# Server Management Objects (SMO) İle SQL Server Kontrolü

Server Management Objects, .Net framework kullanarak SQL Server'ı bütün yönleri ile yönetebileceğiniz yazılımlar geliştirmenizi sağlayan bir kütüphanedir. Örneğin aşağıdaki kod parçası ile sunucu üzerinde kolayca bir veritabanı oluşturuyoruz.

SMO kütüphanesini kullanabilmek için öncelikle Nuget üzerinden Microsoft.SqlServer.SqlManagementObjects projemize referans olarak eklememiz gerekmektedir.

```
SqlConnection sqlConnection = new SqlConnection("Data
Source=localhost;Integrated Security=True");
ServerConnection serverConnection = new
ServerConnection(sqlConnection);
Server server = new Server(serverConnection);

Database database = new Database(server, "Smo");
database.Create();
```

Bu kod ile sınıcı üzerinde Smo adında bir veritabanı oluşur.

Şimdi de başka bir işlem olan tablo oluşturma işlemi yapalım.

```
SqlConnection sqlConnection = new SqlConnection("Data
Source=localhost;Integrated Security=True");
ServerConnection serverConnection = new
ServerConnection(sqlConnection);
Server server = new Server(serverConnection);

Database workDatabase = server.Databases["Smo"];

Table table = new Table(workDatabase, "People");

Column idColumn = new Column(table, "Id", DataType.Int);
idColumn.Identity = true;
table.Columns.Add(idColumn);

Column nameColumn = new Column(table, "Name",
DataType.NVarChar(50));
nameColumn.Nullable = false;
table.Columns.Add(nameColumn);

table.Create();
```

Bu kod ile server nesnesi üzerinde Smo veritabanını seçtikten sonra bu veritabanını parametre olarak verdiğimiz Tablo tipinde bir table nesnesi oluşturuyoruz. Daha sonra bu table nasnesini parametre olarak verdiğimiz ilgili kolonları oluşturup table.Create() ile veritabanını oluşturuyoruz.

Server Management Objects ile diğer işlemleri aşağıdaki videoda inceleyebilirsiniz

SMO İncelemesi

SMO video kodları:

https://github.com/saitorhan/ServerManagementObjects

---

# SQL Server Maintance Plans ve Operatore Plan Sonucunda Mail Gönderme

---

# Veri Tabanı Tablosundan Entity Class Oluşturma

Veri tabanı işlemlerinin gerçekleştirildiği yazılımlarda tabloların entity C# sınıfları gerekebilmektedir. Her ne kadar Visual Studio içerisinde barındırdığı araçlar ile bunu otomatik

yapıyorsa da T-SQL kodları kullanılarak da ilgili sınıf SQL Server tarafında oluşturulup kopyala – yapıştır ile projeye eklenebilir.

Aşağıdaki T-SQL kodu içerisinde ilk satırdaki ‘TableName’ yerine tırnaklar içerisine sınıf karşılığı oluşturulacak tablo yazılıp çalıştırıldığında print komutu ile oluşturulan sınıf ekrana gelecektir.

```
declare @TableName sysname = 'TableName'

declare @Result varchar(max) = 'public class ' + @TableName +
'

{'

select @Result = @Result + '

    public ' + ColumnType + NullableSign + ' ' + ColumnName +
' { get; set; }

'

from

(

    select

        replace(col.name, ' ', '_') ColumnName,

        column_id ColumnId,

        case typ.name

            when 'bigint' then 'long'

            when 'binary' then 'byte[]'
```

```
            when 'bit' then 'bool'

            when 'char' then 'string'

            when 'date' then 'DateTime'

            when 'datetime' then 'DateTime'

            when 'datetime2' then 'DateTime'

            when 'datetimeoffset' then 'DateTimeOffset'

            when 'decimal' then 'decimal'

            when 'float' then 'float'

            when 'image' then 'byte[]'

            when 'int' then 'int'

            when 'money' then 'decimal'

            when 'nchar' then 'char'

            when 'ntext' then 'string'

            when 'numeric' then 'decimal'

            when 'nvarchar' then 'string'

            when 'real' then 'double'

            when 'smalldatetime' then 'DateTime'

            when 'smallint' then 'short'

            when 'smallmoney' then 'decimal'

            when 'text' then 'string'

            when 'time' then 'TimeSpan'

            when 'timestamp' then 'DateTime'
```

```
            when 'tinyint' then 'byte'

            when 'uniqueidentifier' then 'Guid'

            when 'varbinary' then 'byte[]'

            when 'varchar' then 'string'

            else 'UNKNOWN_' + typ.name

        end ColumnType,

        case

            when col.is_nullable = 1 and typ.name in
('bigint', 'bit', 'date', 'datetime', 'datetime2',
'datetimeoffset', 'decimal', 'float', 'int', 'money',
'numeric', 'real', 'smalldatetime', 'smallint', 'smallmoney',
'time', 'tinyint', 'uniqueidentifier')

            then '?'

            else ''

        end NullableSign

    from sys.columns col

        join sys.types typ on

            col.system_type_id = typ.system_type_id AND
col.user_type_id = typ.user_type_id

    where object_id = object_id(@TableName)

) t

order by ColumnId

set @Result = @Result  + '

}'
```

```
print @Result
```

---

# Veri Tabanındaki Bütün Kolonlar

Veri tabanı yöneticilerinin görmek istedikleri konulardan biri de veri tabanı içinde yer alan kolonların listesidir. Aşağıdaki Sql kodu ile MS Sql server içerisinde veri tabanında yer alan bütün kolonları görebilirsiniz.

```
SELECT t.name AS table_name,
SCHEMA_NAME(schema_id) AS schema_name,
c.name AS column_name
FROM sys.tables AS t
INNER JOIN sys.columns c ON t.OBJECT_ID = c.OBJECT_ID
ORDER BY schema_name, table_name;
```

# 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_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;
```


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;
```

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
```


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
```

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
```

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)
```

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)
```

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
```

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 Birden Fazla Tabloyu Beraber Sorgulama

İlişkisel veri tabanı sistemlerinin (RDMS) hedeflerinden biri de veri tekrarını azaltmak ve performansı üst düzeyde tutmaktır. Bu sebeple tekrar eden veriler tablolara ayrılarak ilgili yerlere verilerin refensı verilir. Örneğin bir sipariş tablosunda veriler temel olarak aşağıdaki iki şekilde tutulabilir.

- Her sipariş satırına ürünün bilgileri ve müşteri bilgilerini girmek
- Müşteri bilgilerini ve ürün bilgilerini kendilerine has tablolarda tutup sipariş tablosuna sadece ilgili müşteri ve ürünün referans kolonlarını vermek.

İlk seçenekte oluşacak örnek bir sipariş tablosu aşağıdaki gibi olur.

| Sipariş No | Müşteri Adı | Müşteri Soyadı | Müşteri Tel | Müşteri Adres | Ürün Adı | Sipariş Tarihi | Sipariş Miktarı |
|---|---|---|---|---|---|---|---|
| 1 | Tolga | Yalçın | 5539152030 | İpragaz Mah. Veli Çakmak Cad. No:72 | Bebek Arabası | 21.10.2018 | 1 |
| 2 | Çetin | Songur | 5475245555 | Mevalana Mah. İkbal Cad. No:32 | Tabak | 22.12.2018 | 1 |
| 3 | Tolga | Yalçın | 5539152030 | İpragaz Mah. Veli Çakmak Cad. No:72 | Tava | 25.12.2018 | 1 |
| 4 | Tolga | Yalçın | 5539152030 | İpragaz Mah. Veli Çakmak Cad. No:72 | Çanta | 30.12.2018 | 2 |
| 5 | Çetin | Songur | 5475245555 | Mevalana Mah. İkbal Cad. No:32 | Şemsiye | 02.01.2019 | 3 |
| 6 | Tolga | Yalçın | 5539152030 | İpragaz Mah. Veli Çakmak Cad. No:72 | Kitap | 05.01.2019 | 1 |

Tabloda da görülebileceği gibi birinci seçenekte müşteri bilgisi sürekli bir tekrarda dönüyor. Bunun yerine ikinci seçenekte belirtildiği gibi müşteri bilgileri ayrı tabloda tutulursa yapı aşağıdaki gibi olur.

| No | Adı | Soyadı | Telefon | Adres |
|---|---|---|---|---|
| 1 | Tolga | Yalçın | 5539152030 | İpragaz Mah.<br>Veli Çakmak Cad. No:72 |
| 2 | Çetin | Songur | 5475245555 | Mevalana Mah. İkbal Cad. No:32 |

| Müşteri No | Ürün Adı | Tarih | Miktar |
|---|---|---|---|
| 1 | Bebek Arabası | 21.10.20.18 | 1 |
| 2 | Tabak | 22.12.2018 | 1 |
| 1 | Tava | 25.12.2018 | 1 |
| 1 | Çanta | 30.12.2018 | 2 |
| 2 | Şemsiye | 02.01.2019 | 3 |
| 1 | Kitap | 05.01.2019 | 1 |

Görüldüğü üzere ilgili kayıtlar kendi tablolarında tutulup gerekli yerlere referans numaraları verilirse sipariş tablosunda veri tekrarı engellenmiş olur. Bu şekilde olduğunda müşteri bilgilerinde oluşacak bir değişiklik için bütün sipariş tablosunu dolaşmak yerine sadece müşteri tablosundan kaydı değiştirmek yeterli olacaktır. Referans konusunu Tablo Seviyesinde Veri Bütünlüğü isimli yazımızda bulabilirsiniz.

Veriler tek tabloda iken bir SELECT * FROM ile alınabiliyor iken birden fazla tabloda tutulduğunda JOIN komutlarından faydalanmak gerekmektedir. JOIN komutları birden fazla tablo üzerinde sorgulama yaparak istenen sonuç kümesini sunar.

JOIN komutları üç başlıkta toplanır. Bunlar;

- INNER JOIN
- OUTER JOIN
    - LEFT OUTER JOIN
    - RIGHT OUTER JOIN
    - FULL OUTER JOIN
- CROSS JOIN

# JOIN TEMEL YAPISI

Join işleminin temel yapısı aşağıdaki gibidir:

```
SELECT kolon1, kolon2, kolon3...
FROM Tablo1 [JOIN TÜRÜ] Tablo2 ON [JOIN şarı]
```

## INNER JOIN

INNER JOIN sadece iki tabloda da eşleşen kayıtlar olması durumunda kaydı getirir. Örneğin müşteri ve sipariş tabloları INNER JOIN ile sorgulandığında siparişi olmayan müşteri, sipariş tablosunda eşleşen kaydı olmadığından sonuç ekranına gelmeyecektir. Örnek bir INNER JOIN sorgusu ve sonuç kümesi:

```
SELECT m.Ad, m.Soyad, m.Telefon, m.Adres, s.UrunAd, s.Adet,
s.Tarih
FROM Musteriler m INNER JOIN Siparisler s ON m.Id =
s.MusteriId
```


INNER JOIN

# OUTER JOIN

## LEFT OUTER JOIN

LEFT OUTER JOIN, join kelimesinin solunda kalan tabloyu referans alarak bütün kayıtları getiri. Join kelimesinin sağında kalan tabloda eşleşen kayıt yok ise ilgili kolona NULL değer döner.

```
SELECT m.Ad, m.Soyad, m.Telefon, m.Adres, s.UrunAd, s.Adet,
s.Tarih
FROM Musteriler m LEFT JOIN Siparisler s ON m.Id = s.MusteriId
```


LEFT JOIN

Göründüğü gibi son satırda ilgili müşteriye ait sipariş olmadığından değerler NULL olarak dönüyor.

## RIGHT OUTER JOIN

RIGHT OUTER JOIN, join kelimesinin sağında kalan tabloyu referans alarak bütün kayıtları getiri. Join kelimesinin solunda kalan tabloda eşleşen kayıt yok ise ilgili kolona NULL değer döner.

```
SELECT m.Ad, m.Soyad, m.Telefon, m.Adres, s.UrunAd, s.Adet,
s.Tarih
FROM Musteriler m right JOIN Siparisler s ON m.Id =
s.MusteriId
```


RIGHT JOIN

## FULL OUTER JOIN

FULL OUTER JOIN, her iki tablodan da bütün kayıtları getirir, eşleşmeyen kayıtların karşılıklarını NULL değer olarak döner.

```
SELECT m.Ad, m.Soyad, m.Telefon, m.Adres, s.UrunAd, s.Adet,
s.Tarih
FROM Musteriler m FULL JOIN Siparisler s ON m.Id = s.MusteriId
```


FULL JOIN

# CROSS JOIN

CROSS JOIN, matematikteki kartezyen çarpımı gibi çalışır ve sol tabloya karşılık sağdaki bütün kayıtları getirir. Diğer join türlerinden farklı olarak ON şartı kullanılmaz. Örneğin sol tabloda üç, sağ tabloda 8 kayıt var ise sonuç kümesinde 3 * 8 = 24 kayıt döner.

```
SELECT m.Ad, m.Soyad, m.Telefon, m.Adres, s.UrunAd, s.Adet,
s.Tarih
FROM Musteriler m cross JOIN Siparisler s
```


CROSS JOIN

---

# 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
```