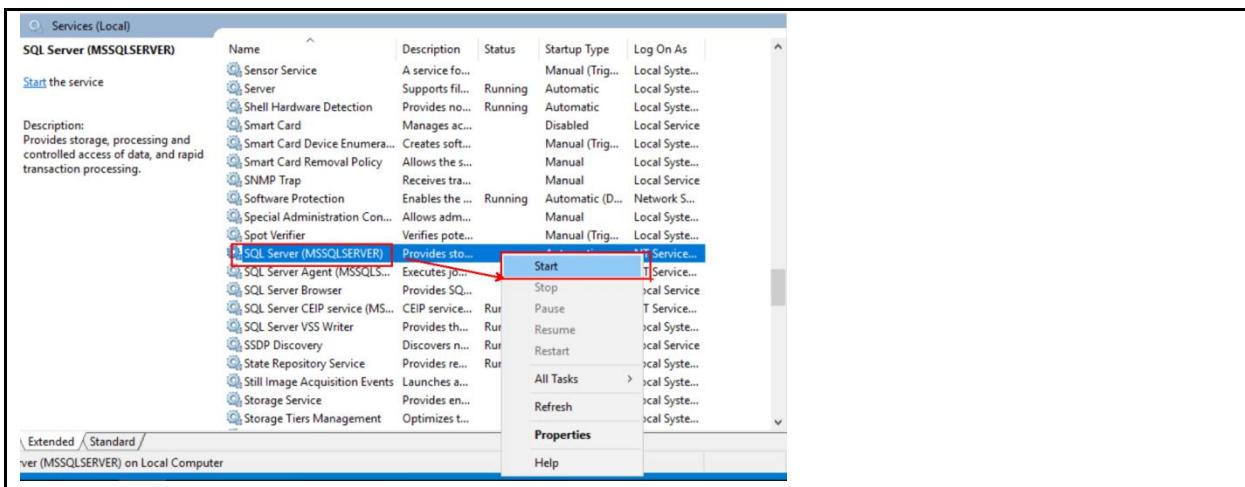


Login yapmak istediğimizde ekran bu şekilde kalacak ancak Sql Server Management Studio'ya giriş yapamayacağız. Sql Server'i tekrar başlatmaktan başka yolumuz yok.

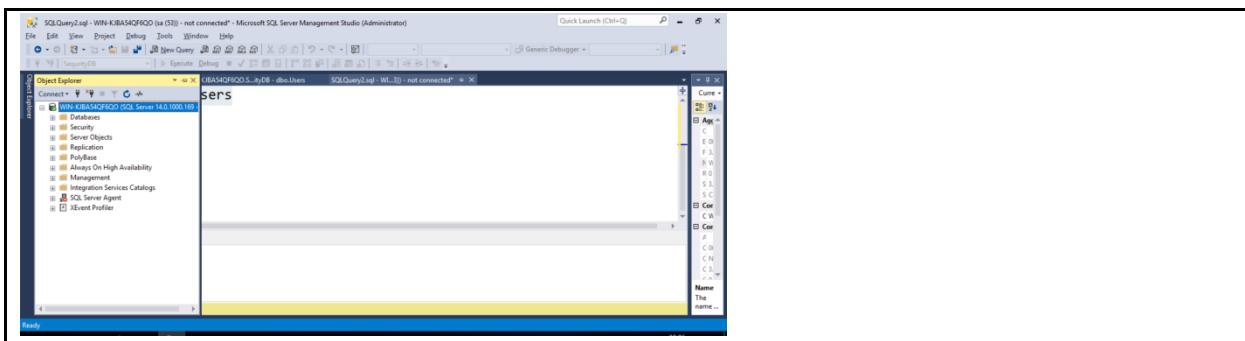
Sql Server'i tekrar başlatmak için Başlat'a Services yazıyoruz.



Services (Hizmetler) alanını açarak burada SQL Server(MSSQLSERVER) alanını buluyoruz. Buraya sağ tık ve start dediğimizde Sql Sevrer’ımız tekrar başlayacak ve biz Sql Server Management Studio’ya giriş yapabileceğiz,



Şimdi Sql Server Management Studio’ya bağlanabildik



Sql Injection Savunma Mekanizmaları

Kullanmadığınız stored procedure'leri kaldırın

master..Xp_cmdshell, xp_startmail, xp_sendmail, sp_makewebtask vb.

Karakterler filtrelenmeli

Yaygın SQL Enjeksiyon saldırıları SQL deyimlerinin girdilerdeki gereksiz (') tırnak işaretleri yardımıyla yeniden oluşturulması sayesinde yapılır.

Küçük bir filtreleme fonksiyonu veya tek tırnağı çift tırnağa çeviren bir fonksiyon muhtemel bir saldırıyı engellemek için yeterli olabilir. Gerçekleştirdiğimiz SQL Injection test çalışmalarının birçoğunda özel karakterler kullandık. Bu karakterlerin çoğu, SQL Server tablosundaki veri içerisinde kullanılmayacaktır. Örneğin; bir isim ya da e-mail bilgisi içerisinde tek tırnak ya da boşluk gibi karakterler kullanılmaz. İlgili sütunlara gönderilecek bu tür verilerin filtrelenmesi gereklidir. Genel olarak yazılımlarda özel karakterler filtrelenirler. İyi bir güvenlik için bu karakterlerin filtrelenmesi, SQL Injection saldırılarını önlemede önemli bir fayda sağlar.

Kayıt uzunluklarını sınırlandırmalı

SQL saldırılarında sorgu kayıt uzunluğu önemlidir. Tablodaki bir sütun için 10 karakterlik bir uzunluk ayrıldıysa, sorguların da bu karakter uzunluğundan fazla bir değeri kabul etmemesi gereklidir.

Veri tiplerini kontrol edilmeli

Kayıt uzunlığında olduğu gibi, uygulama katmanı, veritabanı katmanı ve ara katmanlardaki aynı işi yapan tüm veri tipleri aynı olmalıdır. Kullanıcıdan sayısal bir değer alınması gerekiyorsa, metinsel ya da bir tarih türünü değer olarak almamalıdır. Bu işlemin uygulama katmanındaki yazılım ile kontrol edilmesi gereklidir.

Yetkiler Sınırlanılmalı

Bölümün ilerleyen kısımlarında inceleyeceğimiz izinler ve yetkilerin minimum seviyede olması önerilir. Bir kullanıcı, sadece **SELECT** ile veri seçme işlemi gerçekleştirmeye sahipse; **INSERT**, **UPDATE** ve **DELETE** gibi izinlere sahip olmaması gereklidir.

Mümkün olduğunda Stored Procedure kullanılmalı

SQL Injection'ı engellemek için bilinen yanlışlardan biri de Stored Procedure kullanıldığı takdirde SQL Injection saldırılarına maruz kalınmayacağıdır. Stored Procedure kullanımı bazı saldırıları yöntemlerini kısıtladığı için kod enjektesini de kısıtlayacaktır. Ancak önemli olan, prosedür içerisindeki kodların, enjeksiyona uğramayacak şekilde güvenli kod yazım teknikleriyle geliştirilmesidir.