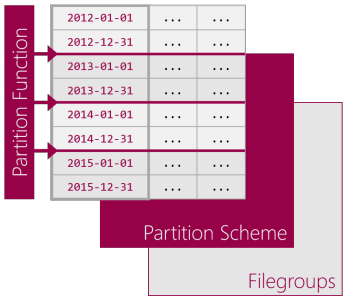
SQL Server Büyük Tabloları Bölmek



Bir güzel SQL Server özelliği ile daha beraberiz. Veri tabanları işlevleri gereği milyonlarca hatta milyarlarca kayıt tutabiliyorlar. Bu kayıtlardan bazıları üzerinden zaman geçtikçe sorgulanmayan veriler olmaya başlıyor. Örneğin online alışveriş sitelerinde üç dört yıl önceki siparişler zorunlu bir işlem olmadığı sürece sorgulanmamaya başlar. Ama sorgularda bu kayıtlar da sorgulandığından sorgular yavaş çalışmaya başlar. Bu durumlar da SQL Server Partitioned Table özelliği ile büyük tabloları bir özelliğine göre farklı dosyalara ayırabiliriz. Böylece gelen sorguda bütün verileri değil de sadece ilgili verinin olduğu dosya üzerinde çalışılır. Saatlerce sürebilecek sorgular bu sayede saniyeler içerisinde bite bilmektedir.

Bölünmüş Tablo Oluşturma

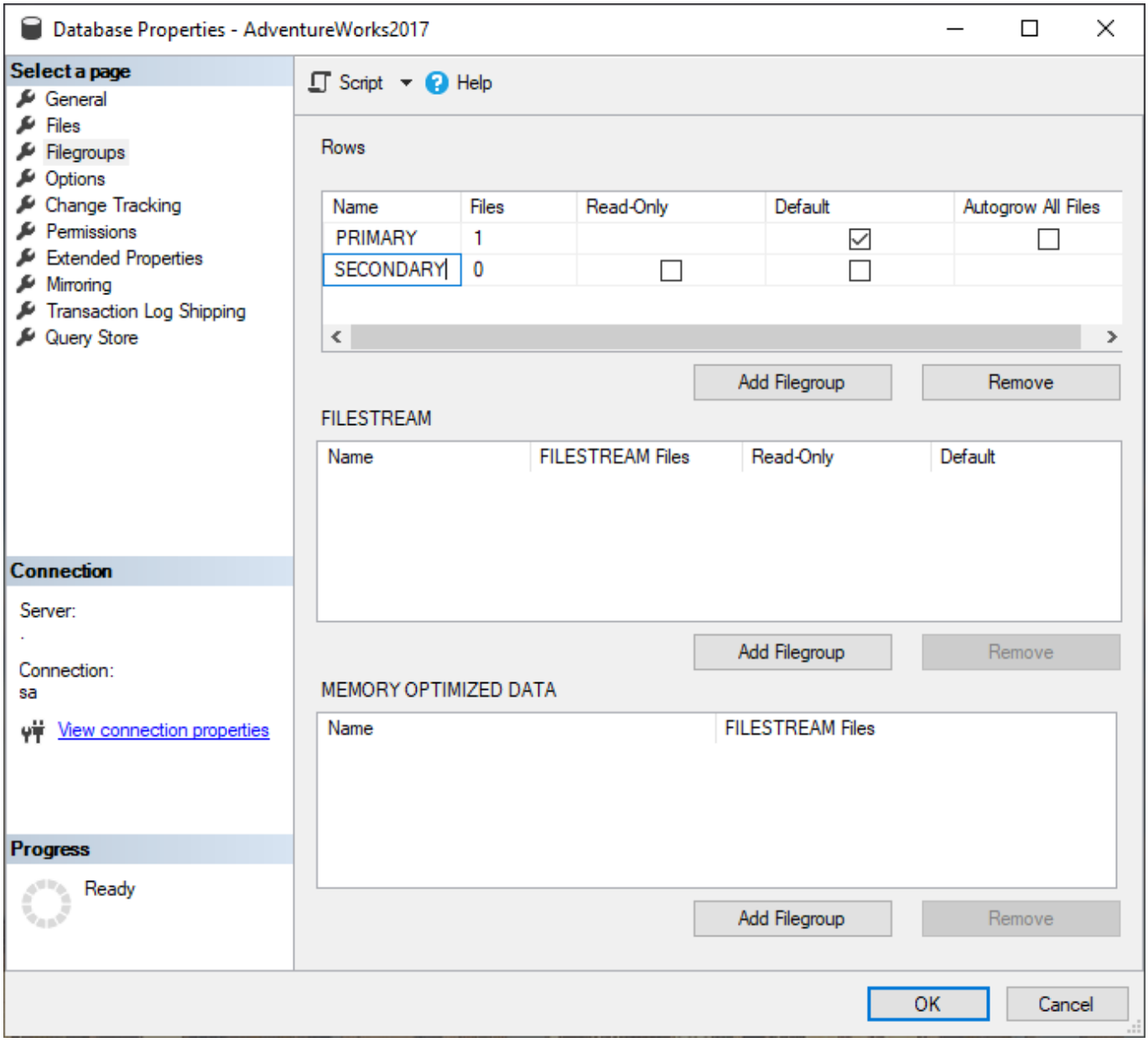
SQL Server Management Studio veya T-SQL ile tablo bölünebilir.

Tablo bölme işlemi genel olarak dört adımda olmaktadır.

- Bölünecek tablonun verilerini tutacak FILEGROUPS ve dosyaların oluşturulması
- Bölünme kurallarını oluşturacak fonksiyonun oluşturulması
- Bölünmüş dosyaların tutulacağı schemanın oluşturulması
- Tablonun oluşturulan fonksiyon ile schema üzerinde bölünmesi

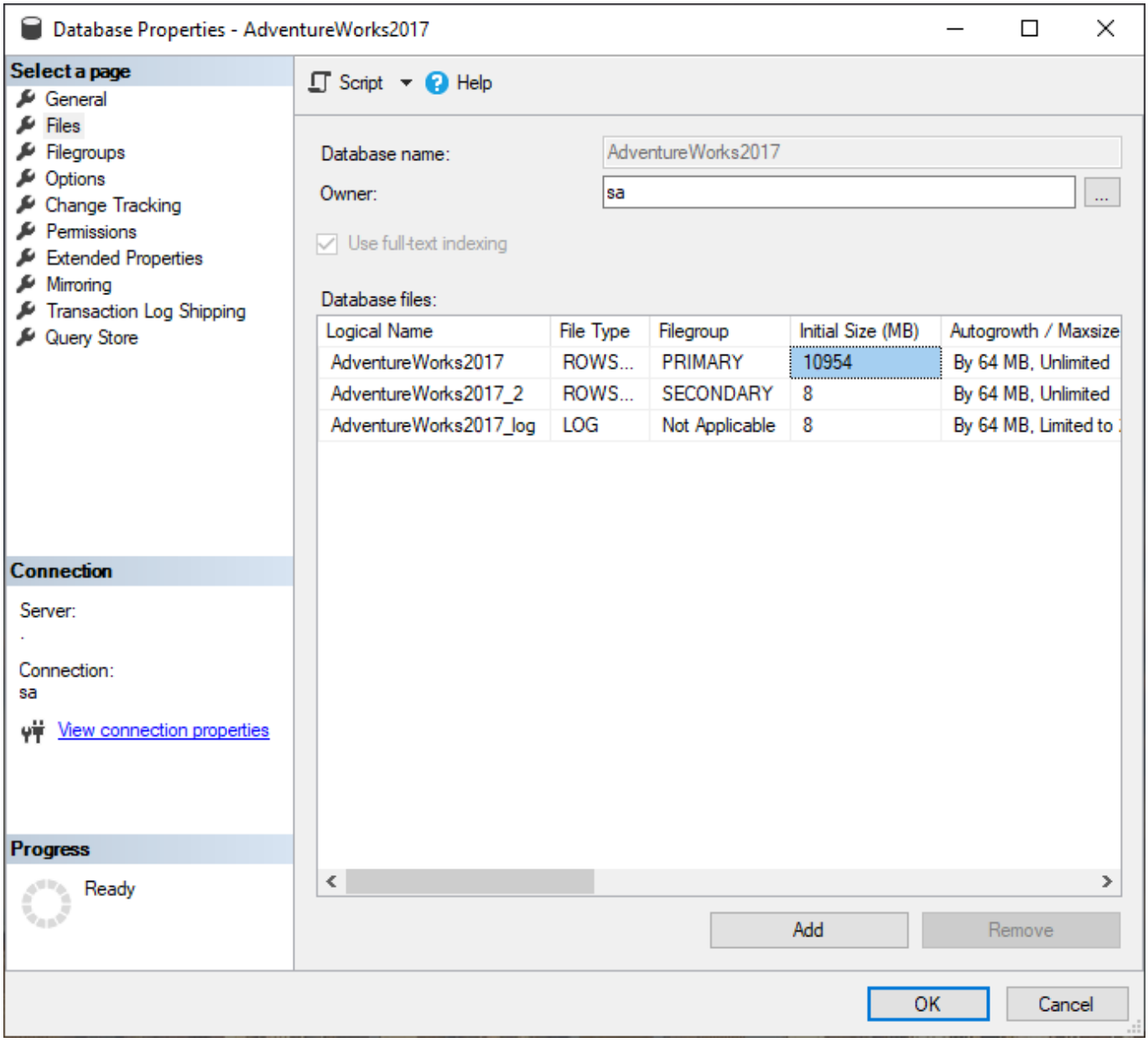
SQL Server Management Studio İle Bölme İşlemi

İlk işlem olarak ilgili veri tabanını sağ tıklayıp özellikler (Properties) ekranını açarak aşağıdaki ekran görüntülerine göre FILEGROUP ve File ekliyoruz.



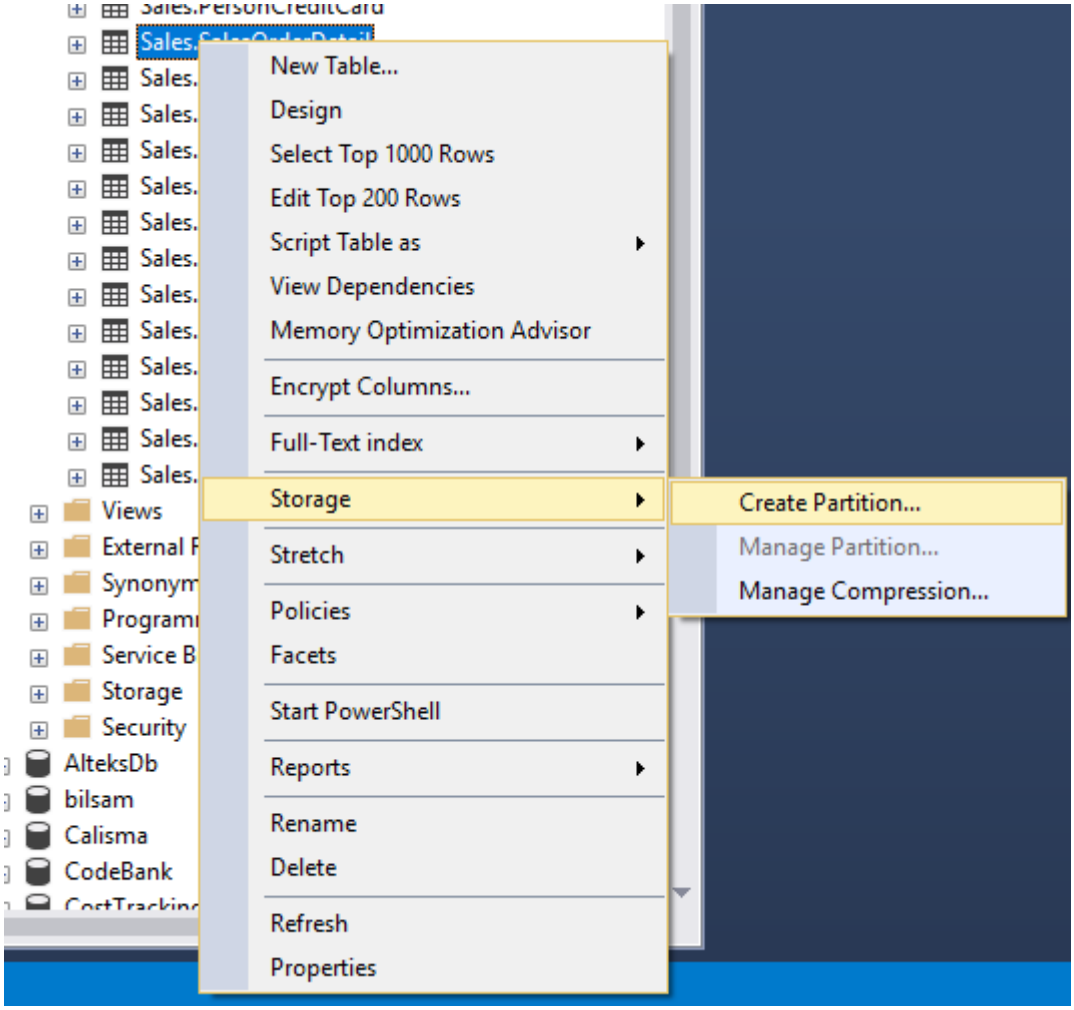
FILEGROUP Ekleme

“Rows” kısmında “Add Filegroup” butonuna tıklayarak bir filegroup ekliyoruz. Filegroup ekledikten sonra bu filegroup içerisinde yer alacak file ekliyoruz.



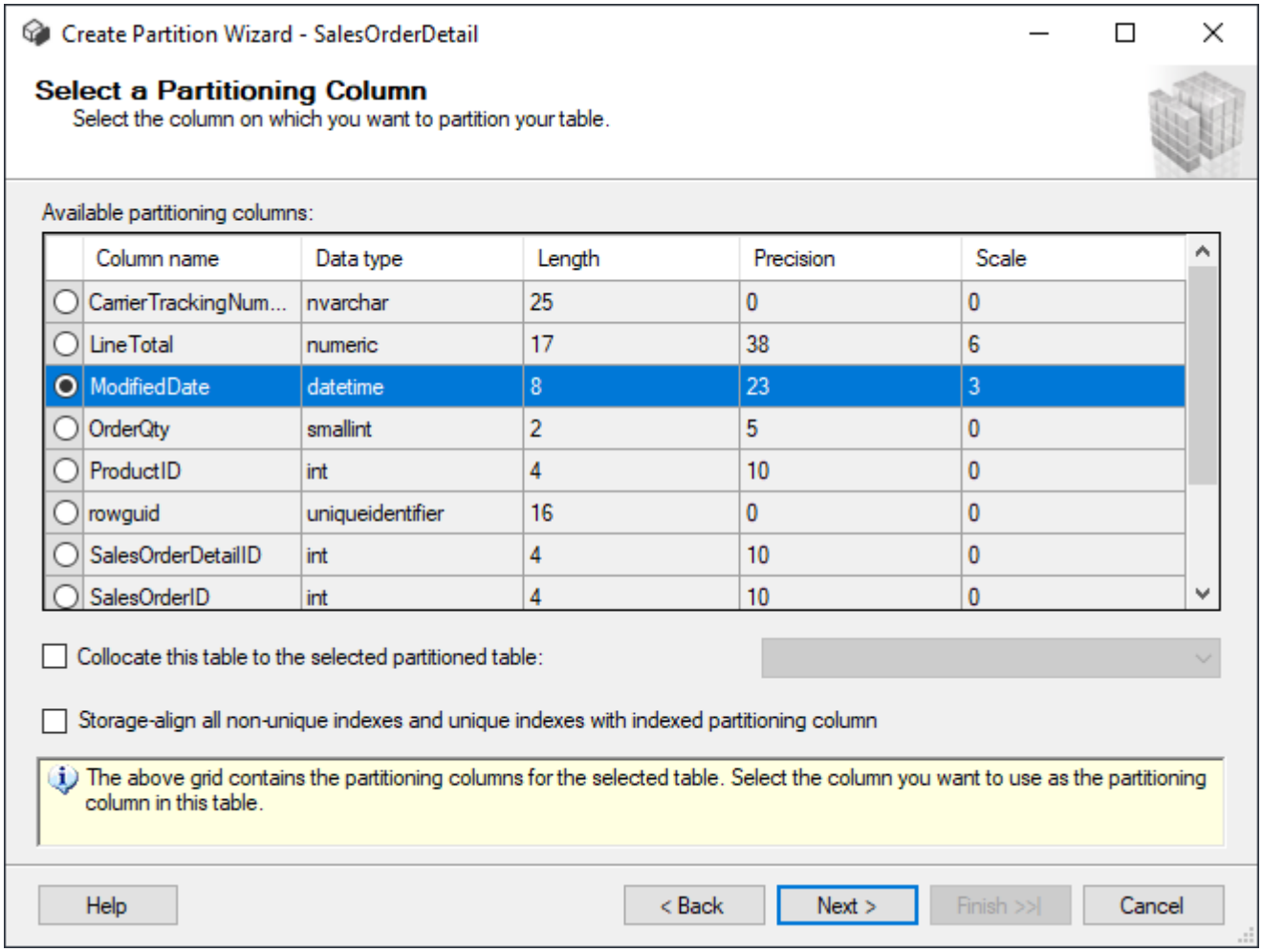
FILE Ekleme

“Add” butonu ile bir önceki adımda oluşturduğumuz filegroup içerisinde yer alacak bir dosya ekliyoruz. Burada dikkat etmemiz gereken nokta dosyanın ndf uzantılı olması gerektiğidir.



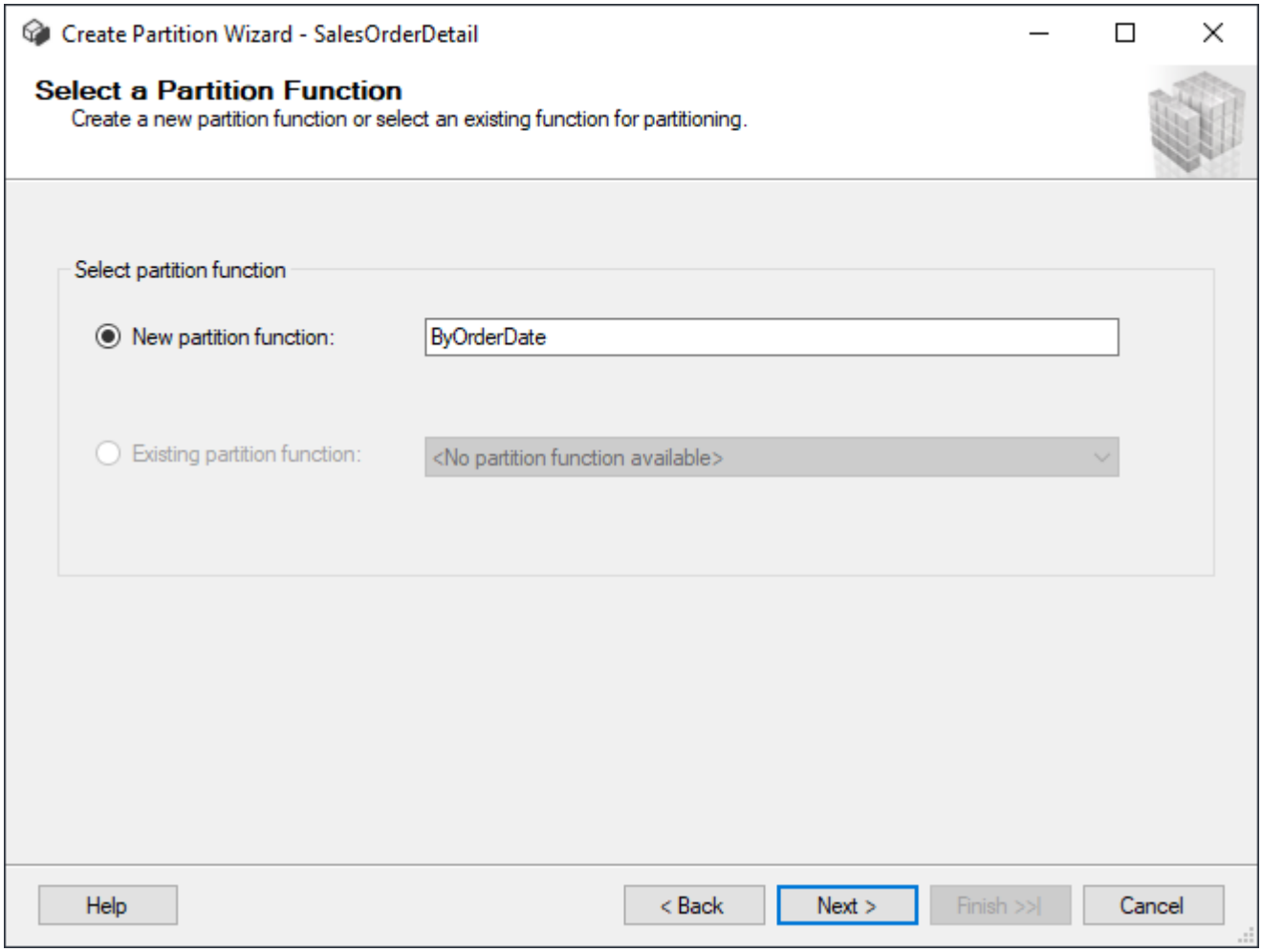
Tablo Bölme İşlemi Başlatma

Filegroup ve file oluşturma işlemlerinden sonra tabloyu bölme işlemine başlıyoruz. Bölünecek tablo üzerine sağ tıklayarak Storage > Create Partition… yolunu izleyerek işleme başlıyoruz.



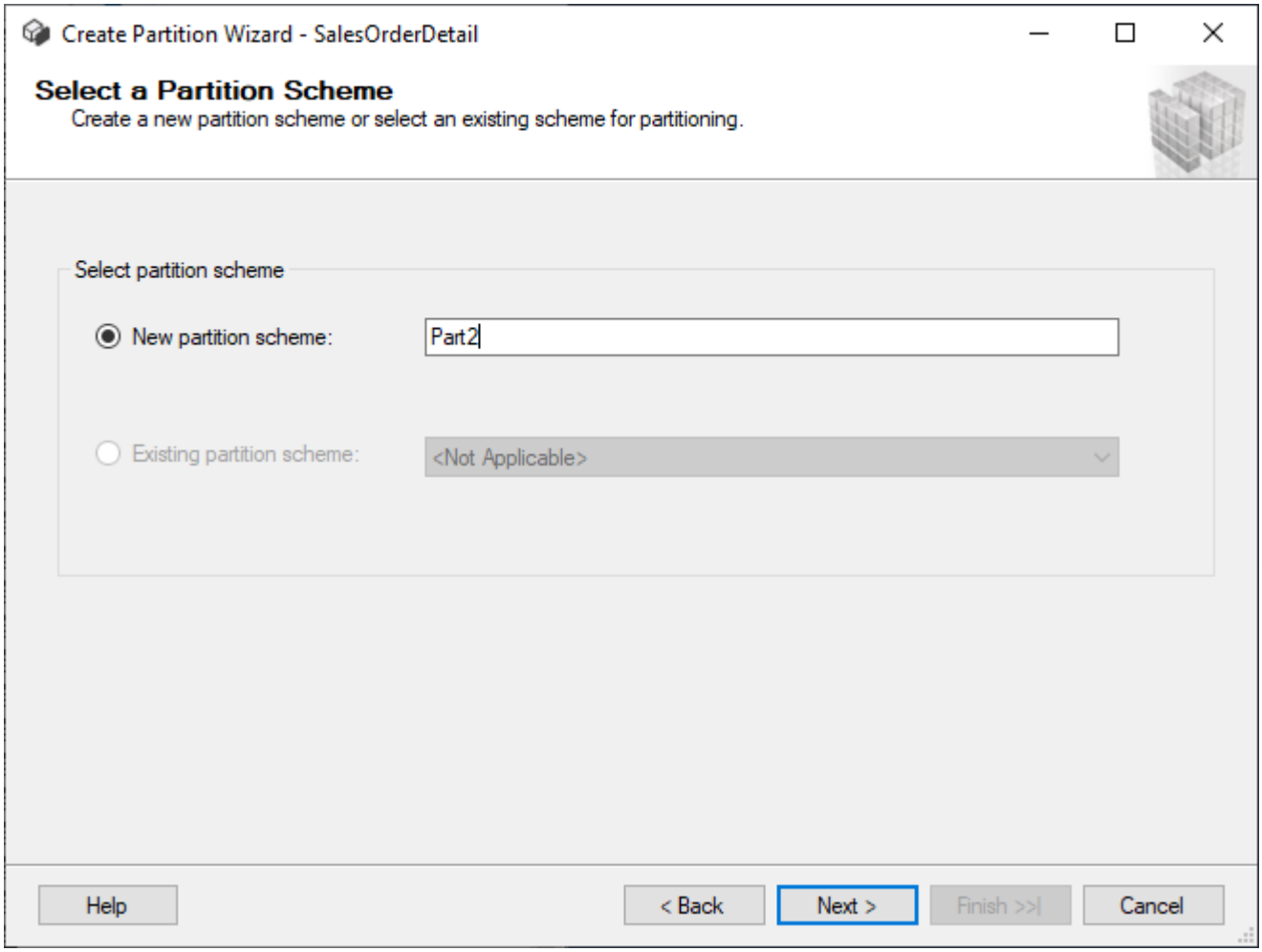
Üzerinde Bölme Yapılacak Kolon Seçimi

Açılan “Select a Partitioning Column” ekranında üzerinde bölme şartının çalışacağı kolonu seçiyoruz. Mantıksal olarak gruplanabilen her hangi bir kolon seçilebilir.



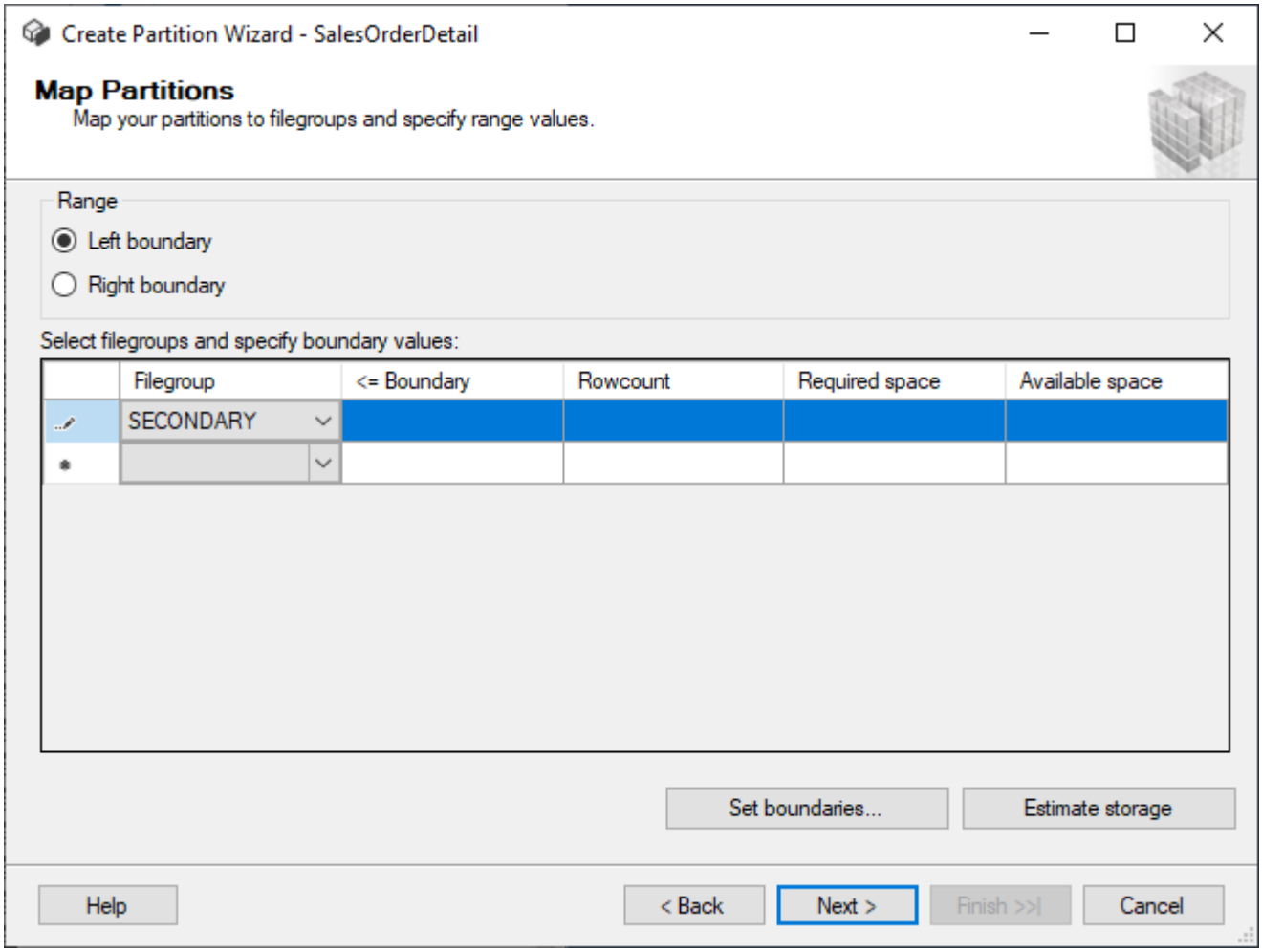
Partition Function

“Select a Partition Function” ekranında bölme kuralını içeren partition fonksiyonunu seçiyoruz. Daha önceden oluşturulan bir fonksiyon var ise “Existing partition function” seçeneği ile seçiyoruz. Oluşturulan fonksiyon yok ise “New partition function” seçeneği ile fonksiyon adını giriyoruz. Sistem girilen isimde fonksiyonu otomatik oluşturacaktır.



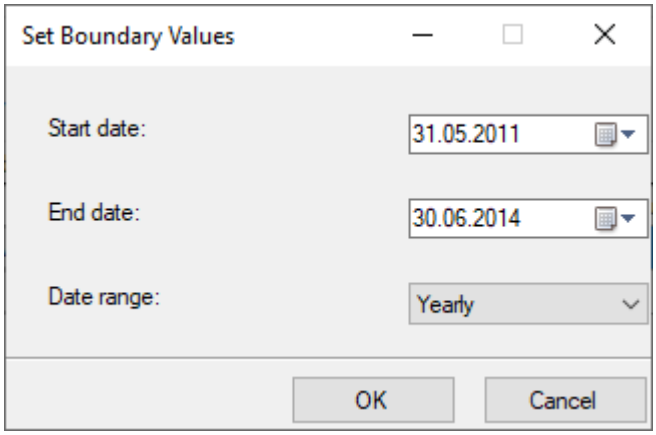
Şema Seçimi

“Select a Partition Scheme” ekranında da bölünme işlemini tutacak schema seçimi yapıyoruz.



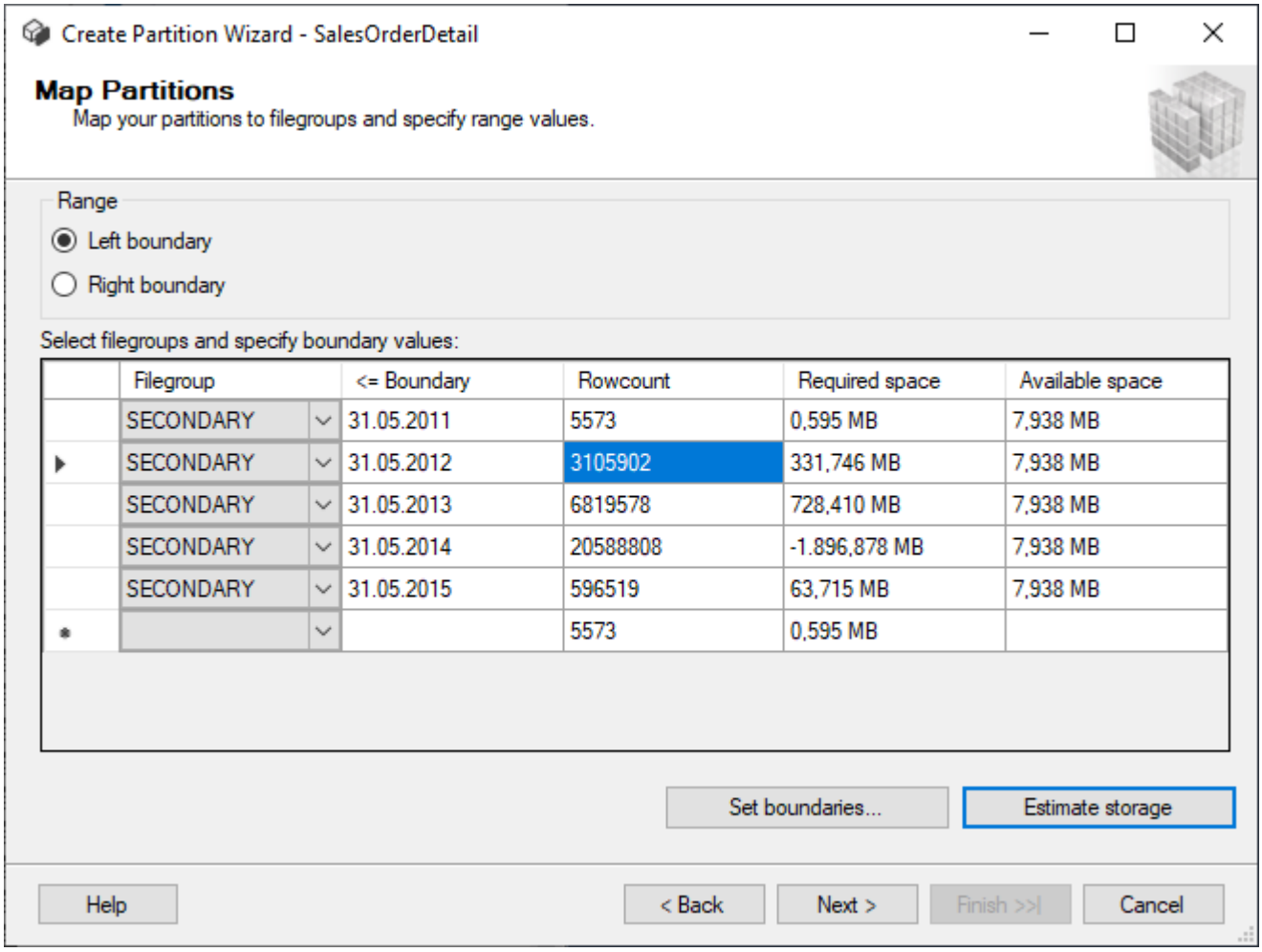
Bölme Kurallarının Belirlenmesi

“Map Partition” ekranında verilerin hangi kurala göre hangi dosyalara bölüneceği kurallarını belirliyoruz. Tarih verisi içeren kolona göre bölme işlemi yapıyor isek “Set Boundaries…” butonuna tıklayarak aralıkları otomatik hesaplatabileceğimiz bir diyalog penceresi açabiliriz.



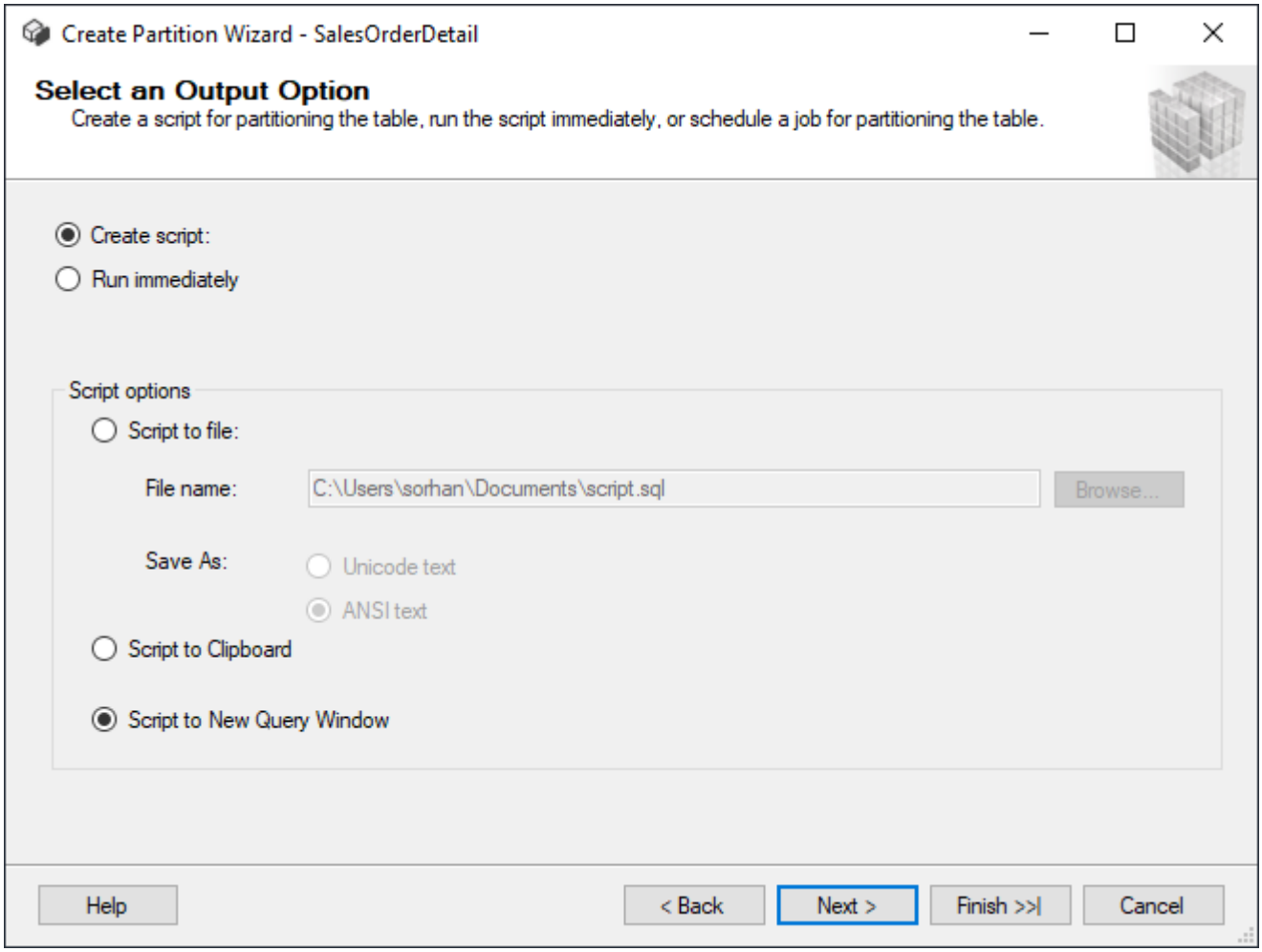
Set Boundaries…

“Set Boundaries…” ekranında başlangıç ve bitiş tarihlerini girdikten soran bölünmenin aralığını seçiyoruz. Örneğimizde verileri yılına göre böleceğimizi seçmişiz mesela.



Set Boundaries… işlemi sonrası

“Estimate storage” butonuna tıklayarak verilerin bölünme sonrası durumlarını ön izleyebilirz.



İşlem Başlatma

Son adım olarak işlemin ne zaman yapılacağı ile ilgili seçimi de yaptıktan sonra tablo bölme işlemi tamamlanmış olacaktır.

T-SQL İle Tablo Bölme

```
USE [AdventureWorks2017]
GO
BEGIN TRANSACTION
CREATE PARTITION FUNCTION [ByOrderDate](datetime) AS RANGE
LEFT FOR VALUES (N'2011-05-31T00:00:00',
N'2012-05-31T00:00:00', N'2013-05-31T00:00:00',
N'2014-05-31T00:00:00', N'2015-05-31T00:00:00')
```

```
CREATE PARTITION SCHEME [Part2] AS PARTITION [ByOrderDate] TO
([SECONDARY], [SECONDARY], [SECONDARY], [SECONDARY],
[SECONDARY], [SECONDARY])

ALTER TABLE [Sales].[SalesOrderDetail] DROP CONSTRAINT
[PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID] WITH (
ONLINE = OFF )


ALTER TABLE [Sales].[SalesOrderDetail] ADD  CONSTRAINT
[PK_SalesOrderDetail_SalesOrderID_SalesOrderDetailID] PRIMARY
KEY NONCLUSTERED
(
        [SalesOrderID] ASC,
        [SalesOrderDetailID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]


CREATE CLUSTERED INDEX
[ClusteredIndex_on_Part2_636815093769025611] ON
[Sales].[SalesOrderDetail]
(
        [ModifiedDate]
)WITH (SORT_IN_TEMPDB = OFF, DROP_EXISTING = OFF, ONLINE =
OFF) ON [Part2]([ModifiedDate])


DROP INDEX [ClusteredIndex_on_Part2_636815093769025611] ON
[Sales].[SalesOrderDetail]

COMMIT TRANSACTION
```

SQL Server Veri Girme (Insert)

SQL Server üzerinde tanımlı tablolara veri girişi INSERT komutu ile olmaktadır. INSERT komutunun kullanım şekli aşağıdaki gibidir:

```
INSERT INTO tablo_adı (kolon1, kolon2, kolon3, ...)
VALUES (değer1, değer2, değer3, ...);
```

- tablo_adı: Giriş yapılacak tablo adı
- (kolon1, kolon2, kolon3, …): Giriş yapılacak kolon isimleridir. Insert işleminde otomatik artan ve NULL kabul eden kolonlar dışında kolonlara giriş yapılması zorunludur.
- VALUES (değer1, değer2, değer3, …): Kolon listesine girilen kolonlara verilecek değerlerdir. Buradaki değerler sırası ile kolon listesindeki kolonlarla eşleştirilir.

Tek Satır Veri Girişi

```
INSERT INTO Musteriler (Ad, Soyad, Telefon, Adres)
VALUES ('Sait', 'ORHAN', '1234567890', 'saitorhan.com')
```

(1 row affected) mesajı işlemin başarılı olduğunu ve bir satır

verinin girildiğini belirtir.

Varsayılan Değer Girme

Insert anında değeri hesaplanıp girilecek kolonlarda tablo tasarlanırken “Default Value or Binding” özelliğine varsayılan değeri girilir. Örneğin Müşteriler tablosunda “Kayıt Tarihi” kolonu kayıt anındaki zamanı alır. Her defasında girmek yerine kolonun varsayılan değerine “GETDATE()” fonksiyonu yazılırsa tarih değerini alıp kolon değerine girer.

```
INSERT INTO Musteriler (Ad, Soyad, Telefon, Adres,
KayitTarihi)
VALUES ('Sait', 'ORHAN', '1234567890', 'saitorhan.com',
DEFAULT)
```

Sorgunun VALUES parametrelerinden DEFAULT değeri kolonun varsayılan değeri olan GETDATE() fonksiyonunu çağırarak değerini KayitTarihi kolonuna atar.

Birden Fazla Kayıt Girmek

Insert işlemi tek satır veri girmeyi sağladığı gibi aynı sorguda birden fazla satır veri girmeyi de destekler. Girilecek değerler virgül (,) ile ayırarak girilebilir.

```
INSERT INTO Musteriler (Ad, Soyad, Telefon, Adres,
KayitTarihi)
VALUES
('Sait', 'ORHAN', '1234567890', 'saitorhan.com', DEFAULT),
('Bilal', 'ORHAN', '1234567890', 'saitorhan.com', DEFAULT),
('Said Nur', 'Yağmahan', '1234567890', 'saitorhan.com',
DEFAULT),
('Suheyb', 'Yağmahan', '1234567890', 'saitorhan.com', DEFAULT)
```

Bu sorgu sonucunda dört satırın girildiğini belirten “(4 rows affected)” mesajı gösterilecektir.

IDENTITY (Otomatik Artan Değer) Kolonuna Değer Girme

Otomatik artan olan kolona veri girildiğinde aşağıdaki gibi hata alırız.

```
Msg 544, Level 16, State 1, Line 3
Cannot insert explicit value for identity column in table
'Musteriler' when IDENTITY_INSERT is set to OFF.
```

Otomatik artan kolonuna değer girebilmek için ilgili tablo için IDENTITY_INSERT değerinin ON değerine alınması gerekmektedir. İşlem bittikten sonra değeri OFF değerine ayarlamayı unutmayınız.

```
SET IDENTITY_INSERT Musteriler ON

INSERT INTO Musteriler (Id,Ad, Soyad, Telefon, Adres,
KayitTarihi)
VALUES
(17,'Sait', 'ORHAN', '1234567890', 'saitorhan.com', DEFAULT),
(18,'Bilal', 'ORHAN', '1234567890', 'saitorhan.com', DEFAULT),
(19,'Said Nur', 'Yağmahan', '1234567890', 'saitorhan.com',
DEFAULT),
(20,'Suheyb', 'Yağmahan', '1234567890', 'saitorhan.com',
DEFAULT)

SET IDENTITY_INSERT Musteriler OFF
```

Sorgunun ilk satırında IDENTITY_INSERT değeri açılmış, son satırında da kapatılmıştır.

SELECT Sonucunun Başka Tabloya INSERT Edilmesi

Çoklu INSERT işlemi elle girilen değerler olabileceği gibi SELECT sorgusunun sonucu da olabilir. INSERT komutunda kolon listesi yazıldıktan sonra sırası eşleşecek şekilde yazılan SELECT sorgusunun sonucu INSERT komutuna iletilir.

```
INSERT INTO Musteriler2 (Ad, Soyad, Telefon, Adres,
KayitTarihi)

SELECT Ad, Soyad, Telefon, Adres, KayitTarihi
```

```
FROM Musteriler
```

Örnek sorguda , Musteriler tablosundan alınan kayıtlar Musteriler2 tablosuna INSERT edilmektedir.

INSERT İle Girilen Değerlerin Gösterilmesi

INSERT komutu ile kullanılacak OUTPUT parametresi ile girilen değerler gösterilebilir. Normal şartlarda INSERT işlemi sonucunda eklenen satır sayısı gösterilirken OUTPUT parametresi ile değerler ekrana gösterilebilir.

```
INSERT INTO Musteriler (Ad, Soyad, Telefon, Adres,
KayitTarihi)
OUTPUT inserted.Id, inserted.Ad, inserted.Soyad,
inserted.Telefon, inserted.Adres, inserted.KayitTarihi
VALUES
('Sait', 'ORHAN', '1234567890', 'saitorhan.com', DEFAULT),
('Bilal', 'ORHAN', '1234567890', 'saitorhan.com', DEFAULT),
('Said Nur', 'Yağmahan', '1234567890', 'saitorhan.com',
DEFAULT),
('Suheyb', 'Yağmahan', '1234567890', 'saitorhan.com', DEFAULT)
```

Sorgunun ikinci satırında gösterilen OUTPUT parametresi kendisine verilen kolonları yeni gelen kayıtlardan seçerek gösterir. “inserted” mevcut sorguda eklenen kayıtları tutan sanal bir

tablodur.

SQL Server Ranking Fonksiyonları

Ranking fonksiyonları sıralama işlevinde satırlara sıra numarası vermek için kullanılan fonksiyonlardır. Kullanılan fonksiyona göre bazı sıra numaraları aynı olabilir.

ROW_NUMBER

Sorgu sonucunu numaralandırır. Kullanılan yönteme göre bütün sorgu sonucunu tek grup olarak algılayıp numaralandırabileceği gibi her grubu kendi içinde tekrar birden başlayarak numaralandırabilir.

Kullanım Şekli

```
ROW_NUMBER ( )
    OVER ( [ PARTITION BY value_expression , ... [ n ] ]
order_by_clause )
```

Örnekler

Basit Örnek

Aşağıdaki kod sistem veri tabanlarını listeler. “ROW_NUMBER() OVER(ORDER BY name ASC) AS Row#” komut parçası veri tabanı isimlerini küçükten büyüğe sıralayarak numarasını verir.

```
SELECT
  ROW_NUMBER() OVER(ORDER BY name ASC) AS Row#,
  name, recovery_model_desc
FROM sys.databases
WHERE database_id < 5;
```

Row#	**name**	**recovery_model_desc**
1	master	SIMPLE
2	model	FULL
3	msdb	SIMPLE
4	tempdb	SIMPLE

“[PARTITION BY value_expression , … [n]]” kısmı kullanıldığında value-expression değerlerine göre sonucu gruplara ayırıp her grubu kendi içerisinde birden başlayarak numaralandırır. Gruplama için seçilen değer değiştiğinde sayaç tekrar bire alınır. Örneğin aşağıdaki sorguda işlemi “recovery_model_desc” kolonuna göre gruplama işlemi yapılmış.

```
SELECT
  ROW_NUMBER() OVER(PARTITION BY recovery_model_desc ORDER BY
name ASC)
    AS Row#,
  name, recovery_model_desc
FROM sys.databases WHERE database_id < 5;
```

Row#	**name**	**recovery_model_desc**
1	model	FULL
1	master	SIMPLE
2	msdb	SIMPLE
3	tempdb	SIMPLE

Sonuçta da gördüğümüz gibi recovery_model_desc kolonu “FULL” değeri için 1 verilmişken, “SIMPLE” değeri de kendi içinde numaralandırılmıştır.

Satış Temsilcilerinin Satış Miktarlarının Alınması

Başka bir örnek olarak aşağıdaki sorgu, satış temsilcilerinin yıla göre satış miktarını sorguluyor. Sorguyu yaparken de “ROW_NUMBER() OVER(ORDER BY SalesYTD DESC)” kısmı ile satış miktarlarını azalan sırada sıralayıp numaralandırmaktadır.

```
USE AdventureWorks2012;
```

```
GO
SELECT ROW_NUMBER() OVER(ORDER BY SalesYTD DESC) AS Row,
    FirstName, LastName, ROUND(SalesYTD,2,1) AS "Sales YTD"
FROM Sales.vSalesPerson
WHERE TerritoryName IS NOT NULL AND SalesYTD <> 0;
```

Row	FirstName	LastName	Sales YTD
1	Linda	Mitchell	4251368,54
2	Jae	Pak	4116871,22
3	Michael	Blythe	3763178,17
4	Jillian	Carson	3189418,36
5	Ranjit	Varkey Chudukatil	3121616,32
6	José	Saraiva	2604540,71
7	Shu	Ito	2458535,61
8	Tsvi	Reiter	2315185,61
9	Rachel	Valdez	1827066,71
10	Tete	Mensa-Annan	1576562,19
11	David	Campbell	1573012,93
12	Garrett	Vargas	1453719,46
13	Lynn	Tsoflias	1421810,92
14	Pamela	Ansman-Wolfe	1352577,13

ROW_NUMBER

ROW_NUMBER’IN PARTITION İLE KULLANILMASI

Yukarıdaki sorguyu bu sefer satış temsilcilerini bölgelerine göre gruplayıp kendi grupları içerisinde sıralamak için değiştirirsek;

```
USE AdventureWorks2012;
GO
SELECT FirstName, LastName, TerritoryName, ROUND(SalesYTD,2,1)
AS SalesYTD,
ROW_NUMBER() OVER(PARTITION BY TerritoryName ORDER BY SalesYTD
DESC)
  AS Row
```

```
 FROM Sales.vSalesPerson
 WHERE TerritoryName IS NOT NULL AND SalesYTD <> 0
 ORDER BY TerritoryName;
```

FirstName	LastName	TerritoryName	SalesYTD	Row
Lynn	Tsoflias	Australia	1421810,92	1
José	Saraiva	Canada	2604540,71	1
Garrett	Vargas	Canada	1453719,46	2
Jillian	Carson	Central	3189418,36	1
Ranjit	Varkey Chudukatil	France	3121616,32	1
Rachel	Valdez	Germany	1827066,71	1
Michael	Blythe	Northeast	3763178,17	1
Tete	Mensa-Annan	Northwest	1576562,19	1
David	Campbell	Northwest	1573012,93	2
Pamela	Ansman-Wolfe	Northwest	1352577,13	3
Tsvi	Reiter	Southeast	2315185,61	1
Linda	Mitchell	Southwest	4251368,54	1
Shu	Ito	Southwest	2458535,61	2
Jae	Pak	United Kingdom	4116871,22	1

ROW_NUMBER PARTITION

Sonuç ekranında da göründüğü gibi bölge adı (TerritoryName) değiştikçe sayaç bire eşitleniyor.

RANK

Çalışması ve kullanımı ROW_NUMBER gibidir. Farklı olarak ROW_NUMBER her bir satırı artan sırada numaralandırırken (1,2,3,4,5) RANK eşit değere sahip satırları aynı numara ile numaralandırmaktadır (1,2,2,3,4) dolayısı ile aynı değerleri dönderebilir.

Kullanım Şekli

```
RANK ( ) OVER ( [ partition_by_clause ] order_by_clause )
```

Eğer bir değer tekrar ederse sonra ki değer tekrar edenlerin sayısı toplamı kadar olacaktır. Örneğin 1, 2, 2, 2, 4, 5 vb. İki değerinden sonra 4 değerinin gelmesinin nedeni 4 değerinin gerçekten dördüncü sırada olmasıdır ve kendisinden önce bir tane birinci ve üç tana ikinci eleman bulunmaktadır. Bu özelliğinin sonucu olarak RANK fonksiyonu ardışık numaralar döndürmeyebilir.

Aşağıdaki örnek, maaşlarına göre sıralanmış ilk on çalışanı döndürmektedir. PARTITION BY deyimi belirtilmediğinden, RANK işlevi sonuç kümesindeki tüm satırlara uygulandı.

```
USE AdventureWorks2012
SELECT TOP(10) BusinessEntityID, Rate,
       RANK() OVER (ORDER BY Rate DESC) AS RankBySalary
FROM HumanResources.EmployeePayHistory AS eph1
WHERE RateChangeDate = (SELECT MAX(RateChangeDate)
                        FROM HumanResources.EmployeePayHistory
AS eph2
                        WHERE eph1.BusinessEntityID =
eph2.BusinessEntityID)
ORDER BY BusinessEntityID;
```

BusinessEntityID	Rate	RankBySalary
1	125,50	1
2	63,4615	4
3	43,2692	11
4	29,8462	28
5	32,6923	22
6	32,6923	22
7	50,4808	6
8	40,8654	14
9	40,8654	14
10	42,4808	13

RANK

Aşağıdaki örnek sorguda da ürünleri stok durumlarına ve konumlarına göre sıralar. Ürünleri LocationID değerine göre gruplayıp Quantity değerine göre azalan sırada listeler. Listenin ilk iki ürünü olan 494 ve 495 numaralı ürünlerin aynı RANK değerine sahip olduğuna dikkat edin. Sonra gelen değer tekrar eden iki 1'den dolayı üç olmuştur. Bir diğer nokta PARTITION BY ile kullanılan LocationID değeri değiştiğinde sayacın bire eşitlendiğine dikkat edin.

```
USE AdventureWorks2012;
GO
SELECT i.ProductID, p.Name, i.LocationID, i.Quantity
    ,RANK() OVER
    (PARTITION BY i.LocationID ORDER BY i.Quantity DESC) AS
Rank
FROM Production.ProductInventory AS i
INNER JOIN Production.Product AS p
    ON i.ProductID = p.ProductID
WHERE i.LocationID BETWEEN 3 AND 4
ORDER BY i.LocationID;
GO
```

ProductID	Name	LocationID	Quantity	Rank
494	Paint - Silver	3	49	1
495	Paint - Blue	3	49	1
493	Paint - Red	3	41	3
496	Paint - Yellow	3	30	4
492	Paint - Black	3	17	5
495	Paint - Blue	4	35	1
496	Paint - Yellow	4	25	2
493	Paint - Red	4	24	3
492	Paint - Black	4	14	4
494	Paint - Silver	4	12	5

RANK PARTITION

DENSE_RANK

Kullanımı RANK ile birebir aynıdır. RANK fonksiyonundan farkı, RANK fonksiyonunda tekrar eden değerden sonra kayıtların sayısı kadar olan değer geliyorken DENSE_RANK fonksiyonunda tekrar eden değerlerden sonra ardışık değer gelmektedir. Yukarıdaki sorgunun DENSE_RANK ile yazılmış hali:

```
USE AdventureWorks2012;
GO
SELECT i.ProductID, p.Name, i.LocationID, i.Quantity
    ,DENSE_RANK() OVER
    (PARTITION BY i.LocationID ORDER BY i.Quantity DESC) AS
Rank
FROM Production.ProductInventory AS i
INNER JOIN Production.Product AS p
    ON i.ProductID = p.ProductID
WHERE i.LocationID BETWEEN 3 AND 4
ORDER BY i.LocationID;
GO
```

ProductID	Name	LocationID	Quantity	Rank
494	Paint - Silver	3	49	1
495	Paint - Blue	3	49	1
493	Paint - Red	3	41	2
496	Paint - Yellow	3	30	3
492	Paint - Black	3	17	4
495	Paint - Blue	4	35	1
496	Paint - Yellow	4	25	2
493	Paint - Red	4	24	3
492	Paint - Black	4	14	4
494	Paint - Silver	4	12	5

DENSE_RANK

SQL Server Tabloların Özetlenmesi

Veri tabanı sistemlerin sağladığı özelliklerden biri de tabloların özet bilgi halinde sunulabilmesi imkanıdır. Bu özet bilgi tekniklerinden biri de kayıtların belli kriterlere göre gruplanabilmesidir. SQL Server sisteminde gruplama işlemi **GROUP BY** komutu ile yapılabilmektedir. Gruplama işlemi sonucunda her grup için bir satır veri dönecektir. SQL Server üzerinde yapılabilecek gruplama işlemi türlerini incelemeye başlayalım.

Gruplama İşlemi Yazım Şekli

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

SELECT yazılan kolonların GROUP BY kısmında geçmesi gerekmektedir. GROUP BY ile gruplanmayan ve herhangi bir gruplama fonksiyonu (SUM, AVG vb.) ile kullanılmayan kolonlar SELECT içerisinde kullanılamazlar.

GROUP BY

Aşağıdaki sql sorgusu “Satışlar” tablosunu şehir ve ilçeye göre gruplayarak satış toplamlarını vermektedir.

```
select Sehir, Ilce, sum(Tutar) [Toplam Satış]
from Satislar
group by Sehir, Ilce
```

Sehir	Ilce	Toplam Satış
Batman	Hasankeyf	8880,00
Batman	Merkez	19988,00
İstanbul	Merkez	18203,00
İstanbul	Zeytinburnu	9355,00

GROUP BY

GROUP BY ROLLUP

Gruplama işlemi için verilen kolon listesini her adımda bir azaltarak ara toplamı verir. Bu işlem için sorguyu her defasında sondan bir kolonu NULL değeri ile değiştirerek sonucu hesaplar. Örneğin GROUP BY ROLLUP(a,b,c,d) şeklinde verilen bir gruplama sorgusunu sonuçlarını aşağıdaki sonuçları verecek şekilde oluşturur.

- a, b, c, d
- a, b, c, NULL
- a, b, NULL, NULL
- a, NULL, NULL, NULL
- NULL, NULL, NULL, NULL

Örnek bir ROLLUP sorgusu

```
select Sehir, Ilce, sum(Tutar) [Toplam Satış]
from Satislar
group by rollup (Sehir, Ilce)
--(a,b,c,d) =>
--(a,b,c,d),
--(a,b,c, NULL),
--(a, b, NULL, NULL),
--(a, NULL, NULL, NULL),
--(NULL, NULL, NULL, NULL)
```

Sehir	Ilce	Toplam Satış
Batman	Hasankeyf	8880,00
Batman	Merkez	19988,00
Batman	NULL	28868,00
İstanbul	Merkez	18203,00
İstanbul	Zeytinburnu	9355,00
İstanbul	NULL	27558,00
NULL	NULL	56426,00

GROUP BY ROLLUP

GROUP BY CUBE

GROUP BY ROLLUP mantığında çalışır ancak farkı olarak mümkün olan bütün kombinasyonlar için ara toplam hesabı yapar. Örneğin GROUP BY CUBE(a, b) sorgusu için aşağıdaki kombinasyonların ara toplam hesaplamalarını yapar.

- a,b
- a, NULL
- NULL, b
- NULL, NULL

Örnek bir GROUP BY CUBE sorgusu

```
select Sehir, Ilce, sum(Tutar)  [Toplam Satış]
from Satislar
group by cube (Sehir, Ilce)
-- (a,b) =>
--(a,b),
--(NULL, b),
--(a, NULL),
```

```
--(NULL, NULL)
```

Sehir	Ilce	Toplam Satış
Batman	Hasankeyf	8880,00
NULL	Hasankeyf	8880,00
Batman	Merkez	19988,00
İstanbul	Merkez	18203,00
NULL	Merkez	38191,00
İstanbul	Zeytinburnu	9355,00
NULL	Zeytinburnu	9355,00
NULL	NULL	56426,00
Batman	NULL	28868,00
İstanbul	NULL	27558,00

GROUP BY CUBE

GROUP BY GROUPING SETS

Bazı durumlarda sorguyu birden fazla şarta göre gruplamak gerekiyor. group by GROUPING sets sorgusu parametre olarak aldığı gruplama şartlarını union all ile birleştirerek tek sonuç kümesi olarak döner.

Örneğin aşağıda yazılan sql sorgusu yukarıda açıkladığımız rollup ve cube sorgularını birleştirerek sonuç dönüyor.

```
select Sehir, Ilce, SUM(Tutar)  [Toplam Satış]
from Satislar
group by GROUPING sets(ROLLUP(Sehir, Ilce), cube(Sehir, Ilce))
--rollup ve cube işlemlerini union all ile birleştirir
```

Sehir	Ilce	Toplam Satış
Batman	Hasankeyf	8880,00
NULL	Hasankeyf	8880,00
Batman	Merkez	19988,00
İstanbul	Merkez	18203,00
NULL	Merkez	38191,00
İstanbul	Zeytinburnu	9355,00
NULL	Zeytinburnu	9355,00
NULL	NULL	56426,00
Batman	Hasankeyf	8880,00
Batman	Merkez	19988,00
Batman	NULL	28868,00
İstanbul	Merkez	18203,00
İstanbul	Zeytinburnu	9355,00
İstanbul	NULL	27558,00
NULL	NULL	56426,00
Batman	NULL	28868,00
İstanbul	NULL	27558,00

GROUP BY GROUPING SETS

GROUP BY ()

GROUP BY GROUPING SETS, parametresine () şeklinde verilen parametre (NULL, NULL, NULL …) şeklinde tablonun genel toplamını verir.

Genel Notlar

SELECT

- AVG, SUM gibi fonksiyonlar select içerisinde kullanıldığında sonuç olarak ilgili gruba ait hesaplamayı döner
- fonksiyon(DISTING kolon) şeklinde kullanılan fonksiyon sadece farklı değerleri dikkate alarak hesaplama yapar. Tekrar eden değerlerden sadece birini alır.

WHERE

WHERE ile verilen şarta uymayan kayıtlar sorguda dikkate alınmayacaktır.

HAVING

Gruplar üzerinde sorgu oluşturmak için having sorgusu kullaılır. Örneğin satış toplamı 10000’den büyük olan değerleri almak için:

```
select Sehir, Ilce, SUM(Tutar)  [Toplam Satış]
from Satislar
group by GROUPING sets(ROLLUP(Sehir, Ilce), cube(Sehir, Ilce))
having sum(Tutar) > 50000
```

NULL Değerler

SQL Server bütün NULL eşit kabul edilip tek grup altında toplanacaktır.

SQL Server Startup Procedures

- SQL Server sunucum başladığında başlangıç zamanını bir tabloya kaydetse ve bana mail gönderse ne iyi olurdu?

Gibi taleplere cevap SQL Server'da yer alan startup stored procedürlerde saklıdır. Bu procedürler SQL Server başladığında otomatik çalıştırılan prosedürlerdir.

Her hangi bir prosedürü startup prosedür olarak ayarlamak için **sp_procoption** sistem prosedürü kullanılır.

```
exec sp_procoption @ProcName = ['stored procedure name'],
@OptionName = 'STARTUP',
@OptionValue = [on|off]
```

sp_procoption parametreleri:

- @ProcName : Otomatik çalışmaya ayarlanacak prosedürün adı
- @OptionName: Sadece 'STARTUP' değerini destekler

- @OptionValue: Otomatik çalışmayı açmak için ‘ON’, kaoatmak için ‘OFF’ değerleri kullanılır.

sp_procoption prosedürü aşağıdaki kısıtlamalara sahiptir;

- sysadmin rolünde bir kullanıcı ile oturum açılması gerekmektedir.
- Sadece standart prosedürler üzerinde çalışmaktadır.
- Otomatik çalıştırılacak prosedür master veri tabanında olmak zorundadır.
- Otomatik çalıştırılacak prosedür giriş veya dönüş parametresi bulunduramaz.

Sunucu başlama zamanlarını tutacak veri tabanı, tablolarının oluşturulması:

```
USE MASTER
GO

CREATE DATABASE SERVER_METRICS
GO

USE SERVER_METRICS
GO

CREATE TABLE DBO.SERVER_STARTUP_LOG
(
LOGID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
START_TIME DATETIME NOT NULL
CONSTRAINT DF_START_TIME DEFAULT GETDATE()
)
```

```
GO
```

Otomatik çalışacak prosedürün tanımlanması:

```
USE MASTER
GO

CREATE PROCEDURE DBO.LOG_SERVER_START
AS
SET NOCOUNT ON
PRINT '*** LOGGING SERVER STARTUP TIME ***'
INSERT INTO SERVER_METRICS.DBO.SERVER_STARTUP_LOG DEFAULT
VALUES
GO
```

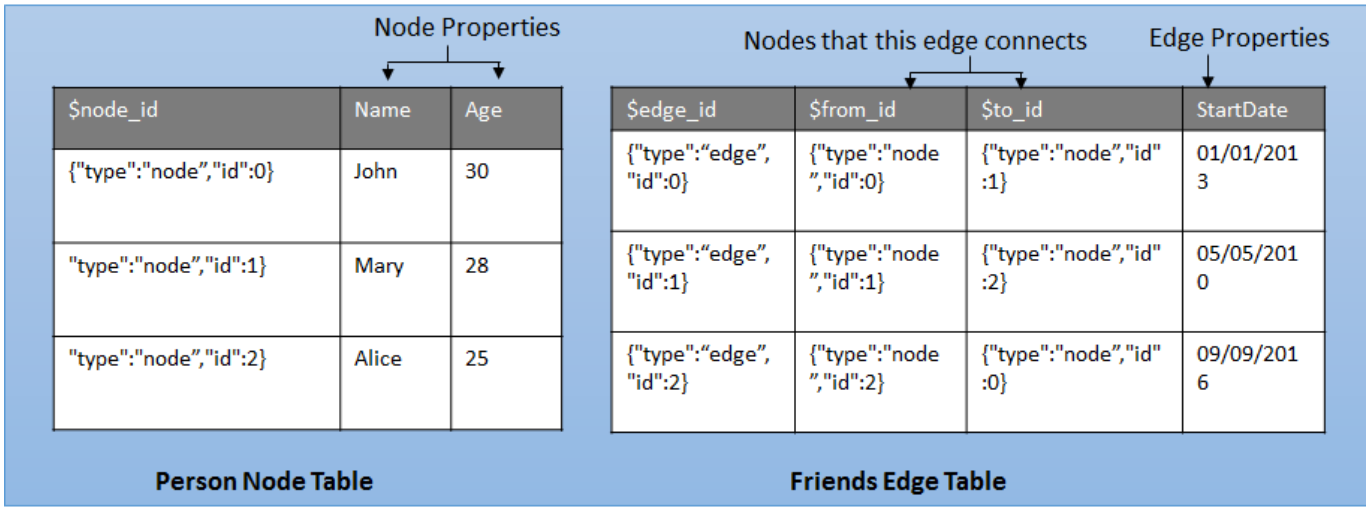
Prosedürün otomatik çalışmaya ayarlanması

```
USE MASTER
GO
EXEC SP_PROCOPTION LOG_SERVER_START, 'STARTUP', 'ON'
GO
```

Prosedürün otomatik çalışmasını engelleme:

```
USE MASTER
GO
EXEC SP_PROCOPTION LOG_SERVER_START, 'STARTUP', 'OFF'
GO
```

SQL Server İle Graf İşleme



SQL Server 2017 versiyonu ile gelen güzel özelliklerden biri de graf işleme özelliğidir. Facebook, Instagram vb. sitelerde yer alan arkadaşlık kayıtları, beğeni kayıtları sistemin arka planında graf yapısı üzerinde tutmaktadır.

SQL Server üzerinde graf yapısı node ve edge olmak üzere iki tablodan oluşmaktadır. Node tablosu, asıl kayıtların tutulduğu tablodur. EDGE tablosu ise bağlantıların tutulduğu tablodur.

Node tablosunda graf yapısında kullanılacak olan ve otomatik atanıp, düzenlenemeyen $node_id isimli bir kolon tutulmaktadır. EDGE tablosuna kayıt girilirken bu kolon değeri kullanılır. EDGE tablosunda da otomatik oluşturulan ve düzenlenemeyen $from_id ve $to_id kolonları bulunmaktadır. Bu kolon değerleri Node tablosundan gelen $node_id değerleridir. Bu bilgilerden sonra örneğimizle devam edelim.

Node Tablolarının Oluşturulması:

```
CREATE TABLE People (
Id int primary key identity,
[Name] nvarchar(100) not null,
) as Node

CREATE TABLE Books (
Id int primary key identity,
[Name] nvarchar(100) not null,
) as Node
```

EDGE Tablolarının Oluşturulması

```
create TABLE Friends (
ContactDate datetime not null,
) as edge

create TABLE Likes (
LikeDate datetime not null,
```

```
) as edge
```

Kitap ve Kişi kayıtlarının girilmesi:

```
insert into People values ('Mustafa Orhan'), ('Muhammed
Yıldız'), ('Bilal Orhan'), ('Said Nur'), ('Hüseyin Varol')
insert into Books values ('SQL Server'),('C#
Yazılım'),('JAVA'),('JavaScript'),('HTML'),('C++')
```

Yeni arkadaş kaydının girilmesi

```
insert into Friends values
        ((select $node_id from People where Id = 7), (select
$node_id from People where Id = 8), '20181121')
```

Bu komutun ilk parametresi EDGE tablosunu anlatırken söz ettiğimiz $from_id ve ikinci parametresi $to_id değerleridir. Son parametre de Friends tablosunu oluştururken girilen kayıt tarihidir.

Yeni Kitap Beğeni kaydının girilmesi

```
insert into Likes values
        ((select $node_id from People where Id = 8), (select
$node_id from Books where Id = 4), '20181121')
```

Sorgu Örnekleri

“Mustafa Orhan” kişisinin beğendiği kitapların listesini alan kod:

```
select Books.Name
from People, Likes, Books
where match (People-(Likes) -> Books)
and People.[Name] = 'Mustafa Orhan'

------------
--SQL Server
--JavaScript
--C++
```

Bu sorguda graf sorgusunu yapan fonksiyonumuz MATCH fonksiyonudur. Yazdığımız bu sorguda graf yapısını şu şekilde çalıştırmasını istedik. People ve Books tablolarını Likes tablosu üzerinde eşleştir.

“Mustafa Orhan” kişisinin arkadaş listesini gösteren kod:

```
select p2.Name
from People p1, Friends f, People p2
where match (p1-(f) -> p2)
and p1.Name = 'Mustafa Orhan'

--------------------
--Muhammed Yıldız
--Bilal Orhan
--Said Nur
--Hüseyin Varol
```

Not:

Bu sorgularda dikkat edilmesi gereken nokta, grafın tek yönlü çalışıyor olmasıdır. Eğer iki taraflı graf çalıştırmak istiyorsanız iki seçeneğiniz bulunmaktadır.

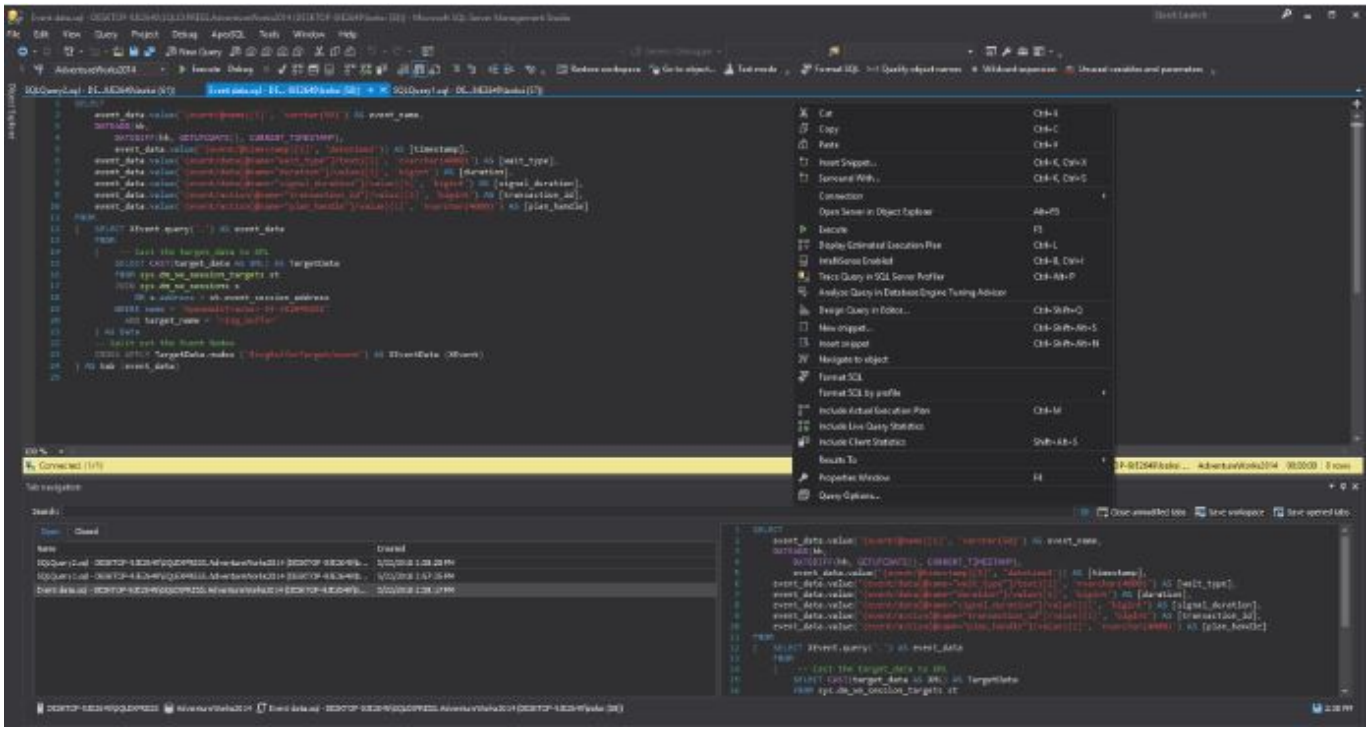
- Girilen her yeni kayıt için, tersi kayıt girmek
- Select sorgusunda MATCH fonksiyonu iki taraflı çalıştırıp iki sorguyu UNION komutu ile birleştirmek. MATCH fonksiyonu OR, AND, NOT gibi operatorlar ile çalışmadığından bu şekilde bir sorgu çalıştırmak gerekmektedir. Örnek sorgu:

```
select p2.Name
from People p1, Friends f1, Friends f2, People p2
where match (p1-(f1) -> p2)
and p1.Name = 'Hüseyin Varol'
union
select p2.Name
from People p1, Friends f, People p2
```

```
where match (p2-(f) -> p1)
and p1.Name = 'Hüseyin Varol'
```

Bu sorgunun ilk parçasında “Hüseyin Varol” kişisinin eklediği arkadaşlar seçiliyorken, ikinci parçasında da “Hüseyin Varol” kişisini ekleyenlerin listesi alınmaktadır. UNION komutu ile bu sorgular tek sonuç seti olarak birleştirilmektedir.

SQL Server Management Studio Koyu Tema Etkinleştirme

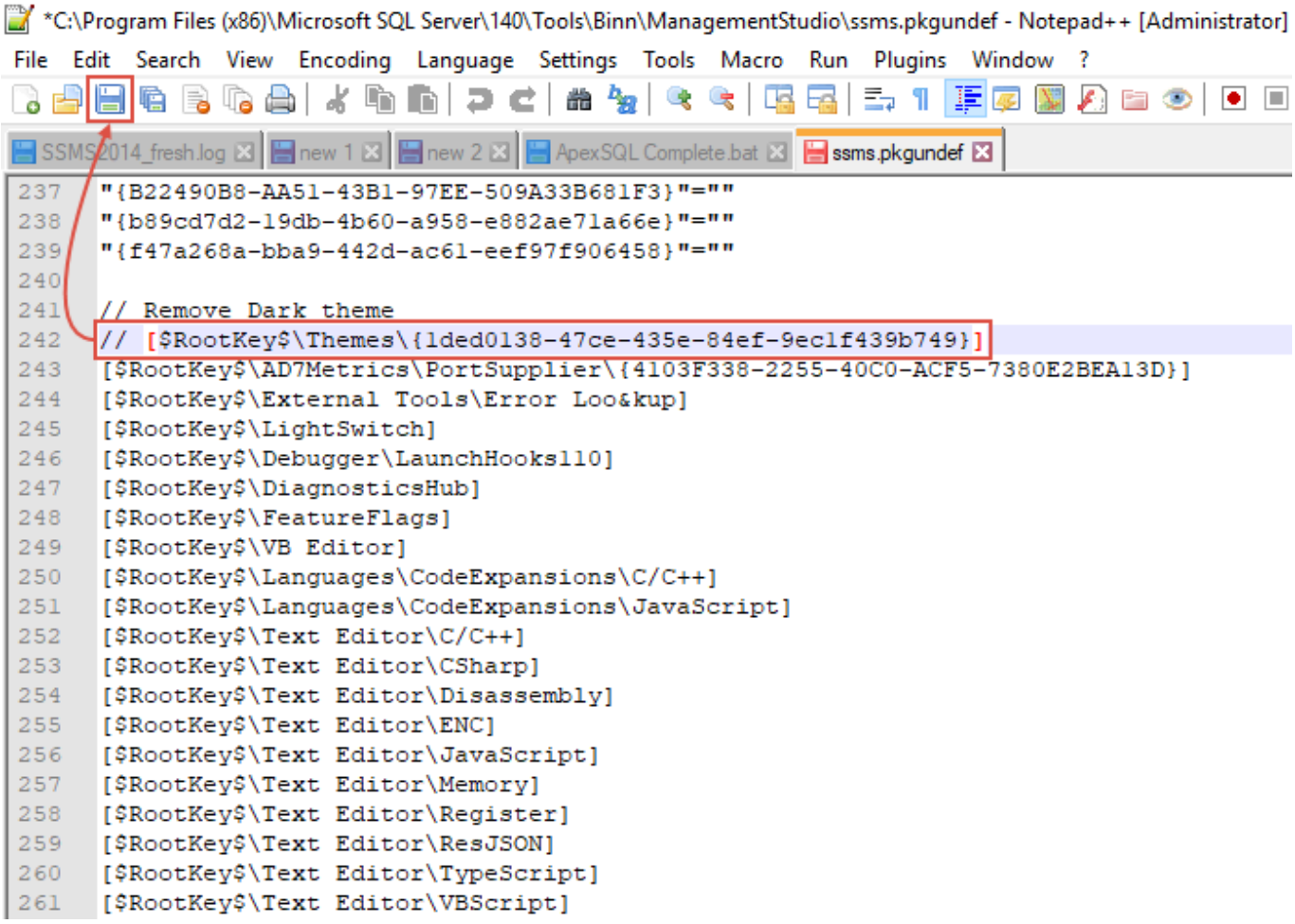


Koyu tema gerek Visual Studio gerek diğer editörlerde hep sevdiğim bir tema olmuştur ancak ne yazık ki SQL Server Management Studio'da maalesef varsayılan olarak pasif gelmektedir. Bu yazımızda SSMS'yi de koyu tema da kullanmayı öğreneceğiz.

Öncelikle Not Defteri veya Notepad++ hangisini kullanıyorsanız sağ tıklayarak yönetici olarak çalıştıralım. Daha sonra kullandığınız SSMS versiyonuna göre aşağıda adresleri verilen dosyayı açalım

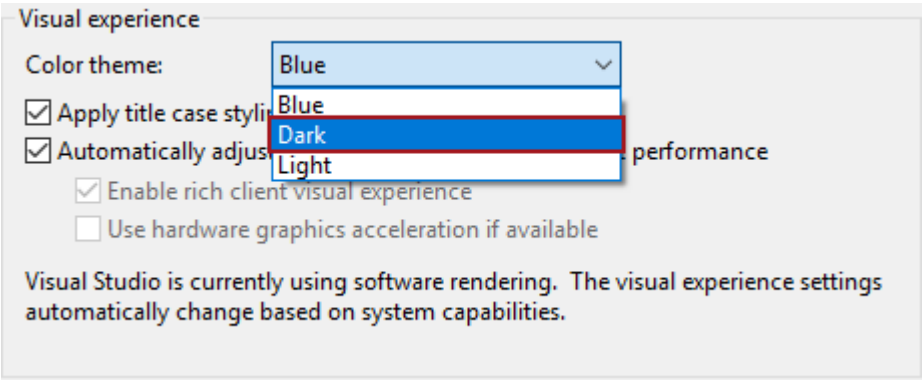
- SSMS 16
    - C:\Program Files (x86)\Microsoft SQL Server\130\Tools\Binn\ManagementStudio\ssms.pkgundef
- SSMS 17
    - C:\Program Files (x86)\Microsoft SQL Server\140\Tools\Binn\ManagementStudio\ssms.pkgundef

Dosyayı açtıktan sonra aşağı doğru arayarak // Remove Dark thema yazılı satırı bulup hemen altındaki satırın başına // işaretini koyarak koyu temayı devre dışı bırakmayı iptal ediyoruz.



Koyu Tema Devre Dışı Bırakma İptal

Değişikliği kaydedip SSMS’yi açtığımızda koyu temanın da seçeneklere geldiğini görürüz.



SSMS Koyu Tema

Her güncellemeden sonra dosya varsayılan ayarına döneceğinden

dolayı SSMS güncellemesinden sonra bu işlemi tekrar edilmesi gerekmektir.

Şartlı INDEX (Filtered Index) Oluşturma

Merhabalar, bu yazımızda daha önceki çalışmalarımda öğrendiğim, üzerinde unique index tanımlanan null değer alabilen kolonda ikinci bir null değer geldiğinde bu kaydın unique indexten dolayı kaydedilmeyeceği, bilgisinin yeni sql server versiyonlarında artık şartlı index yazarak geçersiz olabileceğini ve nasıl yapılacağını öğreneceğiz.

Örneğin bir kişinin kredi kartı numarası olmak zorunda değil (nullable) ama var ise kişiye özel bir numaradır yani unique.

Şartlı INDEX yazmak aslında normal index yazmaktan çok farkı yoktur. Yapılması gereken index ifadesinden sonra WHERE ile koşulların yazılmasıdır. Bu şekilde index sadece şartın sağlandığı durumlarda geçerli olur.

Bu şekilde sadece [Name] kolonu üzerinde sadece [Name] kolonunun null olmadığı zaman çalışacak bir unique index tanımlamış oluruz.

SQL Server Trigger İle Güvenli Alış Veriş

Merhaba arkadaşlar, bu yazımızda sql server üzerinde trigger özelliğini kullanarak güvenli alış veriş sistemini simile etmeye çalışacağız.

NOT: Mail gönderebilmek için Sql Server üzerinde “Management > Database Mail” özelliğinin ayarlanmış ve aktifleştirilmiş olması gerekmektedir.

Senaryomuz şu şekilde olacak:

Sistem ayarlarında izin verilen max güvenli alış veriş miktarı kaydı bulunacak. Diğer bir tabloda da sistemin şüpheli bulduğu ve engellediği marketlerin listesi bulunacak.

Sistemin alış veriş tablosuna bir alış veriş kaydı geldiğinde sipariş tutarı izin verilen max. tutardan fazla ise veya alış veriş yapılan market şüpheli marketler listesinde ise kullanıcıya bir bilgi maili atılacak ve işlem iptal edilecek.

Bunun için gerekli tablolar scripti aşağıdaki gibidir.

Gerekli kontrolleri yapacak olan trigger kodumuz.

Ms SQL Server'ın Gizli Gücü xp_cmdshell

Sql Server dersini alırken sadece kullandığımız belli başlı 3 – 4 komut vardı, SELECT, UPDATE, DELETE… Ama hep dediğimiz bir şey daha vardı "Adamlar bu kadar büyük bir programı sırf SELECT için yazmış olamaz", öyle ya bir yüklüyoruz bilgisayarda neredeyse 10GB yer kaplıyor.

Şimdi Sql Server'ın SELECT dışında yapabildiği binlerce işlemden birini beraber görelim.

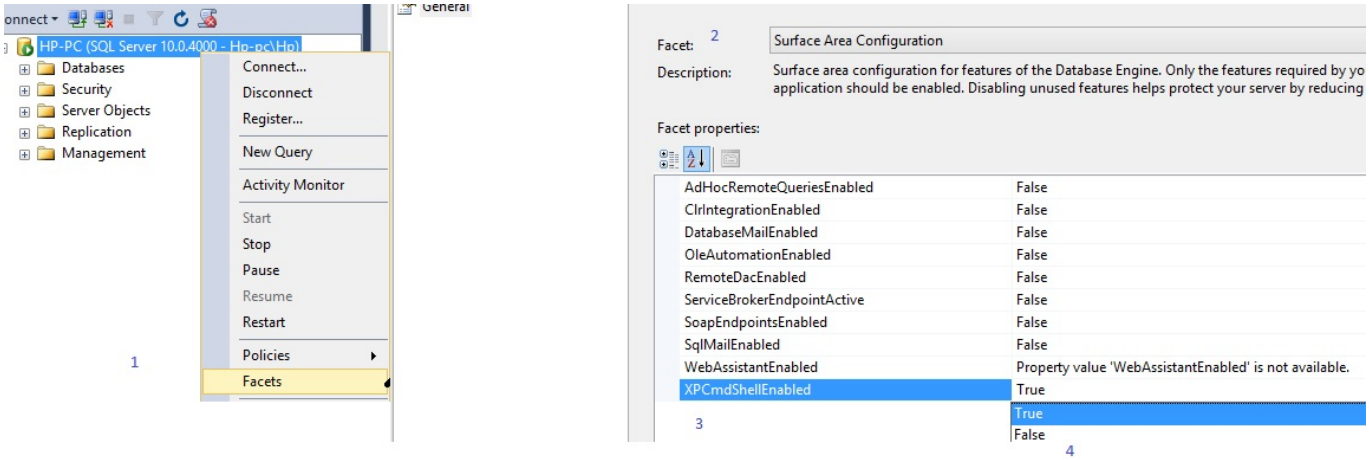
xp_cmdshell, SQL Server üzerinden cmd komut satırına komut göndermeyi ve çalıştırmaya imkan sağlayan stored prosedür. Bu özelliği sayesinde cmd ekranında yapabildiğimiz herşeyi:

- Windows Server üzerinde yeni kullanıcı açıpi buna full yetki verilip kuruluşumuzun her türlü gizli verisi ve işlemlerine ulaşılabilir
- "format X" komutunu göndererek Sql Server üzerinden X diskini formatlanabilir
- Windows Server içerisine her hangi bir .exe dosyası atılarak çalıştırılabilir, bu bir keyloger da olabilir

ve daha fazlası kısaca herşeyi.

xp_cmdshell stored prosedürü varsayılan olarak güvenlik politikası olarak devre dışıdır. Prosedürü devreye almak için aşağıdaki kod çalıştırabilir veya arayüz üzerinde de işlem yapılabilir. Sql koduna ve hemen ardından arayüz adımlarına geçelim.

Alternatif olarak arayüz adımları:



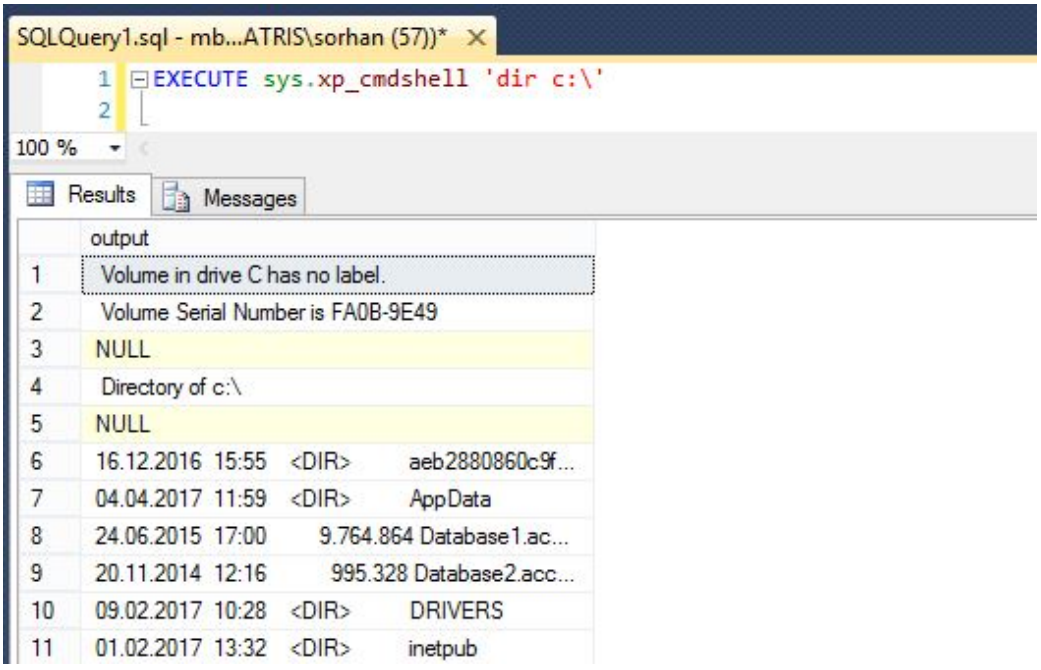
xp_cmdshell aktifleştirme

Bu işlemler sonucunda aktifleştirme gerçekleştirilir.

xp_cmdshell komutu çalıştırıldığında dönen sonuş tek kolon ve text şeklinde olur. Çıktı vermeyen komutlar için başarılı olması durumunda 1, başarızı olması durumunda 0 sonucu döner.

Şimdi de bazı örnekler yaparak çalışma şeklini görelim.

- Çıktı Veren Örnek
  Komut satırına ‘DIR C:\’ komutunu göndererek C:\ diski altında yer alan dosyaları gösterelim.



xp_cmdshell ile C:\ diski altında yer alan dosyaların gösterilmesi

- Çıktı Vermeyen Örnek
  Komut satırın copy komutu göndererek C:\XP\Deneme.txt dosyasını aynı klasör altına Deneme2.txt adıyla kopyalayalım.