SQL 11. Gün içeriği

1. **Grouping and Aggregating Data**
   1.1. Group By
   1.2. Having
2. **SubQuery**
   2.1. Basic Subquery
   2.2. Corralated Subquery
3. **Execution Plan**

Group By

Group By (yani gruplandır) yardımcı sözcüğü kayıtları bir alana göre gruplandırmak için kullanılır.

Genel yazılımı aşağıdaki gibidir. GROUP BY deyimini Sql'de WHERE deyiminden sonra kullanılır.

```
Select sütun(lar) from Tablo(lar)
where sart(lar)
group by sütun(lar)
```

Northwind örneği;

```
SELECT CategoryID, COUNT(*) FROM Products
WHERE ProductID BETWEEN 5 AND 50
GROUP BY CategoryID
```

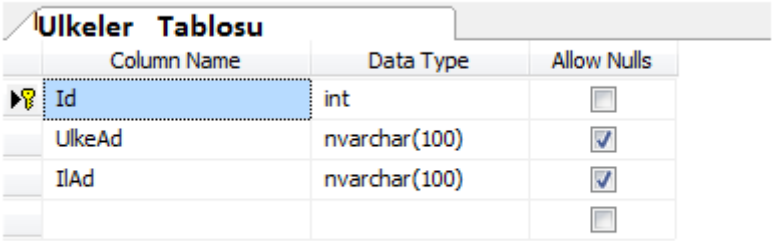
Group By örnek sorular;

```
--Müşterilerinin ülkelere göre sayılarını veren sorguyu yaz.
SELECT Country, COUNT(CustomerID) AS [Müşteri Sayısı] FROM Customers GROUP
BY Country

--Hangi siperiş bana ne kadar kazandırmış.
SELECT OrderID, SUM((UnitPrice*Quantity)*(1-Discount)) AS Gain FROM [Order
Details]
GROUP BY OrderID ORDER BY Gain DESC

--Kategorilere göre stoklarda en çok bulunan ürünler.
SELECT CategoryID, MAX(UnitsInStock) AS Adet FROM Products
GROUP BY CategoryID ORDER BY Adet ASC

--Çalışanlara göre aldıkları sipariş sayılarını raporlayınız.
SELECT EmployeeID, COUNT (*) AS TOTAL FROM Employees
GROUP BY EmployeeID ORDER BY TOTAL DESC
```

```
--Her bir ProductID'de kaç tane ürün var olduğunu bul.
SELECT ProductID, COUNT(ProductID) AS UrunMik FROM Products GROUP BY
ProductID

--Ürünlere göre satışım nasıl? (Ürün-Adet-Gelir)
SELECT P.ProductName,SUM(OD.Quantity*OD.UnitPrice) AS Gelir,
SUM(OD.Quantity) AS Adet FROM [Order Details] OD INNER JOIN Products P ON
OD.ProductID=P.ProductID GROUP BY P.ProductName ORDER BY Gelir DESC

--Hangi kargo şirketine toplam ne kadar ödeme yapmışım?
SELECT S.CompanyName, SUM(O.Freight) AS [Kargo Ödemesi] FROM Orders O INNER
JOIN Shippers S ON O.ShipVia=S.ShipperID GROUP BY S.CompanyName ORDER BY
[Kargo Ödemesi]

-- Ürünlerin ortalama satış fiyatı nedir?
SELECT ProductName, AVG(UnitPrice) AS OrtSatisFiyati FROM Products GROUP BY
ProductName ORDER BY OrtSatisFiyati DESC

--Hangi ülkelere ne kadarlık satış yapmışım? (Join)
SELECT C.Country, SUM(OD.Quantity*OD.UnitPrice) AS Gelir FROM Customers C
INNER JOIN Orders O ON O.CustomerID=C.CustomerID
INNER JOIN [Order Details] OD ON OD.OrderID=O.OrderID
GROUP BY C.Country ORDER BY Gelir DESC

--Çalışanlarım ne kadarlık satış yapmışlar? İstenilen kolonlardan biri:
Çalışanların ad-soyadları (Gelir bazında)
SELECT E.FirstName+' '+E.LastName AS 'AdSoyad',
SUM(OD.UnitPrice*OD.Quantity) AS Gelir FROM Orders O
INNER JOIN [Order Details] OD ON OD.OrderID=O.OrderID
INNER JOIN Employees E ON O.EmployeeID=E.EmployeeID GROUP BY E.FirstName+'
'+E.LastName ORDER BY Gelir ASC
```

Eğitmen için ek bilgiler;

Bunun nerelerde gerekli olduğunu en iyi bir örnek üzerinde açıklayalım. Aşağıdaki şekildeki gibi bir tablomuz olsun. İller ve ülkelerinin bulunduğu bir tablodur.



Tablomuzda bulunan verilerde aşağıdaki gibi olsun.

Ulkeler Tablosu Verileri

Id	UlkeAd	IlAd
1	Türkiye	İstanbul
2	Türkiye	Ankara
3	Türkiye	Mersin
4	Hollanda	Amsterdam
5	İngiltere	Londra
6	İngiltere	Liverpol
7	Amerika	New York
8	Amerika	San Francisco
9	Hindistan	Karaçi
10	Hindistan	New Delhi
11	Pakistan	İslamabad

**Sorumuz şu , hangi ülkenin kaç tane ili olduğunu görmek istiyorum..

Ben şu şekilde bir listeleme istiyorum;

Türkiye 3
Hollanda 1
İngiltere 2
Hindistan 2
Pakistan 1

şeklinde bana hangi ülkenin kaç ili olduğunu getirsin.Yani her ülke gruplara ayrılsın her gruptaki il sayısı gözüksün. İşte gruplama deyimi buradan çıkmaktadır. Belli nitelikleri sağlayan değerleri bir ana çatıda gruplamak istiyoruz. Bunun için Sql'de **GROUP BY** deyimi kullanılır.

SELECT * FROM Ulkeler GROUP BY UlkeAd bu şekilde çalıştırıldığında;

“”Column 'Ulkeler.Id' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause. “”
şeklinde bir hata verecektir. Gruplama yaparken dikkat edilmesi gereken ikinci nokta select deyiminde hangi sütunları göstermek istediğin yanlış bir sütun göstermeye çalışıldığında yukarıdaki group by hatasını verecektir.

Hatada açıklaması şöyle , biz Gruplama yapmak istedik * ile tüm sütunları çeksin dedik, Ulkeler tablosundaki değerlere baktığımızda Türkiye 3 satırda var biz Gruplama ile Türkiye'yi bir satırda yazdırmak ve il sayısını da yazdırmak istiyorduk. * ile Id değerini de yazdırmak istedik fakat hata verdi.Hata da sen gruplama yapıyorsun Id değerini yazdırmak istiyorsun ben Türkiye'yi bir satırda yazacağım ve bu yazacağım satırda hangi Id değerini kullanayım Türkiye 3 satırda 1,2,3 Id değerlerine sahip hangisi kullanacağım.İşte bu bilinmezlikten olayı hata verecektir.

Aynı şekilde sorguyu şu şekilde yapmış olsaydık;

```
SELECT UlkeAd, ilAd FROM Ulkeler GROUP BY UlkeAd
```

“”Column 'Ulkeler.IlAd' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.”” şeklinde benzer hatayı verecekti Türkiye'yi bir satırda gruplarken ben hangi ili yazayım 3 satırda da farklı il değerleri bulunmakta.

Yani Sql Server'da GROUP BY yöntemi ile gruplama yaparken dikkat etmemiz gereken 2.kavram Select ile göstermek istediğimiz sütunlara dikkat edelim , mantıken gittiğimizde zaten hangi sütunların gösterilmeyeceğini rahatlıkla anlayabiliriz.

Sorgumuzun Doğru hali ile yazılımı;

```
SELECT UlkeAd, COUNT(ilAd) FROM Ulkeler GROUP BY UlkeAd
```

Having

Gruplandırma yaparken sorguda bir koşulun da verilmesi gerekiyorsa devreye girer. Kullanılan “HAVING” yardımcı kelimesi “GROUP BY” ile gruplanan kayıtlar üzerinde kısıtlama yapar.

Having, aggregate function’larla kullanılır. Having varsa Group By da olmalıdır.

Genel kullanımı aşağıdaki şekildedir;

```
Select sütun(lar) from Tablo(lar)
where sart(lar)
group by sütun(lar)
having grup_kisitlamasi
```

Northwind örneği;

```
SELECT E.FirstName+' '+E.LastName AS ÇALIŞAN, COUNT(*) AS SATIŞ FROM
Employees E
INNER JOIN Orders O ON E.EmployeeID = O.EmployeeID
GROUP BY E.FirstName+' '+E.LastName HAVING COUNT(O.OrderID) > 50
```

HAVING Kullanma Kuralları

1. Select komutunda GROUP BY yoksa HAVING geçersiz olur.
2. HAVING sözcüğünü izleyen ifade içinde SUM, MIN, MAX, AVG, COUNT fonksiyonlarından en az biri mutlaka olmalıdır.
3. HAVING sözcüğü sadece ve sadece gruplanmış verilerin işlemleri için geçerlidir.
4. WHERE ile birlikte bir Select komutu içinde kullanılabilir.

WHERE ve HAVING arasındaki fark

WHERE bir tablonun tek satırları üzerinde işlem yapan koşullar içinde geçerlidir. HAVING gruplanmış verilerin işlemleri için geçerlidir.

Örnek:

```
select ShipCountry, COUNT(*) as toplam from orders
where ShipCountry LIKE 'B%'
group by ShipCountry
having COUNT(*)>50
order by toplam desc
```

Having örnek sorular;

```
--50'den fazla satış yapan elemanlarımı bul

--Her çalışan ne kadar satış yaptı?
SELECT E.FirstName, E.LastName, COUNT(O.OrderID) AS SatışMiktarı FROM Orders O
JOIN Employees E ON O.EmployeeID=E.EmployeeID
GROUP BY E.FirstName, E.LastName ORDER BY SatışMiktarı DESC
--COUNT(*) yazarsam null değerleri de sayar. COUNT(KolonAdi) yazarsam null değerleri
almaz. Burada OrderID zaten PK olduğu için null olamaz.

SELECT E.FirstName, E.LastName, COUNT(O.OrderID) AS SatışMiktarı FROM Orders O
JOIN Employees E ON O.EmployeeID=E.EmployeeID
GROUP BY E.FirstName, E.LastName HAVING COUNT(O.OrderID)>50 ORDER BY SatışMiktarı DESC
```

```
--Kategorilerine göre, her bir kategori için ortalama ürün fiyatlarını bulunuz, ancak
ürünün birim fiyatı ve ortalama fiyatı 10'dan büyük olan kategorileri bulunuz.
SELECT C.CategoryID, AVG(P.UnitPrice) AS OrtFiyat FROM Categories C JOIN Products P ON
C.CategoryID=P.CategoryID WHERE P.UnitPrice>10 GROUP BY C.CategoryID HAVING
AVG(P.UnitPrice)>10 ORDER BY OrtFiyat DESC
**Where koşulunu yazmasaydık, Having tüm ürünlerin fiyatını alıp hepsinin ortalamasını
alırdı. Where'i koyunca fiyatı 10'dan büyüklerin ortalamasını almış oldu.
```

Eğitmen için ek bilgi

```
**Having - Where = Where, fiziksel tablolardaki fiziksel kolonlar üzerine koşul
koydurur. Products'taki UnitPrice'ı 10'dan büyük olanlar gibi. Having, Select sorgusu
sonucunda gelen Result Set'in üzerinden bir koşul koymak istediğimizde -gruplama
üzerine koşul koyma- kullanılır. Buradaki Result Set:

/* SELECT C.CategoryID, AVG(P.UnitPrice) AS OrtFiyat FROM Categories C JOIN Products P
ON C.CategoryID=P.CategoryID GROUP BY C.CategoryID */
```

Subquery (Alt Sorgu)

Alt sorgu(Subquery) sorgu içerisinde sorgu demektir. İçte ki alt sorgu problemin bir kısmının çözümünü verir kalan kısmı ise ana sorguda çözülür. Bir alt sorgudaki ilk SELECT ifadesine "dış sorgu" ikinci SELECT ifadesinede "iç sorgu" denilir. İç sorgular her zaman ilk önce değerlendirilmelidir çünkü dış sorgular iç sorguların değerlerini kullanmaktadırlar.

Alt sorgu, iç içe sorgu, select içinde select olarak anılan bir yapıdır. Sorguların başka sorgularla beslenmesini sağlayan ve sorgu gücünü artıran bir yapıdır.

İkiye ayrılır;

1. **Basic Subquery**: Ana sorgudan bağımsız alt sorgulardır.

Eğer subquery tek bir yanıt döndürüyorsa where içerisinde '=' ile kullanılabilir.

Eğer subquery bir kolonda birden fazla veri dönüyorsa where içerisinde 'in' ile kullanılabilir.

Eğer tablo dönüyorsa bu alt sorguya bir tablo adı vererek tablo olarak kullanabiliriz yani joinleyebiliriz ya da sorgulayabiliriz.

2. **Correlated Subquery**: Ana sorguyla bağıntılı alt sorgulardır.

Basic(Temel) Subquery

Ana sorgudan bağımsız alt sorgulardır.

```
--Ortalama ücretin üzerinde yer alan ürünler
SELECT ProductName FROM Products WHERE UnitPrice > (SELECT AVG(UnitPrice)
FROM Products)

--Beverages kategorisine ait ürünleri raporla.
SELECT ProductName FROM Products WHERE CategoryID = (SELECT CategoryID FROM
Categories WHERE CategoryName='Beverages')



--Çalışanlarımın ortalama yaşının üzerinde olan çalışanlarım?
SELECT FirstName FROM Employees WHERE
DATEDIFF(YEAR,BirthDate,GETDATE())>(SELECT
AVG(DATEDIFF(YEAR,BirthDate,GETDATE())) FROM Employees)

--fiyatı 30 liradan büyük olan ürünlerin listesi (Subquery & Join)
SELECT DISTINCT P.ProductName FROM Products P INNER JOIN [Order Details] OD
ON OD.ProductID=P.ProductID WHERE OD.UnitPrice>30

SELECT ProductName FROM Products WHERE ProductID IN (SELECT ProductID FROM
[Order Details] WHERE UnitPrice>30)
```

Ekstra zor sorular..

```
--chai ürününden 5ten fazla sipariş vermiş olan müşterilerin listesi
```

Birden fazla müşteri geleceği için " = " değil IN koymalıyız. Çoğul bir sonucu ancak IN'le karşılayabiliriz.

SELECT CompanyName FROM Customers WHERE CustomerID IN (***5ten fazla chai sipariş veren müşteriler***)

5ten fazla sipariş yapılmış sipariş ID'leri;

SELECT CustomerID FROM Orders WHERE OrderID IN (***Chai'den 5'ten fazla sipariş yapılmış***)

5ten fazla chai siparişi yapılmış sipariş ID'leri;

SELECT OrderID FROM [Order Details] WHERE Quantity > 5 AND ProductID = (***Chai'nin ID'si***)

Chai'nin ID'si;

SELECT ProductID FROM Products WHERE ProductName = 'Chai'

```
--kategoriname i B ile D arasında olan fiyatı 30liradan fazla olan ürünler
(Join & Subquery)
SELECT P.ProductName FROM Products P
INNER JOIN [Order Details] OD ON OD.ProductID=p.ProductID
INNER JOIN Categories C ON C.CategoryID=p.CategoryID
WHERE c.CategoryName LIKE '[BD]%' AND P.UnitPrice>30

SELECT ProductID FROM [Order Details] WHERE ProductID IN (SELECT ProductID
FROM Products WHERE UnitPrice>30 AND CategoryID IN (SELECT CategoryID FROM
Categories WHERE CategoryName LIKE '[BD]%'))
```

Correlated(ilişkili, iç içe) Subquery

Ana sorguyla bağıntılı alt sorgulardır.

```
--- ürünler ve kategori isimleri;

SELECT ProductName, (SELECT CategoryName FROM Categories C
WHERE C.CategoryID = P.CategoryID) FROM Products P

---siparişlerimi ID'ler ve siparişin toplam tutarı olarak listeleyiniz;

SELECT OrderID, (SELECT SUM(Quantity*UnitPrice) FROM [Order Details]
D WHERE D.OrderID = O.OrderID ) AS Tutar FROM Orders O ORDER BY
Tutar DESC

--Çalışanlarımın toplam kazandırdıkları kazancı bul. (Subquery & Join)
SELECT E.FirstName, SUM(OD.UnitPrice*OD.Quantity*(1-OD.Discount)) AS
Kazanc FROM Orders O JOIN Employees E ON
E.EmployeeID=O.EmployeeID JOIN [Order Details] OD ON
OD.OrderID=O.OrderID GROUP BY E.FirstName

SELECT FirstName,(SELECT SUM(UnitPrice*Quantity*(1-Discount)) FROM
[Order Details] WHERE OrderID IN (SELECT OrderID FROM Orders O
WHERE O.EmployeeID=E.EmployeeID)) AS Kazanc FROM Employees E
```

Execution Plan

Execution plan en basit ifadesiyle Query Optimizer tarafından hesaplanan ve bir sorgunun en ideal şekilde çalışması için bize önerilen en optimum yoldur. Diğer bir ifadeyle bir Execution plan bize bir sorgunun çağrıldığında nasıl çalışacağını gösterir. Özellikle Veritabanı yöneticilerinin çok sık karşılaştığı performans problemlerini analiz ederken öncelikle çalışma süresi çok uzun süre alan sorgular tespit edilir ve daha sonra bu sorguların Execution planlarına bakılarak sorun tespit edilir. Execution plan konusunu daha iyi anlayabilmek için bir sorgunun SQL Servera gönderildikten sonra işleyişine göz atalım.

Öncelikle SQL Servera herhangi bir sorgu geldiği zaman istediğimiz sonucu bize vermesi için SQL Server tarafından bir dizi işlem yapılır. Kabaca bu işlemleri açıklamak gerekirse öncelikle SQL Server gönderilen sorguyu Parse eder yani yazım kurallarına uygunluğunu denetler ve daha sonra sorguya ilişkin

çalışma planı(Execution plan) Query Optimizer tarafından oluşturulur. Daha sonra Storage engine denilen SQL Server bileşenine gönderilen bu plana göre sorgu işletilerek veriye erişim sağlanır.

Query Optimizer optimum Execution planı hesaplarken minimum CPU ve I/O miktarına göre hesaplar. Bu hesaplama yapılırken objeler üzerinde daha önce oluşturulmuş istatistikler varsa onlardan da yararlanılır ve sorgu sonucunu hesaplar. Bu hesaplama işlemi sorgu için Estimated Cost(tahmini maliyet) olarak adlandırılır. Daha sonra hesaplanan Execution plan daha sonra sorgu çalıştırıldığında tekrar kullanılmak üzere plan Cache konulur.

İki farklı Execution plan vardır;

1. Estimated Execution plan

Query Optimizer tarafında tarafından oluşturulabilecek tahmini planı temsil eder.

2. Actual Execution plan

Actual Execution plan sorgu sonucunun çalışmasıyla elde edilen gerçek planı temsil eder. Estimated Execution plan ve Actual Execution plan birbirinden farklı olabilir. SQL Server'ın plan Cache denilen bellek bölgesinde depolanan planlar Actual Execution planlardır.

Aslında SQL Server'ın her defasında yeni bir Execution plan oluşturmak yerine bir kere oluşturup bunu plan Cache denilen belleğe kaydetmesi ve daha sonra aynı sorgu çalıştırıldığında tekrar kullanılması performans amaçlı yapılmıştır. Her defasında Execution plan hesaplaması SQL Server için ayrıca bir yük teşkil etmektedir.

Eğitmen ekstra bilgi..

Bu yüzden SQL Server bir sorgu için plan Cache'de bir Execution plan varsa o sorgu için aksi belirtilmediği sürece yeni bir Execution plan oluşturmayacak ve var olan Execution planı kullanacaktır.

SQL Server tarafından oluşturulan Execution planlar sürekli plan Cache denilen bellek bölgesinde saklanmaz. SQL Server'ın Lazy Writer adlı işlemi belli aralıklarla plan Cache de dahil olmak üzere Cache'de tutulan tüm verileri temizler. Bunun dışında manuel olarak yaptığımız bazı işlemler de sorgunun Recompile olmasını yani Execution planın tekrar oluşturulmasını sağlar. Fakat yukarıda da belirttiğimiz gibi Execution planın çok sık Recompile edilmesi SQL

Server için yorucu işlemdir. Şimdi Execution planın tekrar oluşturulmasını sağlayan bazı işlemleri listeleyelim.

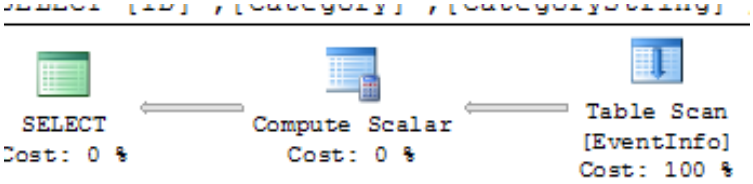
1-Sorguda referans edilen bir tablo üzerinde yapılan Schema değişliği (kolon eklenmesi çıkarılması gibi)

2-Sorgunun kullandığı indekslerden birinin silinmesi

3-Manuel olarak sp_recompile sistem Stored Procedure 'unun çağrılması

4-Aynı sorgu içinde DDL ve DML işlemlerinin beraber kullanılması gibi işlemler her defasında tekrar Execution planın hesaplanmasına yol açar.

5-Sorgunun kullandığı istatistiklerin değişmesi



SQL Server bize Query Optimizer tarafından kullanılan Execution planı yukarıdaki gibi grafiksel şekilde sunacaktır. SQL Server kullandığı Execution planları 3 farklı formatta gösterebilir.

Bu formatlar:

- Graphical Plan (eğitimde sadece bunu gösteriyoruz,diğerleri eğitmen için ek bilgi)
- Text Plan
- XML Plan

Her format aynı Execution planı simgelemesine rağmen arlarında bazı farklar bulunmaktadır. Örneğin graphical planların okunması ve değerlendirilmesi Text planlara göre daha kolay olmasına rağmen, Text planlar graphical planlara göre daha detaylı bilgi içermektedirler.

Grafiksel Plan (Graphical Plan) (eğitimde sadece bunu gösteriyoruz,diğerleri eğitmen için ek bilgi)

Diğer formattaki planlara göre daha kolay ve anlaşılır olmasına rağmen diğer formattaki Execution planlara nazaran daha yüzeysel bilgi içerirler.

Text Plan

Text planların graphical planlara göre daha zor anlaşılmasına rağmen Text planlar daha detaylı bilgi içermektedir.

XML Plan

XML Planlar da Text planlara göre anlaşılması daha kolay olup çok detaylı bilgileri içerirler.

Eğitmen ekstra bilgi..

Execution planlar sayesinde hâlihazırda çalışan sorgularımız incelenerek daha efektif hale getirilmesi sağlanabilir. Fakat SQL Server üzerinde Execution planları görmek için özel bir yetkiye sahip olunması gerekmektedir. Bu yetkiler varsayılan olarak sysadmin,db_owner ve dbcreator rollerinden birine sahip kullanıcılara verilmesine rağmen bu roller dışında herhangi bir user için bu rolleri kullanmadan sadece planları görmesi için özel için verilebilir.

Eğitmen için ek bilgiler;

Query Optimizer: Execution Planı hazırlar. İlk aşama pre-optimization denilen ve sorgunun karmaşıklığına göre kaç plan çıkaracağına karar verdiği aşamadır. Tek tablodan çekilen basit bir SELECT sorgusu için sadece bir plan vardır ve maliyetsiz (zero cost) olarak tanımlanır.

Diğer veritabanları gibi SQL Server'da bulunan query optimizer bir sorguyu etkin bir şekilde çalıştırmak için bir planlama sunar. Bu planlamanın en önemli parçası veri arama aşamasında hangi yöntemi kullanacağıdır. Bu süreçte farklı yol haritaları çıkarılır ve içlerinen maliyeti en düşük olan seçilir. SQL Server'in bir sorguyu çalıştırma yöntemini değiştirmek için hint denilen sorgu ipuçları mevcuttur. Bunları kullanarak o esnada SQL Server için yönlendirilme

yapılabilir. Bu ipuçlarından biri de JOIN hintleridir (LOOP | HASH | MERGE). INNER ve OUTER join türleri sorguların mantıksal olarak operasyonuyla ilgili iken LOOP, HASH ve MERGE yöntemleri sorguların fiziksel operasyonuyla ilgilidir.

NESTED LOOPS JOIN : İki tabloyu birleştirirken döngüleme yöntemini kullanır. Bu esnada tablolardan birini iç(inner) diğerini dış(outer) olarak işleme alır. Dış tablodaki herbir satır için iç tablodaki tüm satırlar okunur. Bu döngü işleminde dış ve iç tablolara ait satırlar eşleştikçe sonuç listesine eklenir. Bu yöntem kayıt sayısı az olan tablolarda iyi bir performans sunar. SQL Server bu yöntemi RIGHT ve FULL join türlerinde kullanmaz.

MERGE JOIN : Bazı kaynaklarda “Merge Scan Join” veya “Sort Merge Join” olarak ta isimlendirilen Merge Join yöntemi Nested Loop Join’den farklı olarak herhangi bir Join yöntemiyle çalışmaktan ziyade koşul olarak verilmiş alanların sıralı olmasını şart koşar. Örneğin “T1.a = T2.b” şeklinde bir koşulda T1 tablosunun T1.a kolonuna göre, T2 tablosunun da T2.b kolonuna göre sıralı olması gerekmektedir. Merge Join, birim zamanda sıralı durumda olan “a” ve “b” kolonları okuyup karşılaştırır. Eğer iki alan eşit ise sözkonusu satırı sonuç listesine ekler. Eğer tablolar sıralı değilse ve sorguda Merge Join işlemi kullanılması isteniyorsa Query Optimizer tarafında öncelikle tablolar sıralı hale getirilir ardından karşılaştırma sürecine geçilir.

HASH JOIN : Daha çok sıralı olmayan büyük tablolarda tercih edilen Hash Join yöntemi öncelikle iki tablodan küçük olanını kullanarak bellekte bir hash table oluşturur ardından büyük tabloyu tarayıp büyük tabloya ait hash değerini bellekteki tabloya ait hash değeriyle karşılaştırarak sonuç listesini oluşturur. Temelde her iki tarafta her satır için bir hash değeri oluşturup bu değer üzerinden ilişki kurar.

Hash Join, diğer Join çalışma yöntemine göre daha maliyetlidir. Query Optimizer bu yöntemi sıralı olmayan büyük tablolar veya beklenilenden fazla dağılmış (big fraction) küçük tablolarda tercih eder.

Son olarak JOIN yöntemlerinin performansıyla ilgili şunu söyleyebiliriz; bu yöntemlerin hangisinin en iyisi olduğunu söylemek zor. Kullanılan tabloların büyüklüğüne ve üzerinde index yapısına bağlı olarak farklı performanslar sergileyebilir. Kısacası tek doğru JOIN yöntemi şudur şeklinde birşey söylemek yanıltıcı olacaktır. Tabloların birleştirilmesinde perfomansı iyileştirecek en önemli konu tabloların ilişkili alanlar üzerinden sıralı olmasıdır.