VIEW'LERLE ÇALIŞMAK

7

SQL Server içerisinde hazırlanan sorgular basit sorgular olabildiği gibi birden fazla tablodan veri çeken, karmaşık sorgulardan da olabilmektedir. View'ler veritabanındaki veriler için hazırlanan **SELECT** sorgularına sanal bir görünüm oluşturmak için kullanılır. Yani tablodaki gerçek verileri içermezler. **SELECT** ifadelerinin uzun ve **JOIN** gibi karmaşık sorguları içerdiği durumlarda, bu sorguların kolay anlaşılabilir, kullanılabilir ve yönetilebilir olması için view'ler kullanılır. Aynı zamanda uygulamalarımızda kullandığımız SQL sorgularının içeriğinin son kullanıcı tarafından görüntülenmesini engellemek için de kullanılabilir. View içeriğini şifreleyerek hem SSMS hem de SQL tarafından, bu view'in içeriğinin görüntülenmesini engelleyebiliriz. View'ler sanılanın aksine, sadece **SELECT** işlemlerinin gerçekleştirildiği bir yapı değildir. Oluşturulan bir view üzerinde, belli kurallar ve kısıtlamalara uymak şartıyla DML sorguları (**Insert**, **Update**, **Delete**) gerçekleştirilebilir. Bu özelliklerin tamamını bölümün ilerleyen konularında detaylarıyla inceleyeceğiz.

View'ler Neden Kullanılır?

SQL Server'da view kullanımı doğru yapılmalıdır. Her işlemde view kullanmak doğru değildir ve performansı etkileyebilir.

View'lerin kullanılması gereken durumlardan bazıları şunlardır;

- Uzun veritabanı sorgularını tekrarlamanın önüne geçmek.
- Tabloda son kullanıcının erişmemesi gereken sütunları son kullanıcıdan gizleyip sadece gerekli sütunları göstermek.

- Sorgulama hızını artırmak.
- Parçalanmış tablolar ve Linked Server kavramlarını anlamak ve avantajlarından yararlanmak.
- Sorguları şifreleyerek son kullanıcının sorgu detaylarını görmesini engellemek.

View Türleri

- Regular View

  Sadece tanımlamayı gösteren, veritabanında gerçek verinin saklanmadığı view'ler.

- Indexed View

  Tanımlanan indeks ile, view'den dönecek kayıtlar veritabanında saklanır. Sorgu performansı yüksektir.

- Distributed Partitioned View

  Veritabanı işlem yükünü sunucuya dağıtmaya yarar. SQL Server verilerini bölümler (*partitioning*). Yüksek veri içeren, büyük veritabanlarında performansı artırır.

Alternatifler

View'ler genel olarak **`SELECT`** sorgularını kısaltmak ya da farklı formatlarda gösterebilmek için kullanılır. Ancak programsal tarafta farklı özelliklere ihtiyacımız olabilir. Bu durumda view'in yaptığı işi gerçekleştirecek ancak view ile yapılamayan işlemleri de bilmemizde fayda var.

Tecrübeli bir geliştirici takım çantasındaki araçların olumlu ve olumsuz yönlerini çok iyi bilir.

- View'leri sadece takma isimler vermek için kullanıyorsanız sizin ihtiyacınızı Synonyms nesnesi görebilir. Synonyms nesnesi, herhangi bir veritabanı nesnesi için lakap olarak kullanılabilir.
- View kullanma amacınız sorguları kısa hale getirerek kullanmak olabilir. Ya da view içinde oluşturduğunuz **`SELECT`** sorgusundaki **`WHERE`** filtresinin, dinamik olarak dışarıdan göndereceğiniz değere göre sorgu sonucu getirmesini isteyebilirsiniz.

Bu ve bunun gibi sorgularınızda T-SQL blokları (**`if-else`** ya da döngüler vb.) kullanmak için kullanmanız gereken nesne Stored Procedure'lerdir. Bunlara kısaca **Sproc** denir. Önceden derlenmiş sproc'lar SQL Server'ın başlatılmasıyla birlikte kullanıma hazır hale gelir ve çalıştırılırlar. View'ler ise sorgulanırlar. View'lerin normal **`SELECT`** cümlelerinden farkı yoktur. Hatta aksine basit bir tek tablo sorgusu içeren view, içerisindeki **`SELECT`** sorgusuna göre daha yavaş çalışır.

- View'lerin parametrik olanına ihtiyacınız varsa, doğru seçim **Kullanıcı Tanımlı Fonksiyonlar** olabilir.

VIEW OLUŞTURMAK

View'lerin en temel kullanımını inceleyelim. Aşağıdaki sorgu yapısı genel olarak kullanılan ve genelde geliştirme aşamasında işinizi görecek yapıdır.

```
CREATE VIEW view_ismi
AS
Sorgu_ifadeleri
```

View'lerden daha etkin yararlanabilmek için diğer özelliklerini de kullanabilirsiniz.

```
CREATE VIEW [schema_ismi].view_ismi [sutun_isim_listesi]
[WITH [ENCRYPTION] [, SCHEMABINDING] [, VIEW_METADATA]]
AS
Sorgu_ifadeleri
WITH CHECK OPTION
```

`Production.Product` tablosundaki veriler için bir view tanımlayalım. Bu view'de ürünlerin (*product*) temel bilgileri olan 4 sütun olsun.

```
USE AdventureWorks
GO
CREATE VIEW vw_Urunler
AS
SELECT ProductID, Name, ProductNumber, ListPrice FROM Production.Product;
```

Bu sorgu ile oluşturduğumuz view'i çalıştırarak sonuçlarına bakalım.

```
SELECT * FROM vw_Urunler;
```

	ProductID	Name	ProductNumber	ListPrice
1	1	Adjustable Race	AR-5381	0,00
2	2	Bearing Ball	BA-8327	0,00
3	3	BB Ball Bearing	BE-2349	0,00
4	4	Headset Ball Bearings	BE-2908	0,00

500 civarında kayıt listelenmiştir.

Görüldüğü gibi bu view artık bir tablo gibi sorgulanabilir. İsterseniz yıldız (*) işaretiyle değil, sütun isimlerini yazarak da sorguya kapsam sınırlaması uygulayabilirsiniz.

```
SELECT ProductID, Name, ProductNumber, ListPrice FROM vw_Urunler;
```

Bu sorgu ile tabloyu sorgulamak arasında veri gösterimi açısından bir fark yoktur. Bir view sorgularken kapsam sınırlaması yapabileceğiniz gibi **WHERE** filtreleme komutu gibi diğer **SELECT** sorgularında kullandığınız SQL komutlarını da kullanabilirsiniz.

View'lerde sütunlara isim vermeye ve veri tipi belirtmeye gerek yoktur. View içerisindeki **SELECT** sorgusu ile birlikte gelen sütunların ismi ve veri tipi kullanılacaktır.

View işleminde gerçekleşen asıl olay, basit ya da karmaşık bir sorgunun sanal bir isimle kısa ve daha anlaşılabilir, kolay kullanılabilir hale getirilmesidir. Bu işlemin doğru zamanda, doğru yerde kullanılması gerekir. Çünkü normal bir **SELECT** sorgusunda sorgulama işlemi ve sorgunun karşılığındaki verinin elde edilmesi işlemi gerçekleşir. Ancak view kullanımında bu sorgulama sürecini uzatacak ek katman oluşturulur. Öncelikle view yapısı metadata bilgileriyle ayrıştırılır, daha sonra elde edilen SQL sorgusu veritabanına iletilerek sorgu sonucu alınır. View sorgularında kullandığınız **SELECT** sorgusu, view'ın kendisinden daha hızlı ve performanslı çalışacaktır. Bu nedenle özel gereksinim duymuyorsanız view kullanımından kaçınmalısınız.

Kısıtlamalar

Birçok veritabanı nesnesinde olduğu gibi view'lerin da bazı kısıtlama kuralları vardır. Bunlar;

- View oluşturmak için kullanılacak sorgularda en fazla 1024 sütun (kolon) seçilebilir.
- View'ler geçici tabloları base tablo olarak kullanamazlar.

```
CREATE VIEW vw_hataliView
AS
SELECT sutun1, sutun2
FROM #geciciTablo
```

- View içerisindeki **SELECT** ifadesi, **ORDER BY**, **COMPUTE**, **INTO**, **OPTION** ya da **COMPUTE BY** yan cümleciklerini alamaz.

```
CREATE VIEW vw_hataliView
AS
SELECT sutun1, sutun2
FROM tablo_ismi
COMPUTE SUM(sutun1) BY sutun2
```

Gelişmiş Sorgular İle View Kullanımı

View konusunun başında kullanım sebeplerini sayarken bahsettiğimiz "kod karmaşıklığını azaltmak" özelliği, sanıyorum view kullanımının en çok tercih edilme sebebidir. Uzun kod bloklarını daha kısa ve kolay okunabilir hale getirmek için view ideal bir yöntemdir.

Karmaşık view sorgularında genellikle **JOIN** işlemleri için kullanılır. Bir ya da daha fazla tablodan birçok birleştirme işlemi gerçekleştirilerek elde edilen veriler tek bir kayıt kümesi olarak görüntülenmek istenir. Ancak her seferinde belki onlarca satır SQL kodu içeren **JOIN** sorgularını tekrar yazmak ya da çalıştırmak, uygulama içerisinde kullanmak pek kullanışlı bir yöntem değildir. İşte bu sorunu ortadan kaldırmak için view kullanılabilir.

Production.Product tablosunu kullanarak bir örnek yapalım. Bu tablodaki ürünlerden en çok istenebilecek istatistik sorgusu şu olabilir.

Production.Product tablomuzda hangi ürünler için kaç adet sipariş olduğunu, her siparişte kaç adet satıldığını ve elde edilen geliri gösteren sorgu hazırlayalım.

```
USE AdventureWorks
GO
CREATE VIEW vw_MusteriSiparisler
AS
SELECT
     soh.SalesOrderID,
     soh.OrderDate,
     sod.ProductID,
     p.Name AS UrunAd,
     sod.OrderQty,
     sod.UnitPrice AS BirimFiyat,
     sod.LineTotal AS SatirToplam
FROM
     Sales.SalesOrderHeader AS soh
     JOIN Sales.SalesOrderDetail AS sod
     ON soh.SalesOrderID = sod.SalesOrderID
     JOIN Production.Product AS p
     ON sod.ProductID = p.ProductID;
```

Şimdi hazırladığımız view'i çalıştıralım.

```
SELECT * FROM vw_MusteriSiparisler;
```

	SalesOrderID	OrderDate	ProductID	UrunAd	OrderQty	BirimFiyat	SatirToplam
1	43659	2005-07-01 00:00:00.000	709	Mountain Bike Socks, M	6	5,70	34.200000
2	43659	2005-07-01 00:00:00.000	711	Sport-100 Helmet, Blue	4	20,1865	80.746000
3	43659	2005-07-01 00:00:00.000	712	AWC Logo Cap	2	5,1865	10.373000
4	43659	2005-07-01 00:00:00.000	714	Long-Sleeve Logo Jersey, M	3	28,8404	86.521200
5	43659	2005-07-01 00:00:00.000	716	Long-Sleeve Logo Jersey, XL	1	28,8404	28.840400
6	43659	2005-07-01 00:00:00.000	771	Mountain-100 Silver, 38	1	2039,994	2039.994000
7	43659	2005-07-01 00:00:00.000	772	Mountain-100 Silver, 42	1	2039,994	2039.994000

Bu sorgumuz ile istenilen küçük rapor isteğini gerçekleştirdik ve sonuç olarak da 121.000 üzerinde kayıt listelendi. Bu ve daha gelişmiş sorgulardaki gibi

kodlaması uzun sürecek sorguları gelen isteklere göre hazırlayıp son kullanıcıya sadece view isimlerinden oluşan liste vererek, bu listedeki view'lar ile çalışıp istediği sonuçları hızlıca almasını sağlayabiliriz.

Tablo şeklinde sorgulayabildiğimiz müşteri siparişleri view'ı üzerinde **WHERE** ile filtreleme sorguları da oluşturulabilir.

```
SELECT UrunAd, BirimFiyat, SatirToplam
FROM vw_MusteriSiparisler
WHERE OrderDate = '2005-07-01';
```

	UrunAd	BirimFiyat	SatirToplam
1	AWC Logo Cap	5,1865	5.186500
2	AWC Logo Cap	5,1865	31.119000
3	AWC Logo Cap	5,1865	10.373000
4	AWC Logo Cap	5,1865	15.559500
5	AWC Logo Cap	5,1865	10.373000
6	AWC Logo Cap	5,1865	15.559500
7	AWC Logo Cap	5,1865	20.746000

Tanımlanan View'leri Görmek ve Sistem View'leri

Bir veritabanında tanımlanmış tüm view'leri sistem view'leri ile listeleyebiliriz ve sistem view ya da şemaları master veritabanında bulunur.

View'ler hakkında bilgi içeren sistem şemalarından bazıları;

- **information_schema.tables**

  Veritabanında tanımlı view'lerin listesini içerir. Base tablosu olarak sysobjects'i kullanır.

- **information_schema.view_table_usage**

  Hangi tablolar üzerinde view tanımlı olduğunu gösterir. sysdepends tablosunu kullanır.

- **information_schema.views**

  Tanımlı view'lerin adı, kaynak kodu gibi tanımlama özelliklerini tutar. **syscomments** ve **sysobjects**'ten veri çeker.

Veritabanındaki tüm view'leri listeleyelim.

```
SELECT *
FROM sys.views;
```

Veritabanındaki tüm view'leri listelemek için `JOIN` yapısını kullanalım.

```
SELECT
    s.Name AS SchemaName,
    v.Name AS ViewName
FROM
    sys.views AS v
    INNER JOIN sys.schemas AS s
    ON v.schema_id = s.schema_id
ORDER BY s.Name, v.Name;
```

	SchemaName	ViewName
1	dbo	CustomerOrders_vw
2	dbo	pw_ManagementDepartment
3	dbo	pw_ManagementDepartmentProducts
4	dbo	vw_MusteriSiparisler
5	dbo	vw_Urunler
6	HumanResources	vEmployee
7	HumanResources	vEmployeeDepartment

Bütün view'lerin kaynak kodlarını görmek için;

```
SELECT *
FROM INFORMATION_SCHEMA.VIEWS;
```

	TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	VIEW_DEFINITION
1	AdventureWorks2012	Sales	vStoreWithContacts	CREATE VIEW [Sales].[vStoreWithContacts] AS SELECT ...
2	AdventureWorks2012	Sales	vStoreWithAddresses	CREATE VIEW [Sales].[vStoreWithAddresses] AS SELEC...
3	AdventureWorks2012	Purchasing	vVendorWithContacts	CREATE VIEW [Purchasing].[vVendorWithContacts] AS S...
4	AdventureWorks2012	Purchasing	vVendorWithAddresses	CREATE VIEW [Purchasing].[vVendorWithAddresses] AS ...
5	AdventureWorks2012	dbo	vw_Urunler	CREATE VIEW vw_Urunler AS SELECT ProductID, Nam...
6	AdventureWorks2012	dbo	pw_ManagementDepartment	CREATE VIEW pw_ManagementDepartment AS SELECT *...
7	AdventureWorks2012	dbo	pw_ManagementDepartmentProducts	CREATE VIEW pw_ManagementDepartmentProducts AS ...

Her view'e ait sütunları listeleyelim.

```
SELECT
     v.Name AS ViewName,
     c.Name AS ColumnName
FROM
     sys.columns AS c
     INNER JOIN sys.views AS v
     ON c.object_id = v.object_id
ORDER BY
     v.Name, c.Name;
```

	ViewName	ColumnName
1	CustomerOrders_vw	LineTotal
2	CustomerOrders_vw	Name
3	CustomerOrders_vw	OrderDate
4	CustomerOrders_vw	OrderQty
5	CustomerOrders_vw	ProductID
6	CustomerOrders_vw	SalesOrderID
7	CustomerOrders_vw	UnitPrice

View'lerin Yapisini Görüntülemek

Sahip olduğumuz view'lerin hepsini kendimiz geliştirmiş olmayabiliriz. Ya da uzun süre önce geliştirdiğimiz view'lerin tanımlama bilgilerini hatırlamıyor olabiliriz. Bu durumda view'lerin içeriklerini yani yapılarını görüntülememiz gerekir. Bunun için SSMS kullanılabilir. Bunun için aşağıdaki yolu izleyebilirsiniz.

```
Object Explorer -> Databases -> Veritabani_Ismi -> Views ->
[SağKlik] -> Design
```

Ancak her zaman SSMS ya da herhangi bir geliştirme aracına sahip olamayabiliriz. Bir geliştirici olarak bunun SQL tarafında nasıl yapıldığı bilinmelidir.

Bunun için SQL Server'da en çok kullanılan yöntemleri inceleyeceğiz.

Sys.Sql_Modules

Mevcut view'e ait T-SQL kodlarını **`sys.sql_modules`** sistem kataloğu ile görüntüleyebiliriz.

```
SELECT Definition
FROM sys.sql_modules
WHERE Object_ID = OBJECT_ID('vw_MusteriSiparisler');
```

	Definition
1	CREATE VIEW vw_MusteriSiparisler WITH SCHEMABINDING AS SELECT soh....

Bu sorgu sonucunda **vw_MusteriSiparisler** isimli view'e ait kodlar görüntülenir.

Object_Definition

Aynı işlemi **Object_Definition** fonksiyonunu kullanarak da gerçekleştirebiliriz.

```
SELECT OBJECT_DEFINITION(OBJECT_ID('vw_MusteriSiparisler'));
```

	(No column name)
1	CREATE VIEW vw_MusteriSiparisler WITH SCHEMABINDING AS SELECT soh.Sal...

Sys.SysComments

syscomments tablosunda **Object ID**'ler üzerinden bu işlemi gerçekleştirmemizi sağlar. **Object ID** özelliği SQL Server'ın nesneleri izlemek için nesnelere atadığı tam sayılı değerlerdir. **Object ID**'nin detaylarına girmeyeceğiz ancak bu işlem için kullanımı şu şekildedir.

```
SELECT
    sc.text
FROM sys.syscomments sc
    JOIN sys.objects so
    ON sc.id = so.object_id
    JOIN sys.schemas ss
    ON so.schema_id = ss.schema_id
WHERE so.name = 'vw_MusteriSiparisler' AND ss.name = 'dbo';
```

	text
1	CREATE VIEW vw_MusteriSiparisler WITH SCHEMABINDING AS SELECT soh.Sales...

Bu sorguda **vw_MusteriSiparisler** view'ini arıyoruz. **ss.name** kısmında belirttiğim dbo ise, bu nesnenin hangi şema içerisinde ise o şemanın adı olmalıdır. Ben **vw_MusteriSiparisler** view nesnesini dbo schema (*şema*)'sında oluşturduğum için dbo kullandım. Bildiğiniz gibi SQL Server'da önceden tanımlanmış komutlar ve sistem prosedürleri de arka planda aslında yukarıdaki gibi T-SQL sorguları kullanırlar.

SP_HELPTEXT

Değişiklikler meydana geldiği anda sistem tablolarındaki güncel değişiklikleri gösteren **sp_helptext** komutunu da kullanabiliriz.

```
EXEC sp_helptext 'vw_MusteriSiparisler';
```

	Text
1	CREATE VIEW vw_MusteriSiparisler
2	WITH SCHEMABINDING
3	AS
4	SELECT
5	soh.SalesOrderID,
6	soh.OrderDate,
7	sod.ProductID,
8	p.Name AS UrunAd,
9	sod.OrderQty,
10	sod.UnitPrice AS BirimFiyat,
11	sod.LineTotal AS SatirToplam
12	FROM
13	Sales.SalesOrderHeader AS soh
14	JOIN Sales.SalesOrderDetail AS sod
15	ON soh.SalesOrderID = sod.Sales...
16	JOIN Production.Product AS p
17	ON sod.ProductID = p.ProductID;

sp_helptext ile alacağımız sorgu sonucu diğerleri gibi tek satır halinde değil, T-SQL kodunu geliştirilirken oluşturulan satır sayısı ve formatında görüntülenecektir. **sp_helptext** komutu metinsel parametre aldığı için view ismini tek tırnaklar içerisinde belirtmelisiniz.

sp_helptext komutu da aslında **syscomments** kısmında hazırladığımız sorguyu kullanır. Her şey bu şekilde geliştiricilerin işini hızlandırmak ve kolaylaştırmak, daha az kod yazılmasını sağlamak için hazır komutlar haline getirilir.

T-SQL İLE VIEW ÜZERİNDE DEĞİŞİKLİK YAPMAK

Bir view üzerinde değişiklik yapmak için **ALTER** deyimi kullanılır.

```
ALTER VIEW [schema_ismi].view_ismi [sutun_isim_listesi]
[WITH [ENCRYPTION] [, SCHEMABINDING] [, VIEW_METADATA]]
AS
Sorgu_ifadeleri
WITH CHECK OPTION
```

Bir view oluşturmak ile değiştirmek arasındaki tek fark; **CREATE** yerine **ALTER** deyiminin kullanılmasıdır. **ALTER** ile view'e seçenek eklemek için **WITH** komutundan sonra **ENCRYPTION**, **SCHEMABINDING** komutları kullanılabilir.

View Tanımlamalarını Yenilemek

SQL Server'da view ile ilgili bilgiler metadata olarak tutulur. Bu bilgiler anlık olarak değil, belli periyodlar ile güncellenir. Örneğin; view'de kullanılan tablo ya da tabloların sütunlarında yapısal bir değişiklik olduğunda bu değişiklikler anında metadata bilgilerine yansımayabilir. Metadata'ya yansımayan güncellemeler view tarafından bilinemezler. Bunu işlemi manuel olarak biz şu şekillerde gerçekleştirebiliriz.

```
EXEC sp_refreshview 'vw_MusteriSiparisler';
```

Aynı işlemi **sp_refreshsqlmodule** prosedürünü kullanarak da gerçekleştirebiliriz.

```
EXEC sp_refreshsqlmodule @name = 'vw_MusteriSiparisler';
```

Kod Güvenliği: View'leri Şifrelemek

Geliştirilen uygulamalarda genel olarak veritabanları ve sorguları erişime açık ortamlarda çalışmak zorundadır. Bu gibi durumlarda veritabanı ve sorguların güvenliğinin önemi artmaktadır. Uygulamanın güvenliği ve bazı ticari kaygılardan dolayı, geliştirilen SQL kodlarının içeriği veritabanı katmanında güvenlik altına alınması gerekebilir. View, içeriğindeki kodları şifreleyerek kodlarını gizleme ve koruma imkanı sağlar. Bu özellik kullanılmadan önce şifrelenecek SQL sorgularının yedeği alınmalıdır. Çünkü şifreleme işleminden sonra geliştirici dahil hiç kimse şifrelenen sorguları görüntüleyemez.

View'leri şifrelemek için **ENCRYPTION** operatörü kullanılır. Şifreleme işlemi view oluşturulurken (**CREATE**) belirtilebileceği gibi sonradan güncelleme (**ALTER**) işlemiyle de belirtilebilir.

Şifreli bir view oluşturmak için;

```
CREATE VIEW [schema_ismi].view_ismi [sutun_isim_listesi]
WITH ENCRYPTION
AS
Sorgu_ifadeleri
```

Mevcut bir view'i şifreli hale getirmek için ise **CREATE** yerine **ALTER** kullanmanız yeterli olacaktır.

View güncelleme konusunda belirttiğimiz bir konuyu şifreleme konusunda hatırlamalıyız. Bir view'i **ALTER** ile güncellerken her güncelleme işleminde **WITH ENCRYPTION** ile tekrar şifreli view oluşturmak istediğinizi bildirmelisiniz. Bu bildirimi yapmazsanız SQL Server bu durumu "*şifreleme işleminden vazgeçildi*" şeklinde yorumlayarak view tanımlamalarını şifresiz olarak saklayacaktır.

Bir önceki örnekte tanımladığımız **vw_MusteriSiparisler** view'in tanımını gizleyelim.

```
ALTER VIEW vw_MusteriSiparisler
WITH ENCRYPTION
AS
SELECT
     soh.SalesOrderID, soh.OrderDate, sod.ProductID,
     p.Name AS UrunAd,      sod.OrderQty,
     sod.UnitPrice AS BirimFiyat, sod.LineTotal AS SatirToplam
FROM
     Sales.SalesOrderHeader AS soh
     JOIN Sales.SalesOrderDetail AS sod
     ON soh.SalesOrderID = sod.SalesOrderID
     JOIN Production.Product AS p
     ON sod.ProductID = p.ProductID;
```

Şifreleme işlemini kontrol edelim.

```
sp_helptext 'vw_MusteriSiparisler';
```

```
The text for object 'vw_MusteriSiparisler' is encrypted.
```

İçeriğini görüntülemek istediğimiz view'in encrypted (*şifreli*) olduğunu belirten uyarı aldık.

Schema Binding

Bu özelliği ile view'in bağlı olduğu tablo ya da diğer view'ler gibi nesneleri istenen view'e bağlar. **Schema binding** ile bağlanan view'lerin şema bağlantısı kaldırılmadan üzerinden `Create`, `Alter` gibi bir değişiklik yapılamaz.

Schema Binding işlemine neden ihtiyaç duyarız?

- Bir ya da birden fazla tablomuzla ilişkili bir view oluşturduğumuzda başka bir geliştiricinin ya da unutarak kendi hazırladığımız view'in yapısını bozacak (view içindeki bir sütunun `DROP` edilmesi gibi), sorgunun çalışmasında hatalar oluşturacak şekilde ilgili tablolarda değişiklikler yapmamızı engeller.
- View'i referans alan bir schema bağlantılı kullanıcı tanımlı fonksiyon oluşturacaksanız view'da schema bağlantılı olmak zorundadır.
- View'da index kullanılması gereken durumlar olabilir. Bu durumda view'de indeks kullanabilmek için view'i oluştururken `SCHEMABINDING` özelliği kullanılmak zorundadır.

View İle Verileri Düzenlemek: Insert, Update, Delete

Veritabanında herhangi bir tablo üzerinde gerçekleştirdiğimiz `Insert`, `Update`, `Delete` işlemlerini bir tablonun sanal görünümü olan view'ler üzerinden de gerçekleştirebiliriz. Tabi ki view'in yapısından kaynaklanan bazı kısıtlamalar ile birlikte bu işlem gerçekleştirilebilir.

View üzerinden `Insert`, `Update`, `Delete` işlemleri gerçekleştirmenin kısıtlamaları

- Kümeleme fonksiyonları, `GROUP BY`, `DISTINCT` ve `HAVING` kullanılamaz.
- Insert ile veri eklenecek view'de ilişkili bir tablonun Identity sütununa veri eklenemez. Identity otomatiktir.
- Insert gerçekleştirilecek view'de hesaplanmış sütunlara veri eklenemez.
- Birden fazla tablodan veri çeken view'da gerçekleştirilecek ise `Instead Of Trigger` kullanılmalıdır.
- Bir view'in base tablosu üstünde değişiklik yapabilmesi için `Constraint`, `Unique Indeks`'lere takılmaması gerekir. Base tabloda `NULL` olamayan sütunlar varsa ve

bu sütunlar view'de yer almıyorsa Insert işlemi sırasında bu **NULL** geçilemeyen alanlar **NULL** geçilmiş olacağı için hata meydana gelecektir.

- Kullanıcının view üzerinden erişimi olduğu ancak view sorgusu sonucunda eklediği kaydı göremeyeceği durumlarda kullanıcının veri eklemesi doğru bir yaklaşım olmayacaktır. Bu gibi durumları önlemek için **CHECK OPTION** operatörü kullanılmalıdır.

Production.Location tablosu üzerinde bir view oluşturalım.

```
CREATE VIEW Production.vw_Location
AS
SELECT LocationID, Name,
     CostRate, Availability
FROM Production.Location;
```

Oluşturduğumuz view'e veri ekleyelim.

```
INSERT INTO Production.vw_Location(Name, CostRate, Availability)
VALUES('Dijibil Test',1.10,50.00);
```

Hazırladığımız **vw_Location** isimli view'de veri eklemeyi engelleyecek herhangi bir engel bulunmadığı için bu veri ekleme işlemi sorunsuz bir şekilde gerçekleşecektir.

Eklediğimiz kaydı listeleyelim.

```
SELECT *
FROM Production.vw_Location
WHERE Name = 'Dijibil Test';
```

	LocationID	Name	CostRate	Availability
1	61	Dijibil Test	1,10	50.00

View İle Verileri Düzenlemede Instead Of Trigger İlişkisi

View'ler tek tablo kaynak olarak kullanıldığında ve karmaşık sorgular içermediği taktirde bir tablodan farklı değildir. Bu tür view'lerde veri ekleme, güncelleme

ve silme işlemlerini gerçekleştirebilirsiniz. Ancak karmaşık sorgular içeren, `JOIN` ile birden fazla tabloyu birleştiren sorgular kullanıldığında bu işlemleri gerçekleştirmek bazı şart ve zorluklara tabi tutulur.

- View, birden fazla tablonun birleşimini içeriyorsa, birçok durumda **Instead Of Trigger** kullanmadan veri ekleme (`Insert`) ya da silme (`Delete`) işlemini gerçekleştiremezsiniz. Veri güncelleme işleminde ise, tek tabloyu base tablo olarak kullanmak şartıyla Instead Of Trigger kullanmadan güncelleme gerçekleştirilebilir.
- Eğer view sadece bir tabloya bağlı ise ve view veri eklenecek tüm sütunları içeriyorsa ya da sütunlar `DEFAULT` değere sahipse, veri ekleme işlemini, view üzerinden Instead Of Trigger kullanmadan gerçekleştirebiliriz.
- View tek bir tabloya bağlı olsa da veri eklenecek sütun view'de gösterilmemişse ve `DEFAULT` değere de sahip değilse, bu durumda, view üzerinden veri eklemek için Instead Of Trigger kullanmak zorundasınız.

Trigger konusuna derinlemesine giriş yapmadığımız bu konuda Instead Of Trigger konusunu içermesinin sebebi; view ile trigger'lar arasındaki bağ ve zorunlulukları kavrayabilmenizdir. Özetlemek gerekirse; Instead Of Trigger, hangi işlem için tanımlandıysa onun yerine çalışan bir trigger türüdür. Bu trigger işlemi sonucunda işlemin ne yapacağını görebilir ve oluşacak herhangi bir sorunu çözmek için trigger'da önlem alınabilir.

JOIN İşlemi Olan View'lerde Veri Düzenlemek

`JOIN` sorguları içeren view'lerde veri düzenleme işlemi gerçekleştirmek için Instead Of Trigger'lar kullanılmak zorundadır. `JOIN`, yapısı gereği birden fazla tabloda birden fazla sütun ve sorgu içerir. Bu nedenle veri düzenleme işlemi için gereken bilgilerde hatalar oluşması kaçınılmazdır. Microsoft, bu karmaşanın önüne geçmek için, varsayılan olarak çoklu tablo içeren view'lerde veri düzenlemeye izin vermez. Ancak bu işlemi kullanmak bir gereklilik ise Instead Of Trigger kullanabilirsiniz.

WITH CHECK OPTION Kullanımı

Bir view ile oluşturulan **SELECT** sorgusu, base tabloyu görme yetkisi olmayan ancak kısıtlı bir veriye erişebilen bir view kullanma hakkı varsa ve bu yetkisiyle kullandığı view üzerinden veri ekleme işlemi gerçekleştirebilir. Sorgu başarıyla çalışabilir. Ancak bu eklediği veriyi bir daha göremez.

Bir başka deyişle, view kullanılarak eklenecek veri view'i oluşturan **SELECT** ve **WHERE** sorgusu sonucunda dönecek bir kayıt olmalıdır. Bu şartlara uymayan verinin eklenmesini engellemek için **WITH CHECK OPTION** kullanılır.

İndekslenmiş View'ler

İndekslenmiş view, clustered indeksler biçiminde maddeleştirilmiş, **unique** (*benzersiz*) değerler setine sahip view'dir. Bunun avantajı, view ile birlikte view'in temsil ettiği bilgiyi de elde ederek, çok hızlı arama sağlamasıdır.

Clustered olmak zorunda olan ilk indeksten sonra, SQL Server ilk indeksin işaret ettiği clustered key değerini referans alarak, view üzerinde ek indeksler oluşturabilir.

View üzerinde indeks kullanabilmek için bazı kısıtlamalar

- View **SCHEMABINDING** özelliğini kullanmak zorundadır.
- View tarafından referans gösterilen her fonksiyon **belirleyici** (*deterministic*) olmak zorundadır.
- View, referans gösterilen tüm nesnelerle aynı veritabanında olmak zorundadır.
- Sadece tablolar ve **UDF** (*kullanıcı-tanımlı fonksiyonlar*)'lere referans olabilir. View başka bir view'e referans olamaz.
- View herhangi bir kullanıcı-tanımlı fonksiyona referans gösteriliyorsa, view'ler de schema bağlantılı olmalıdır.
- View'de referans gösterilen tüm tablolar ve UDF'ler iki parçalı (**dbo.Production** gibi) isimlendirme kuralına uymak zorundadır. Ayrıca view ile sahibinin de aynı olması gerekir.
- View ve view'i oluşturan tüm tablolar oluşturulurken, **ANSI_NULLS** ve **QUOTED_IDENTIFIER** özellikleri aktif olmalıdır. Eğer aktif değil ise **SET** komutu ile bunu yapabilirsiniz.

Örnek:

SET ANSI_NULLS ON ya da **SET ANSI_NULLS OFF** ile bu özellik aktif ya da pasif edilebilir.

View bölümünün başında hazırladığımız **vw_MusteriSiparisler** view'ini **ALTER** ile yeniden düzenleyerek indeks kullanımına hazır hale getiriyoruz.

```
ALTER VIEW vw_MusteriSiparisler
WITH SCHEMABINDING
AS
SELECT
     soh.SalesOrderID,
     soh.OrderDate,
     sod.ProductID,
     p.Name AS UrunAd,
     sod.OrderQty,
     sod.UnitPrice AS BirimFiyat,
     sod.LineTotal AS SatirToplam
FROM
     Sales.SalesOrderHeader AS soh
     JOIN Sales.SalesOrderDetail AS sod
     ON soh.SalesOrderID = sod.SalesOrderID
     JOIN Production.Product AS p
     ON sod.ProductID = p.ProductID;
```

Bu view'i daha önce oluşturmadıysanız **ALTER** yerine **CREATE** komutu kullanarak, yeni bir view oluşturabilirsiniz.

Burada dikkat edilmesi gereken bir diğer özellik de view'i **WITH SCHEMABINDING** ile oluşturmuş olmamızdır. **SCHEMABINDING** indekslenmiş view'ler için bir gerekliliktir.

Şimdiye kadar yaptıklarımızla indeksli bir view'e henüz sahip olmasak da view'imizi indekslenebilir hale getirdik. Şimdi bu view'i kullanarak yeni bir indeks oluşturalım. İlk oluşturacağımız indeks hem unique hem de clustered olmak zorundadır.

```
CREATE UNIQUE CLUSTERED INDEX indexedvMusteriSiparisler
ON vw_MusteriSiparisler(SalesOrderID, ProductID, UrunAd);
```

Hazırladığımız view ve indeks sorgumuza dikkat ederseniz 3. parametreyi **`UrunAd`** olarak belirttik. Bunu fark edebilmeniz için bilinçli olarak kullandım. Diğer **`SalesOrderID`** ve **`ProductID`** gerçek sütun isimleriyken UrunAd bir takma isim yani sanal bir isimdir. Ancak sorgu içerisinde bu isimlendirme ile belirttiğimiz için indeks içerisinde de bu ismi kullanmanız gerekir.

```
vw_MusteriSiparisler(SalesOrderID, ProductID, UrunAd)
```

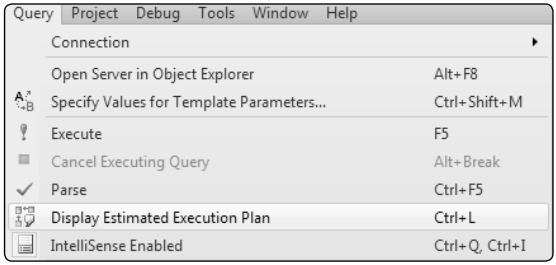
`UrunAd` yerine **`Name`** yazarak çalıştırdığınızda sorgu hata verecektir.

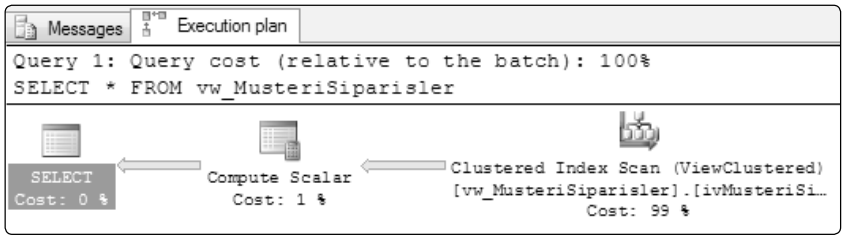
View'i çalıştırarak sorgu sonucunu inceleyelim.

```
SELECT * FROM vw_MusteriSiparisler;
```

	SalesOrderID	OrderDate	ProductID	UrunAd	OrderQty	BirimFiyat	SatirToplam
1	43659	2005-07-01 00:00:00.000	709	Mountain Bike Socks, M	6	5,70	34.200000
2	43659	2005-07-01 00:00:00.000	711	Sport-100 Helmet, Blue	4	20,1865	80.746000
3	43659	2005-07-01 00:00:00.000	712	AWC Logo Cap	2	5,1865	10.373000
4	43659	2005-07-01 00:00:00.000	714	Long-Sleeve Logo Jersey, M	3	28,8404	86.521200
5	43659	2005-07-01 00:00:00.000	716	Long-Sleeve Logo Jersey, XL	1	28,8404	28.840400
6	43659	2005-07-01 00:00:00.000	771	Mountain-100 Silver, 38	1	2039,994	2039.994000
7	43659	2005-07-01 00:00:00.000	772	Mountain-100 Silver, 42	1	2039,994	2039.994000

View'i çalıştırdıktan sonra **SSMS**'de **Query** menüsünde **Display Estamited Execution Plan** menüsünü kullanarak hazırladığımız view'e ait **Execution Plan**'a ulaşabiliriz.





Parçalı View Kullanımı

Parçalı view'ler, aynı yapılara sahip (eşteş) birden fazla tablodaki verilerin sonuçlarını birleştirerek tek kayıt kümesi olarak gösteren view'lerdir. Parçalı view'lerin birkaç farklı kullanım alanı mevcuttur. Örneğin; birden fazla ve farklı veri kaynağından alınan verileri birleştirerek tek bir tablo gibi görüntülenmesini sağlar.

Birden fazla şubesi bulunan bir firmanın, merkez şubesinde bulunan yönetim departmanı diğer şubelerin sipariş bilgilerini görüntüleyebilmek için parçalı view iyi bir çözüm olacaktır. Ayrıca bu şubelerin farklı veritabanlarını kullanıyor olmaları parçalı view kullanımını engellemez. Bir şube SQL Server, diğeri Oracle ya da bir başka veritabanı yazılımı kullanıyor olabilir.

Parçalı view yapısının kullanıldığı bir diğer alan ise, yüksek veri yığınlarının bulunduğu tabloların belli kriterlere göre farklı tablolara ayrılabilmesini sağlamaktır. Örneğin; milyonlarca kayıt içeren bir tablo üzerinde sorgulama, raporlama işlemlerinin gerçekleştirilmesi uzun zaman ve kaynak tüketimine sebep olacağı gibi donanımsal anlamda da yönetimi zor olacaktır. Bu tür sorunların çözümünde parçalı view etkin olarak kullanılabilir.

Parçalı view oluşturmak için kullanılan genel söz dizimi;

```
CREATE VIEW view_ismi
WITH seçenekler
AS
Sorgu_ifadeleri
UNION ALL
SELECT  ifadesi2
...
```

Birden fazla şubesi bulunan ve merkez şube tarafından, şube siparişlerinin takip edildiği bir sipariş takip sistemi geliştiriyoruz. Bu sistemin birden fazla kaynaktan veri almasını sağlayacak sorguyu hazırlayalım.

Öncelikle şubelerimizi oluşturup gerekli kayıtları girmeliyiz.

```
CREATE TABLE Department1Product
(
     DepartmentID   TINYINT NOT NULL,
     ProductID              INT,
     ProductName            VARCHAR(50),
     ProductPrice   MONEY,
     PRIMARY KEY(DepartmentID, ProductID),
     CHECK(DepartmentID = 1)
);
GO
INSERT INTO Department1Product
     (DepartmentID, ProductID, ProductName, ProductPrice)
VALUES
     (1,1,'İleri Seviye SQL Server', 50),
     (1,2,'İleri Seviye Oracle', 50);
```

Kayıt ekleme (**`Insert`**) işleminde, SQL Server özelliklerinden **multiple insert** (*çoklu ekleme*) özelliğini kullandık.

Şubemizde ürünlerle ilgili **`ProductID`**, **`ProductName`**, **`ProductPrice`** bilgilerinin ve ek olarak **`DepartmentID`** sütunumuz var. **`DepartmentID`** sütunu bizim parçalı yapı oluşturmamızı sağlayan sütundur. Şubelerimizi ayrı olarak sorgulayabilmek için **`DepartmentID`** sütununu kullanacağız.

Şimdi diğer şubemizi de tanımlayıp örnek kayıtlar ekleyelim.

```
CREATE TABLE Department2Product
(
     DepartmentID   TINYINT NOT NULL,
     ProductID              INT,
     ProductName            VARCHAR(50),
     ProductPrice   MONEY,
     PRIMARY KEY(DepartmentID, ProductID),
     CHECK(DepartmentID = 2)
);
GO
INSERT INTO Department2Product
     (DepartmentID, ProductID, ProductName, ProductPrice)
VALUES(2,1,'İleri Seviye Java', 50),
      (2,2,'İleri Seviye Android', 50);
```

Şubelerimizin tablolarını oluşturarak örnek kayıtlarımızı ekledik. Şimdi merkez şubemizden bu verileri görüntüleyebilmek, parçalı view yapısını tamamlayabilmek için gerekli view'i oluşturalım.

```
CREATE VIEW pw_ManagementDepartment_Product
AS
SELECT * FROM Department1Product
UNION ALL
SELECT * FROM Department2Product
```

Artık oluşturduğumuz **pw_ManagementDepartmentProducts** view'i çalıştırarak tüm şubelerimizdeki kayıtları görüntüleyebiliriz.

```
SELECT * FROM pw_ManagementDepartment_Product;
```

	DepartmentID	ProductID	ProductName	ProductPrice
1	1	1	İleri Seviye SQL Server	50,00
2	1	2	İleri Seviye Oracle	50,00
3	2	1	İleri Seviye Java	50,00
4	2	2	İleri Seviye Android	50,00

Şubelerimizi oluşturarak merkez şubemizden ürünleri listelemeyi gerçekleştirdik. Dikkat ederseniz şuan sadece şubelerden kayıt girişi yapılabiliyor. Ancak gerçek uygulamada merkez şube bu tablolara müdahale ederek kayıt ekleyebilmelidir. Tablolara kayıt eklemek için oluşturduğumuz view'i kullanabiliriz.

```
INSERT INTO pw_ManagementDepartmentProducts
    (DepartmentID,ProductID,ProductName,ProductPrice)
VALUES
    (1,3,'İleri Seviye Robot Programlama','50'),
    (1,4,'İleri Seviye Yapay Zeka','55'),
    (2,3,'İleri Seviye PHP','40'),
    (2,4,'İleri Seviye JSP','40');
```

Buraya kadar herhangi bir hata ve sorun ile karşılaşmadık. Peki, merkez şubeden veritabanında daha önceden var olan bir ürünü tekrar ekemeye çalışsaydık ne olurdu?

Merkez şube için oluşturduğumuz view'i kullanarak var olan bir kaydı tekrar eklemeye çalışalım.

```
INSERT INTO pw_ManagementDepartmentProducts
     (DepartmentID,ProductID,ProductName,ProductPrice)
VALUES
     (2,1,'İleri Seviye Java',50),
     (2,2,'İleri Seviye Android',50);
```

Bu sorgu sonucunda şube tablolarındaki kısıtlamalardan dolayı, bu verinin var olduğunu ve tekrar eklenemeyeceğini belirten aşağıdakine benzer bir hata alırız.

```
Violation of PRIMARY KEY constraint 'PK__Departme__794757A37141AFB4'.
Cannot insert duplicate key in object 'dbo.Department2Product'. The
duplicate key value is (2, 1).
```

Parçalı View Kullanılırken Dikkat Edilmesi Gerekenler

- Parçalı view'de kullanılan tablolarda **`IDENTITY`** sütun bulunmamalıdır.
- **`UNION ALL`** ifadesi parçalı view için önemlidir. Bu nedenle parçalı view'lerde **`UNION ALL`** ifadesinin şartlarının sağlanması gerekir. Yani birleştirilecek sonuçların alındığı tablolardaki alanlar tür ve sıralama uyumlu döndürülmelidir.
- Parçalı view'de kullanılacak tablo için **Ayırıcı Sütun** (*partitioning column*) bulunmalıdır. Örneğimizde kullandığımız **`DepartmentID`** gibi bir ayırıcı sütun çözüm olabilir. Ayırıcı sütun hesaplanan bir sütun olamaz. Kayıt işlemlerinin kontrolü için örneğimizde oluşturduğumuz gibi ayırıcı sütun bir **`CHECK CONSTRAINT`** tanımı ile denetlenmelidir. Örneğimizdeki son kayıt ekleme işlemindeki hatayı hatırlayın.
- Birden fazla bağlı sunucu üzerindeki tabloyu bir araya getirerek parçalı view oluşturulabilir. Ancak öncelikle bu tabloları barındıran dağıtık sunucuların SQL Server'da tanımlanmış olmalıdır.

View'ları Kaldırmak

Mevcut view'ları basit bir DROP sorgusu ile kaldırabiliriz (silmek).

```
DROP VIEW view_ismi
```

Müşteri siparişleri için hazırladığımız view'ı kaldıralım.

```
DROP VIEW vw_MusteriSiparisler;
```

SSMS İle View Oluşturmak ve Yönetimi (Video)