VERİLERİ SORGULAMAK

3

Veritabanının başlıca amacı; ilişkisel olarak sakladığı veriyi doğru, hızlı, güvenli ve en iyi performans ile sorgulayarak doğru sonuçları vermektir. Tüm veritabanı yönetim sistemlerinde en çok kullanılan sorgular veriyi seçme ve gösterme ile ilgili olan sorgu yapılarıdır.

INSERT, **UPDATE** ya da **DELETE** gibi sorguların yapacakları işler kısıtlıdır. Çünkü işlevsel olarak az sayıda göreve sahiptirler. Ancak **SELECT** sorgusu bir veritabanı uygulamasında sayısız defa kullanılan bir sorgu yapısının temelidir. Bir veriyi gösterirken, formatlarken, sütunları seçerken, filtreleme ve koşullar belirleme, geçici ya da uzaysal veriler ile çalışmak gibi daha birçok iş için kullanılan sorgudur.

Verileri sorgulamak adındaki bu bölümde ilişkisel verileri sorgulayarak, ihtiyaç dahilinde doğru veriyi doğru yöntem ile elde etmeyi öğreneceksiniz.

Operatör Türleri

T-SQL ile programatik işlemler yapabilmek için operatörler kullanılır. Bu operatörlerin bazıları şu şekilde isimlendirilir.

- Karşılaştırma Operatörleri
- Mantıksal Operatörler
- Aritmetik Operatörleri
- Atama Operatörü
- Metin Birleştirme Operatörü

Aritmetik Operatörleri

SQL Server'da T-SQL ile matematiksel hesaplama işlemlerini gerçekleştiren operatörlerdir.

Aritmetik operatörleri şunlardır;

Operatör	Anlamı
%	Mod Alma
*	Çarpma
/	Bölme
+	Toplama
-	Çıkarma

Aritmetik operatörler tüm programlama dilleri ve veritabanı sorgu dillerinde mevcuttur. Hepsinde bir öncelik sıralaması vardır. Bu sıralama işlemi yukarıdaki sıralamaya göre yukarıdan aşağıya doğru tüm dillerde aynıdır.

5 ile 3 sayılarını toplayarak ekranda görüntüleyelim.

```
SELECT 5 + 3;
```

Hesaplama işlemlerinde parantez kullanımına örnek olarak daha karmaşık bir hesap yapalım.

```
SELECT ((5 + 3) * 6) * 2;  -- Bu işlem sonucu 96 olacaktır.
```

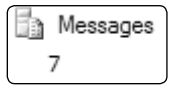
Hesaplama işlemlerini birçok sorgu ifadesiyle birlikte kullanabilirsiniz. Sadece veriyi seçme işlemlerinde değil, stored procedure ya da benzeri özel işlemler yapacağınız nesneler içerisinde de matematiksel işlemlerde kullanabilirsiniz.

Atama Operatörü

Eşit (=) işaretiyle ifade edilir. Değişkenlere değer atarken kullanılır.

Degisken adında bir değişken tanımlayıp **SET** ve atama operatörü (=) kullanarak bir değer atayalım ve ekranda görüntüleyelim.

```
DECLARE @Degisken VARCHAR(10);
SET @Degisken = 7;
PRINT @Degisken;
```



Metin Birleştirme Operatörü

Artı (+) işaretiyle ifade edilir. Birden fazla **string** değeri yan yana birleştirerek, yazmak için kullanılan operatördür. Yan yana getirilen bu kayıtlar gerçekten birleştirilmez, birleşik olarak görüntülemeyi sağlar.

```
SELECT 'DIJIBIL.com' + ' & ' + 'KodLab.com';
```

	(No column name)
1	DIJIBIL.com & KodLab.com

Yukarıdaki sorgu sonucunda, tek tırnaklar arasındaki **string** değerler birleştirilerek ekranda görüntülenecektir.

SELECT İle Kayıtları Seçmek

Kayıtları seçmek (*Selecting*) ve listelemek için kullanılan **SELECT** sorgu yapısını, **T-SQL'e Genel Bakış** bölümünde incelemiştik. Şimdi bir hatırlatma yaparak **SELECT** sorgularının nasıl daha etkili kullanılacağına göz atalım.

SELECT sorgularında tüm tabloyu seçmek için * (*yıldız*) işareti kullanılır. Bu işaret performans açısından verimli olmadığı gibi tüm sütunları getirdiği için ihtiyaç fazlası işlem yapıyor da diyebiliriz.

```
SELECT * FROM Production.Product;
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00
6	317	LL Crankarm	CA-5965	0	0	Black	500	375	0,00	0,00
7	318	ML Crankarm	CA-6738	0	0	Black	500	375	0,00	0,00

Bu işlemi kısıtlayarak daha az sütunu kendi belirttiğimiz şekilde listelemek istersek;

```
SELECT ProductID, Name FROM Production.Product;
```

	ProductID	Name
1	1004	% 20 indirimli ürün
2	1	Adjustable Race
3	461	Advanced SQL Server
4	879	All-Purpose Bike Stand
5	712	AWC Logo Cap
6	3	BB Ball Bearing
7	2	Bearing Ball

SELECT sorgularında sütunların listeleme sırasını belirlemek bizim elimizdedir. Tabloda ilk sırada yer alan **ProductID** sütununu dilersek **SELECT** sorgumuzda son sırada listeleyebilir ya da sorgumuza dahil etmeyebiliriz.

Yaygın bir kullanımı olmasa da şu şekilde de **SELECT** sorgusu oluşturulabilir.

```
SELECT
 Production.Product.Name,
 Production.Product.*
FROM
 Production.Product;
```

	Name	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint
1	Adjustable Race	1	Adjustable Race	AR-5381	0	0	NULL	1000	750
2	Bearing Ball	2	Bearing Ball	BA-8327	0	0	NULL	1000	750
3	BB Ball Bearing	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600
4	Headset Ball Bearings	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600
5	Blade	316	Blade	BL-2036	1	0	NULL	800	600
6	LL Crankarm	317	LL Crankarm	CA-5965	0	0	Black	500	375
7	ML Crankarm	318	ML Crankarm	CA-6738	0	0	Black	500	375

Bu sorgu ile, önce tablodaki **Name** sütunu, sonra * kullanımı ile **Name** dahil tüm sütunlar listelenir.

DISTINCT İLE TEKİLE İNDİRGEMEK

Aynı veriye sahip sütunları sorgularken, sonuç kümesinde her kayıttan sadece bir tane olmasını, yani tekrarlamayı engellemek istiyorsak **DISTINCT** kullanılır.

Personellerin isimlerini, her isimden sadece bir tane olacak şekilde listeleyelim.

```
SELECT DISTINCT FirstName FROM Person.Person;
```

	FirstName
1	Gustavo
2	Sheena
3	Henry
4	Claudia
5	Yvonne
6	Priscilla
7	Scot

UNION VE UNION ALL İLE SORGU SONUÇLARINI BİRLEŞTİRMEK

Bazen **SELECT** sorgularında, birden fazla sorgunun sonucunu birleştirmek ve birleştirilen sorgular üzerinde farklı sorgular oluşturmak gerekebilir.

Bu tür birleştirme işlemleri için **UNION** ve **UNION ALL** kullanılır.

```
SELECT BusinessEntityID, ModifiedDate, SalesQuota
 FROM Sales.SalesPerson
 WHERE SalesQuota > 0
UNION
SELECT BusinessEntityID, QuotaDate, SalesQuota
 FROM Sales.SalesPersonQuotaHistory
 WHERE SalesQuota > 0
 ORDER BY BusinessEntityID DESC, ModifiedDate DESC;
```

Yukarıdaki sorguda iki ayrı sorgu bulunmaktadır. Bu sorgular farklı tablolardan farklı verilerin listelenmesi için kullanılmıştır. Ancak bu sorguların sonucunda listelenecek kayıtların birleştirilerek listelenmesini istiyoruz.

İlk sorguda **Sales.SalesPerson** tablosunda, **SalesQuota** sütun değeri 0'dan büyük olanları, ikinci sorguda ise **Sales.SalesPersonQuotaHistory** tablosundaki bazı koşullara uyan satırların listelenmesini istedik. Son olarak ise, bu kayıtların tamamının birleştirildikten sonra listelenmesi için iki sorgu arasında **UNION** kullandık.

UNION operatörü, ilk sorguda var olan ve ikinci sorguda tekrarlanacak kayıtları filtreler. Yani, sorgular için bir **DISTINCT** uygular. Bu şekilde, kayıt tekrarı olmaması garantilenmiş olur.

UNION ile **UNION ALL** operatörleri aynı işlem için kullanılır. **UNION ALL** operatörünün tek farkı, kayıtlar için bir **DISTINCT** uygulamamasıdır. Yani, **UNION** yerine **UNION ALL** kullanırsanız, ikinci sorguda yer alan kayıtlar herhangi bir kontrole tabi tutulmadan, birinci sorgu sonucunda aynı kayıtlar yer alsa bile ilk birleştirme işlemi gerçekleşir.

WHERE İle Sorgu Sonuçlarını Filtrelemek

Temel **SELECT** işlemlerini gerçekleştirerek birçok kayıt listeledik. Ancak, dikkat ederseniz bu kayıtlar belli koşullara göre hazırlanmış değillerdi. Tablodaki tüm sütunları ya da belirlediğimiz sütunlardaki tüm kayıtları listeleyecek sorgular kullandık.

Gerçek uygulamalarda bu tür sorgular yeterli değildir. Belirli koşulları sağlayan sonuçları isteyeceğimiz yüzlerce farklı istek ve sorunla karşılaşırız. Bu sorunları çözmek için kullanacağımız ve aslında **SELECT** sorgu gücünü tam anlamıyla kullanmamızı sağlayacak özelliği, verileri çeşitli koşullara göre filtreleyerek istediğimiz kayıtları listeleyebilmesidir.

Söz Dizimi:

```
SELECT sutun_ismi1[, sutun_ismi2, ... |  *]
FROM   tablo_ismi
WHERE  şart_ifadeleri
```

Ürünler tablosundan 5 no'lu ürünü getirmek, ürün isminin baş harfi A olan ürünleri getirmek gibi birçok farklı şekilde kullanılabilir.

Mantıksal Operatörler

AND, **OR** ve **NOT** mantıksal operatörleriyle birden çok koşullu ya da birleşik listelemeleri gerçekleştirmek mümkün olmaktadır.

AND

SELECT cümlelerinde **WHERE** filtresi ile birlikte kullanılır. İstenen kayıtların birden fazla özellikle belirtilip filtrelenmesi işleminin gerçekleştirilmesini sağlar.

Production.Product tablosunda **ProductID** sütunu değeri **2** ve **Name** sütunu değeri "*Bearing Ball*" olan kayıtları listelemek için aşağıdaki gibi bir sorgu kullanılabilir.

```
SELECT ProductID, Name FROM Production.Product
WHERE ProductID = 2 AND Name = 'Bearing Ball';
```

	ProductID	Name
1	2	Bearing Ball

Bu sorgu ile istenen kayıtların **ProductID**'si 2 olması ve **Name** sütun değerinin de "*Bearing Ball*" olmasını garanti altına almış oluruz. Bu işlem tek satır üzerinde gerçekleştirilir. Yani **Production.Product** içerisindeki her kayıt için bu sorgulama yapılır ve her kayıttaki **ProductID** ile **Name** sütunları aynı satırda bu değerlere sahip olan kayıtlar sonuç olarak döndürülür.

AND operatörü ile 2 ya da daha fazla karşılaştırma da yapılabilir. Yukarıdaki sorgunun **WHERE** kısmını şu şekilde değiştirerek inceleyiniz.

```
WHERE ProductID = 2 AND Name = 'Bearing Ball' AND ProductNumber = 'BA-8327'
```

	ProductID	Name
1	2	Bearing Ball

Sorgu sonucu aynı olacaktır. Ancak farklı kayıtlar ve işlemler için kullanıldığında daha anlamlı ve farklı sonuçlar üretilebilir.

OR

SELECT cümlelerinde **WHERE** filtresi ile birlikte kullanılır. Birden fazla seçenek belirtip bu seçeneklerin herhangi birine uyan kayıtların getirilmesini sağlar. Satır bazlı değil, tablo bazlı çalışır.

Production.Product tablosunda **Name** sütunu değeri **Blade** ya da **Bearing Ball** olan kayıtları listelemek için aşağıdaki gibi bir sorgu kullanılabilir.

```
SELECT ProductID, Name FROM Production.Product
WHERE Name = 'Blade' OR Name = 'Bearing Ball';
```

	ProductID	Name
1	2	Bearing Ball
2	316	Blade

Bu sorgu ile istenen kayıtların **Name** sütunu değeri Blade ya da Bearing Ball olması garanti altına alınmış olur. Tablo üzerinde **Name** sütunu Blade ve Bearing Ball olanlar getirilecektir.

OR operatörü ile 2 ya da daha fazla karşılaştırma yapılabilir. Yukarıdaki sorgunun **WHERE** kısmını şu şekilde değiştirerek inceleyiniz.

```
WHERE Name = 'Blade' OR Name = 'Bearing Ball' OR Name = 'Chain'
```

	ProductID	Name
1	2	Bearing Ball
2	316	Blade
3	952	Chain

Sorgu sonucunda önceki gibi 2 değil, 3 kayıt dönecektir. Bir OR şartı gerçekleşmez ise diğer OR şartlarına bakılır ve sorgu ile eşleşen kayıtlar döndürülür.

Ürün listesinde seçme işlemi yapan bir kaydın OR 1=1 eklenerek devre dışı bırakılması.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 ProductID < 0 OR 1=1;
```

	ProductID	Name	ProductNumber
1	1	Adjustable Race	AR-5381
2	2	Bearing Ball	BA-8327
3	3	BB Ball Bearing	BE-2349
4	4	Headset Ball Bearings	BE-2908
5	316	Blade	BL-2036
6	317	LL Crankarm	CA-5965
7	318	ML Crankarm	CA-6738

OR 1=1 komutu ,veritabanına yapılan hacking saldırıları için kullanılan en basit yöntemlerden biridir. Bu komut, eklenerek kendisinden önceki tüm şartlar sağlanmasa bile sorgunun kapsadığı tüm kayıtları getirecektir.

SELECT cümlelerinde **WHERE** filtresi ile birlikte kullanılır. Birlikte kullanıldığı operatörün olumsuzluk eki anlamına gelir. Bir koşul ile belirtilen kayıtların haricindeki kayıtları getirir. Örneğin; 10 kayıtlık bir veri kümesinde 1. ve 2. kayıtların koşul içerisinde **NOT** ile birlikte belirtildiği sorgu ifadelerinde bu kayıtlar hariç tüm kayıtlar (8 kayıt) getirilir.

Karşılaştırma Operatörleri

İki değerin birbiriyle karşılaştırılması için kullanılır. Karşılaştırılan verilerin türü aynı olmalıdır. Bir **`string`**, bir başka **`string`** ile karşılaştırılabilir. Bir **`string`** ile sayısal bir tür karşılaştırılamaz.

Operatör	Anlamı
<	Küçük
>	Büyük
=	Eşit
< =	Küçük Eşit
> =	Büyük Eşit
! =	Eşit Değil
< >	Eşit Değil
LIKE	Metin Karşılaştırma Operatörü

`AdventureWorks` veritabanında bazı sorgular geliştirerek bu operatörleri inceleyelim.

`Production.Product` tablosunda **`ProductID`** değeri 5'den küçük olan kayıtları listeleyelim.

```
SELECT ProductID, Name FROM Production.Product WHERE ProductID < 5;
```

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball
3	3	BB Ball Bearing
4	4	Headset Ball Bearings

Aynı sorguyu kullanarak, < işaretini > işareti ile değiştirirseniz 5'den büyük kayıtları listeleyecektir.

`Production.Product` tablosunda **`ProductID`** değeri 4'e eşit olan kayıt ya da kayıtları listeleyelim.

```
SELECT ProductID, Name FROM Production.Product WHERE ProductID = 4;
```

	ProductID	Name
1	4	Headset Ball Bearings

Production.Product tablosunda **ProductID** değeri **316**'dan küçük ve eşit olan kayıtları listeleyelim.

```
SELECT ProductID, Name FROM Production.Product WHERE ProductID <= 316;
```

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball
3	3	BB Ball Bearing
4	4	Headset Ball Bearings
5	316	Blade

Aynı sorguları kullanarak **<=** işareti **>=** işareti ile değiştirirseniz, 316'ya eşit ve büyük olan kayıtları listeler.

Production.Product tablosunda **ProductID** değeri **316** olmayan kayıtları listeleyelim.

```
SELECT ProductID, Name FROM Production.Product WHERE ProductID <> 316;
```

	ProductID	Name
1	1004	% 20 indirimli ürün
2	1	Adjustable Race
3	461	Advanced SQL Server
4	879	All-Purpose Bike Stand
5	712	AWC Logo Cap
6	3	BB Ball Bearing
7	2	Bearing Ball

LIKE

LIKE operatörü karakterler içerisinde karşılaştırma yapmak için kullanılır. **LIKE** (*Liken = Benzerlik Bulmak*) operatörü, karşılaştırma işlemini birçok farklı yöntem ile gerçekleştirebildiği için, basit metinsel arama işlemlerinde tercih edilebilecek yöntemdir.

LIKE ile etkili sorgular oluşturabilmek için joker karakterler kullanılmalıdır.

Joker Karakterler

Sadece **LIKE** operatörü ile kullanılan joker karakterler, arama sorgularında bir ya da daha fazla harfin yerine geçer. Belli aralıklarda, belli harf ile başlayan ya da biten sorgularda joker karakterler kullanılır.

Joker karakterler	Anlamı
%	Birden fazla harf ve rakamın yerini tutar.
_	Bir tek harf ya da rakamın yerini tutar.
[Harf]	Herhangi bir harf yerine gelebilecek harfleri belirtir.
[^Harf]	Herhangi bir harf yerine gelemeyecek harfleri belirtir.
[A-Z]	A ile Z arasındaki harfleri belirtir.

Production.Product tablomuzda **Name** sütunu içerisinde "*ad*" karakterleri bulunan kayıtları listeleyelim.

```
SELECT ProductID, Name FROM Production.Product WHERE Name LIKE '%ad%'
```

	ProductID	Name
1	1	Adjustable Race
2	461	Advanced SQL Server
3	316	Blade
4	399	Head Tube
5	847	Headlights - Dual-Beam
6	848	Headlights - Weatherproof
7	4	Headset Ball Bearings

LIKE operatörü ile basit bir karşılaştırma işlemi yapalım.

Baş harfi "*C*" olan tüm kayıtların **ProductID**, **Name** ve **ProductNumber** sütunlarını getirelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE 'C%';
```

	ProductID	Name	ProductNumber
1	320	Chainring Bolts	CB-2903
2	321	Chainring Nut	CN-6137
3	322	Chainring	CR-7833
4	323	Crown Race	CR-9981
5	324	Chain Stays	CS-2812
6	504	Cup-Shaped Race	RA-2345
7	505	Cone-Shaped Race	RA-7490

LIKE sorgularında dönecek sonucu belirleyen etken, **LIKE**'dan sonra gelen tek tırnaklar içerisindeki joker karakterler ve bunların kombinasyonlarıdır.

Joker karakterleriyle ilgili birkaç farklı örnek yapalım.

"**Be**" ile başlayan tüm ürünlerin kayıtlarını listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE 'Be%';
```

	ProductID	Name	ProductNumber
1	2	Bearing Ball	BA-8327

Son 4 karakteri "**karm**" olan, yani "**karm**" ile biten kayıtları listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE '%karm';
```

	ProductID	Name	ProductNumber
1	319	HL Crankarm	CA-7457
2	317	LL Crankarm	CA-5965
3	318	ML Crankarm	CA-6738

"**hai**" ifadesi içeren tüm kayıtları listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE '%hai%';
```

	ProductID	Name	ProductNumber
1	952	Chain	CH-0234
2	324	Chain Stays	CS-2812
3	322	Chainring	CR-7833
4	320	Chainring Bolts	CB-2903
5	321	Chainring Nut	CN-6137

'**eflector**' ile biten tüm 9 harfli kayıtları listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE '_eflector';
```

	ProductID	Name	ProductNumber
1	506	Reflector	RF-9198

İlk iki harfi belli olmayan, son harfleri '**flector**' olan 9 harfli kayıtları listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE '__flector';
```

	ProductID	Name	ProductNumber
1	506	Reflector	RF-9198

'A' ya da 'C' ile başlayan tüm isimleri listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE '[AC]%';
```

	ProductID	Name	ProductNumber
1	1	Adjustable Race	AR-5381
2	320	Chainring Bolts	CB-2903
3	321	Chainring Nut	CN-6137
4	322	Chainring	CR-7833
5	323	Crown Race	CR-9981
6	324	Chain Stays	CS-2812
7	461	Advanced SQL Server	LR-2398

5 karakterli ve ilk karakteri 'A' ile 'C'arasında olan ve '**lade**' ile biten tüm isimleri listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE '[A-C]lade';
```

	ProductID	Name	ProductNumber
1	316	Blade	BL-2036

'A' ile başlayan ve ikinci karakteri c olmayan tüm isimler.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE 'A[^c]%';
```

	ProductID	Name	ProductNumber
1	1	Adjustable Race	AR-5381
2	461	Advanced SQL Server	LR-2398
3	712	AWC Logo Cap	CA-1098
4	879	All-Purpose Bike Stand	ST-1401

Birden fazla karşılaştırma için **LIKE** kullanım örneği; '**A**' ya da '**B**' ya da '**C**' ile başlayan ürünleri listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
FROM
 Production.Product
WHERE
 Name LIKE 'A%' OR
 Name LIKE 'B%' OR
 Name LIKE 'C%';
```

	ProductID	Name	ProductNumber
1	1	Adjustable Race	AR-5381
2	2	Bearing Ball	BA-8327
3	3	BB Ball Bearing	BE-2349
4	316	Blade	BL-2036
5	320	Chainring Bolts	CB-2903
6	321	Chainring Nut	CN-6137
7	322	Chainring	CR-7833

Karakter karşılaştırma işlemlerinde joker karakterlerin birçok faydasını gördük. Ancak bir metin içerisinde joker karakter bulunma olasılığı da yüksektir. Örneğin; yüzde (%) işareti, birçok durumda kullanılan, bu işareti yukarıda kullandığımız yöntemler ile sorgu içerisinde kullanamayız.

LIKE ile joker karakter karşılaştırması yapabilmek için farklı bazı yöntemler kullanılır.

İçerisinde yüzde (%) işareti geçen kayıtları listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
 FROM
 Production.Product
 WHERE
 Name LIKE '%[%]%';
```

	ProductID	Name	ProductNumber
1	1004	% 20 indirimli ürün	SK-9299

'/' karakteri bulunan kayıtları aramak istediğimizde ise, ek parametrelere gerek yoktur.

```
SELECT
 ProductID, Name, ProductNumber
 FROM
 Production.Product
 WHERE
 Name LIKE '%/%';
```

	ProductID	Name	ProductNumber
1	523	LL Spindle/Axle	SD-2342
2	524	HL Spindle/Axle	SD-9872
3	873	Patch Kit/8 Patches	PK-7098
4	908	LL Mountain Seat/Saddle	SE-M236
5	909	ML Mountain Seat/Saddle	SE-M798
6	910	HL Mountain Seat/Saddle	SE-M940
7	911	LL Road Seat/Saddle	SE-R581

LIKE sorgularına olumsuzluk anlamı katmakta mümkündür. Bunun için **NOT** operatörü kullanılır.

İlk örneğimizde '**Be**' ile başlayan kayıtları listelemiştik. Şimdi ise, **NOT** operatörünü kullanarak, '**Be**' ile başlayan kayıtların olmadığı, yani '**Be**' ile başlamayan tüm kayıtları listeleyelim.

```
SELECT
 ProductID, Name, ProductNumber
 FROM
 Production.Product
 WHERE
 Name NOT LIKE 'Be%';
```

	ProductID	Name	ProductNumber
1	1	Adjustable Race	AR-5381
2	3	BB Ball Bearing	BE-2349
3	4	Headset Ball Bearings	BE-2908
4	316	Blade	BL-2036
5	317	LL Crankarm	CA-5965
6	318	ML Crankarm	CA-6738
7	319	HL Crankarm	CA-7457

Belirli Kayıtlar Arasında Sorgulama Yapmak

SELECT ile **WHERE** koşulu oluştururken genel kullanım, belirli bir kaydı ya da bu kayıttan küçük, büyük ya da eşit olan kayıtların listelenmesidir. Ancak, bazen sadece belirli kayıtlar ya da belirli kayıtların arasındaki kayıtları listelemek isteyebiliriz. Bu gibi durumlarda kullanılacak aralık sorgulama tekniklerini inceleyelim.

BETWEEN .. AND ..

SELECT cümlelerinde **WHERE** koşulu ile birlikte kullanılır. **Between** operatörü iki operand alır ve aldığı operandların aralığını kontrol eder. Between operatöründe ilk değer küçük, ikinci değer büyüktür ve aldığı operandları da sorgulamaya dahil eder.

Production.Product tablosundaki **ProductID** sütun değeri **1** ile **4** arasında olan kayıtları getirelim.

```
SELECT ProductID, Name FROM Production.Product
WHERE ProductID BETWEEN 1 AND 4;
```

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball
3	3	BB Ball Bearing
4	4	Headset Ball Bearings

Sorgu sonucu dönen kayıtlardan görebileceğiniz gibi bu sorgu, 1 ile 4 değerlerini de kapsamaktadır.

`Production.Product` tablomuzdaki 320 ile 400 **`ProductID`** değerine sahip kayıtları listeleyelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 ProductID BETWEEN 320 AND 400;
```

	ProductID	Name
1	320	Chainring Bolts
2	321	Chainring Nut
3	322	Chainring
4	323	Crown Race
5	324	Chain Stays
6	325	Decal 1
7	326	Decal 2

`Between` ile gerçekleştirdiğimiz işlemi, **`AND`** operatörü ile de yapabiliriz.

Bu örneğimizi farklı bir yöntem ile gerçekleştirelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 ProductID >= 320 AND ProductID <= 400;
```

	ProductID	Name
1	320	Chainring Bolts
2	321	Chainring Nut
3	322	Chainring
4	323	Crown Race
5	324	Chain Stays
6	325	Decal 1
7	326	Decal 2

NOT operatörünü kullanarak **BETWEEN** ile belirtilen kayıtları içermeyen sonuçları da listeleyebiliriz.

BETWEEN örneğimizin **WHERE** kısmını değiştirerek **NOT** işlemini uygulayalım.

```
ProductID NOT BETWEEN 320 AND 400;
```

IN ve NOT IN

IN operatörü, sütun içerisinde bir ya da birden fazla belirli kayıt aramak için kullanılır. Aranmak istenen kayıtlar **IN** (bir, iki, ...) içerisinde virgül ile ayrılarak belirtilir.

Production.Product tablomuzda 1, 2, 3 ve 316 **ProductID** değerine sahip kayıtlarımızı listeleyelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 ProductID IN(1, 2, 3, 316);
```

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball
3	3	BB Ball Bearing
4	316	Blade

Sorgumuzun **IN()** kısmında belirtilen kayıtların veritabanında karşılığı olmadığı durumlarda, bulunamayan kayıtlar sorgu dışı bırakılır ve bulunan kayıtlar listelenir.

IN ile gerçekleştirdiğimiz sorguyu, **OR** karşılaştırma operatörü ile de gerçekleştirebiliriz.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 ProductID = 1 OR
 ProductID = 2;
```

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball

IN operatörü ile metinsel arama da yapılabilir.

Name sütununda **Chain** ve **Chairing** isimli ürünleri listeleyelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 Name IN('Chain', 'Chainring');
```

	ProductID	Name
1	952	Chain
2	322	Chainring

IN ile birlikte **NOT** operatörünü kullanarak belirtilen kayıtların dışındaki tüm kayıtlar listelenebilir.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 ProductID NOT IN(1, 2, 3, 316);
```

	ProductID	Name
1	1004	% 20 indirimli ürün
2	461	Advanced SQL Server
3	879	All-Purpose Bike Stand
4	712	AWC Logo Cap
5	877	Bike Wash - Dissolver
6	843	Cable Lock
7	952	Chain

IN operatörünü bir değeri birkaç farklı sütun içerisinde aramak için de kullanabiliriz.

'**Chairing**' değerini **Name** ve **Color** sütunları içerisinde arayarak eşleşen kayıtları listeleyelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 'Chainring' IN(Color, Name);
```

	ProductID	Name
1	322	Chainring

Aynı işlemi bir tarih aramak için de kullanabiliriz.

'*2008-03-11 10:01:36.827*' tarih değerini **SellStartDate** ve **ModifiedDate** sütunlarında arayarak sonuçları listeleyelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
WHERE
 '2008-03-11 10:01:36.827' IN(SellStartDate, ModifiedDate);
```

38.PNG

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball
3	3	BB Ball Bearing
4	4	Headset Ball Bearings
5	316	Blade
6	317	LL Crankarm
7	318	ML Crankarm

AdventureWorks veritabanında neredeyse tüm kayıtlar, aynı **ModifiedDate** değerine sahip olduğu için, bu sorgu sonucunda 500 civarında kayıt listelenecektir. Gerçek uygulamalarda saniye ve salise bilgisinin de aynı olduğu çok fazla kayıt bulunmaz.

SQL Server'da NULL ve Boşluk Kavramı

SQL Server'da boşluk ve **NULL** kavramı önemlidir. Bazen **NULL** değerleri görüntülemek isterken, bazen de bu değerleri manipüle ederek uygulamamızı hata ve **NULL** değer görünümünden kurtarmak isteriz. Bu ve bu gibi durumlarda müdahale edebilmek için boşluk ve **NULL** kavramına hakim olmalıyız.

Metinsel Değerler ile NULL Kullanımı

SQL Server'da metinsel bir değer ile **NULL**'ü birleştirmek varsayılan olarak mümkün değildir. İki metinsel değeri **NULL** ile birleştirerek bunu deneyelim.

```
SELECT 'Cihan' + NULL + 'Özhan';
```

	(No column name)
1	NULL

Sonuç olarak geri dönüşün **NULL** olduğunu görüyoruz. Peki gerçekten birleştirmemiz gerektiği bir durum söz konusu olursa ne yapmalıyız?

SQL Server her ne kadar bu tür durumlarda varsayılan destek sunmasa da bazı esnekliklere sahiptir.

Bir **NULL** ile metinsel değer birleştirme işlemi için aşağıdaki komutu çalıştırmanız yeterlidir.

```
SET CONCAT_NULL_YIELDS_NULL OFF;
```

Şimdi tekrar aynı birleştirme sorgusunu çalıştırdığımızda istediğimiz sonucun görüntülenmesini sağlamış oluruz.

```
SELECT 'Cihan' + NULL + 'Özhan';
```

	(No column name)
1	CihanÖzhan

Bu özelliğin varsayılan değerine geri dönmesi için **OFF** yerine **ON** yazmanız yeterlidir.

```
SET CONCAT_NULL_YIELDS_NULL ON;
```

SPACE Fonksiyonu

SQL Server'da veriler arasında boşluk bırakmak için kullanılan fonksiyondur. Metinsel işlemlerde kullanmak için geliştirilmiştir. Nümerik bir değer ile metinsel bir değer birleştirilmesinde kullanılmaz. **SPACE** fonksiyonu, aldığı parametre değeri kadar boşluk karakteri oluşturur.

Söz Dizimi:

```
SPACE( ifade );
```

İki metinsel değeri aralarında bir boşluk karakteri bırakarak birleştirelim.

```
SELECT 'Cihan' + SPACE(1) + 'Özhan';
```

	(No column name)
1	Cihan Özhan

10 karakter boşluk bırakmak için **SPACE(10)** yazmanız yeterlidir.

Aynı fonksiyonu nümerik değerler ile kullanırsak aralarına boşluk bırakarak birleştirmez, iki nümerik değeri toplayarak sonucu ekrana yazar.

```
SELECT 1 + SPACE(1) + 4;
```

	(No column name)
1	5

İki nümerik değer ile **SPACE** fonksiyonu kullanılırken değerlerden herhangi birinin tek tırnaklar içerisinde yazılmış olması bu değerlerin toplanmamasına

ve hata vermesine sebep olmaz. SQL Server varsayılan olarak tek tırnakları görmezden gelir ve nümerik bir değer olarak algılar.

Bir nümerik ile bir metinsel değer birleştirilmek istendiğinde ise hata verecektir.

```
SELECT 'DIJIBIL' + SPACE(5) + 2013;        --Hata
```

Bu birleştirme işleminin gerçekleştirilebilmesi için tür dönüşümü gerektiğini bildiren bir hata alınır.

```
Conversion failed when converting the varchar value 'DIJIBIL' to
data type int.
```

Sorgularda NULL Değer İşlemleri

Genelde SQL Server'a yeni başlayanlar için **NULL** kavramı zor ve yanlış anlaşılabilmektedir. **NULL** değeri bir alanda boşluk dahil hiç bir verinin bulunmadığını belirtir. Çünkü boşluk bilgisayar dilinde bir değere sahiptir ve bir varlığa sahiptir. **NULL** kavramı bilinmezlik anlamına gelir ve **NULL** olan iki değer birbiriyle eşitlik ya da benzeri bir işleme tabi tutulamaz.

Geliştirilen yazılım ve raporlama gibi veritabanı kayıtlarının işlenerek yönetildiği uygulamalarda **NULL** kayıtlar, kullanılmadığı ya da doğru kullanılmadığı takdirde hem geliştirme aşamasında hatalara sebep olabilir, hem de son kullanıcı açısından **NULL** yazısının bir anlamı olmayacağı için **NULL** yerine farklı değerler gösterilerek kullanışlı bir işlev kazandırılabilir. Uygulamalarda görülen "*Kayıt Bulunamadı*" gibi metinler **NULL** kayıtların son kullanıcıya gösterilmesi için iyi bir örnek olabilir.

Microsoft, **NULL** ile ilgili işlemleri SQL Server'da yönetebilmek için bazı operatör ve fonksiyonlar geliştirmiştir.

IS NULL Operatörü

SQL Server'da sütunlar içerisinde **NULL** olan kayıtların birbiriyle eşitlik gibi sorgulama yapılamayacağını öğrenmiştik. Ancak sorgularımızda **NULL** kayıtları listeleyebilmek için geliştirilmiş bazı komutlar mevcuttur.

IS NULL operatörü **WHERE** filtresiyle birlikte kullanılarak **NULL** olan kayıtları listeler.

Söz Dizimi:

```
SELECT sutun_ismi1 [, sutun_ismi2, ...]
FROM   tablo_ismi
WHERE  sutun_ismi IS [NOT] NULL
```

Production.Product tablosundaki Size sütunu **NULL** olan kayıtları listeleyelim.

```
SELECT
 ProductID, Name, Size
FROM
 Production.Product
WHERE
 Size IS NULL;
```

	ProductID	Name	Size
1	1	Adjustable Race	NULL
2	2	Bearing Ball	NULL
3	3	BB Ball Bearing	NULL
4	4	Headset Ball Bearings	NULL
5	316	Blade	NULL
6	317	LL Crankarm	NULL
7	318	ML Crankarm	NULL

Sorgu sonucunda Size sütun değeri **NULL** olan kayıtlar listelenmiştir. Siz de bu örneği **Color** ve **Weight** sütunları üzerinde kullanarak test edebilirsiniz.

Aynı sorgunun **NOT** kullanımı için aşağıdaki şekilde değiştirmeniz yeterli olacaktır.

```
Size IS NOT NULL
```

	ProductID	Name	Size
1	680	HL Road Frame - Black, 58	58
2	706	HL Road Frame - Red, 58	58
3	709	Mountain Bike Socks, M	M
4	710	Mountain Bike Socks, L	L
5	713	Long-Sleeve Logo Jersey, S	S
6	714	Long-Sleeve Logo Jersey, M	M
7	715	Long-Sleeve Logo Jersey, L	L

ISNULL Fonksiyonu

NULL değerler için kullanılan ve iki parametre alan **NULL** değer fonksiyonudur. İlk parametre olarak kontrol edilecek değeri alır ve **NULL** olduğu takdirde ikinci parametre olarak verilen değeri döndürür.

Söz Dizimi:

```
ISNULL(kontrol_edilecek_deger, NULL_ise_verilecek_deger)
```

Production.Product tablomuzda **Color** sütunu üzerinde bir **NULL** kontrolü yapalım ve eğer kayıt **NULL** ise listelemede "*Renk Yok*" yazarak listeletelim.

```
SELECT
 ProductID, Name, Color,
 ISNULL(Color, 'Renk Yok') AS Renk
FROM
 Production.Product;
```

	ProductID	Name	Color	Renk
1	1	Adjustable Race	NULL	Renk Yok
2	2	Bearing Ball	NULL	Renk Yok
3	3	BB Ball Bearing	NULL	Renk Yok
4	4	Headset Ball Bearings	NULL	Renk Yok
5	316	Blade	NULL	Renk Yok
6	317	LL Crankarm	Black	Black
7	318	ML Crankarm	Black	Black

Production.Product tablomuzda ürünlerin ağırlık bilgisini tutan **Weight** sütununu **ISNULL()** ile **NULL** kontrol işlemine tabi tutalım. Eğer ağırlık bilgisi **NULL** ise sorgu sonucu olarak varsayılan değer 10 verelim.

```
SELECT
 Weight,
 ISNULL(Weight, 10)
FROM
 Production.Product;
```

	Weight	(No column name)
1	NULL	10.00
2	NULL	10.00
3	NULL	10.00
4	NULL	10.00
5	NULL	10.00
6	NULL	10.00
7	NULL	10.00

ISNULL() fonksiyonunu gruplama fonksiyonları ile de kullanabiliriz. Örneğin; yukarıdaki sorgunun sonucunu bir **AVG()** fonksiyonuna aktaralım.

```
SELECT
 Weight,
 AVG(ISNULL(Weight, 10))
FROM
 Production.Product
GROUP BY Weight;
```

	Weight	(No column name)
1	NULL	10.000000
2	2.12	2.120000
3	2.16	2.160000
4	2.18	2.180000
5	2.20	2.200000
6	2.22	2.220000
7	2.24	2.240000

AVG() işleminde **Weight** değeri **NULL** olan kayıtların varsayılan olarak 10 olarak hesaplanması sağlanmış ve buna göre bir ortalama oluşturulmuştur.

Sales.SpecialOffer tablomuzda **Description**, **DiscountPct** ve **MinQty** sütunlarını getiren, **MaxQty** sütununu da **NULL** kontrolü yaparak, eğer **NULL** var ise, **0.00** varsayılan nümerik değeri gösterecek SQL sorgumuzu yazalım.

```
SELECT
 Description, DiscountPct, MinQty,
 ISNULL(MaxQty, 0.00) AS 'Max Miktar'
FROM
 Sales.SpecialOffer;
```

	Description	DiscountPct	MinQty	Max Miktar
1	No Discount	0,00	0	0
2	Volume Discount 11 to 14	0,02	11	14
3	Volume Discount 15 to 24	0,05	15	24
4	Volume Discount 25 to 40	0,10	25	40
5	Volume Discount 41 to 60	0,15	41	60
6	Volume Discount over 60	0,20	61	0
7	Mountain-100 Clearance Sale	0,35	0	0

COALESCE Fonksiyonu

Aldığı parametrelerden **NULL** olmayan ilk ifadeyi döndürür. SQL Server' da yoğun olarak kullanılmayan fonksiyonlardandır.

Söz Dizimi:

```
COALESCE( ifade1, ifade2 [ ...n ] )
```

Production.Product tablosunda bulunan **Color** sütunu değerleri üzerinde **COALESCE()** fonksiyonunu kullanalım. Eğer **Color** sütununda herhangi bir renk değeri girilmemiş ise "*Renk Yok*" yazmasını istiyoruz.

```
SELECT
 ProductID, Name,
 COALESCE(Color,'Renk Yok') AS Renk
FROM
 Production.Product;
```

	ProductID	Name	Renk
1	1	Adjustable Race	Renk Yok
2	2	Bearing Ball	Renk Yok
3	3	BB Ball Bearing	Renk Yok
4	4	Headset Ball Bearings	Renk Yok
5	316	Blade	Renk Yok
6	317	LL Crankarm	Black
7	318	ML Crankarm	Black

Şimdi de **Person.Person** tablosundan **Title**, **FirstName**, **MiddleName** ve **LastName** sütunlarını getirelim. Ve bu sütunlardan **Title** ile **MiddleName** sütunlarına kayıt girilmemiş ise, boş sütunlarda Kayıt Yok yazsın.

```
SELECT
 COALESCE(Title,'Kayıt Yok'),
 FirstName,
 COALESCE(MiddleName,'Kayıt Yok'),
 LastName
FROM
 Person.Person;
```

	(No column name)	FirstName	(No column name)	LastName
1	Kayit Yok	Ken	J	Sánchez
2	Kayit Yok	Terri	Lee	Duffy
3	Kayit Yok	Roberto	Kayit Yok	Tamburello
4	Kayit Yok	Rob	Kayit Yok	Walters
5	Ms.	Gail	A	Erickson
6	Mr.	Jossef	H	Goldberg
7	Kayit Yok	Dylan	A	Miller

ISNULL() fonksiyonundan bazı farkları vardır. **ISNULL()** fonksiyonunun aksine **COALESCE()** birden fazla parametre alabilir. Ayrıca **COALESCE()** fonksiyonu **ANSI** standardı olup, diğer veritabanı yönetim sistemlerinde de kullanılabilirken, **ISNULL()** SQL Server' a özgü bir fonksiyondur.

Şimdi **COALESCE()** fonksiyonunun birden fazla parametre ile nasıl çalıştığına bir örnek verelim.

Production.Product tablomuzdan **Name**, **Class**, **Color** ve **ProductNumber** sütunlarını çekiyoruz. Bu sütunların yanında **COALESCE()** fonksiyonunu kullanarak 3 parametreli bir istek yaptığımız 4. sütunu da ekliyoruz. Bu isteğimiz sonucunda 4. sütuna sahip oluyoruz. Bu sütuna ilk olarak **Class** sütunundan veri gelmesini eğer **Class** sütunu **NULL** ise **Color** sütunundan gelmesini, **Color** sütunu da **NULL** ise **ProductNumber** sütunundan veri getirilerek 4. sütunumuzun doldurulmasını istiyoruz.

```
SELECT
     Name, Class, Color, ProductNumber,
     COALESCE(Class,
               Color,
               ProductNumber) AS FirstNotNull
FROM Production.Product;
```

	Name	Class	Color	ProductNumber	FirstNotNull
1	Adjustable Race	NULL	NULL	AR-5381	AR-5381
2	Bearing Ball	NULL	NULL	BA-8327	BA-8327
3	BB Ball Bearing	NULL	NULL	BE-2349	BE-2349
4	Headset Ball Bearings	NULL	NULL	BE-2908	BE-2908
5	Blade	NULL	NULL	BL-2036	BL-2036
6	LL Crankarm	L	Black	CA-5965	L
7	ML Crankarm	M	Black	CA-6738	M

Bu işlemimizin sonucunda görüntülenecek kayıtların listeleme sırasında, karışmaması için aşağıdaki gibi fonksiyon içerisinde parametreleri **`string`** değerler ile birleştirerek daha anlaşılabilir sorgu sonucu elde edebiliriz.

```
SELECT
     Name, Class, Color, ProductNumber,
     COALESCE('Class : ' + Class,
               'Color : ' + Color,
               'ProductNumber : ' + ProductNumber) AS FirstNotNull
FROM Production.Product;
```

`NULL` işlemlerinde genel olarak **`COALESCE()`** yerine **`ISNULL()`** kullanılmaktadır. Ancak işlem için birden fazla parametre alabilecek bir sorguya ihtiyacınız varsa, seçiminiz **`COALESCE()`** olmalıdır.

NULLIF Fonksiyonu

İki parametre alan ve bu iki parametrenin değerleri aynı ise, geriye **NULL** değer döndüren fonksiyondur. Eğer parametrelerin değeri eşit değil ise ilk aldığı parametreyi geri döndürür.

Söz Dizimi:

```
NULLIF( ifade1, ifade2 );
```

Production.Product tablomuzda **MakeFlag** ve **FinishedGoodsFlag** sütunlarını isteyip 3. sütun olarak da bu iki sütunu **NULLIF()** fonksiyonu ile karşılaştırıp eşit olup olmadığını kontrol ediyoruz. Eğer değerler eşit ise **NULL** değer, değil ise ilk parametrenin değerini döndürür.

```
SELECT
 MakeFlag, FinishedGoodsFlag,
 NULLIF(MakeFlag, FinishedGoodsFlag) AS 'Eşitse Null'
FROM
 Production.Product
WHERE
 ProductID < 10;
```

	MakeFlag	FinishedGoodsFlag	Eşitse Null
1	0	0	NULL
2	0	0	NULL
3	1	0	1
4	0	0	NULL

SELECT ile Verileri Sıralamak

SQL Server'da veriler tablolar içerisinde sütunlarda yer alır. Tablolar ise belirli düzene sahip sütunların sıralı halde sorgulanmasıyla oluşur. Bir veri gösterim (*Select*) sorgusu gerçekleştirildiğinde herhangi bir özel istek belirtilmediği takdirde veriler, tablodaki sütunların sırasına göre listelenir. Bu, SQL Server'ın hızlı veri listelemek için kullandığı varsayılan bir yöntemdir. Ancak istediğimiz takdirde bu sıralamaya müdahale ederek, kendi belirlediğimiz sıraya göre listeleme işlemini gerçekleştirebiliriz. Bu işleme Sort(Sorting = Sınıflandırmak) denir.

ORDER BY

SQL Server'da Sıralama işlemini gerçekleştiren komut **`Order By`**'dır.

Söz Dizimi:

```
SELECT      sutun_isim1, sutun_isim2
FROM        table_name
ORDER BY    siralanacak_sutun [ASC|DESC];
```

`Production.Product` tablomuzdaki ürünleri Name sütununa göre listeleyelim.

```
SELECT
 ProductID, Name
FROM
 Production.Product
ORDER BY Name;
```

	ProductID	Name
1	1004	% 20 indirimli ürün
2	1	Adjustable Race
3	461	Advanced SQL Server
4	879	All-Purpose Bike Stand
5	712	AWC Logo Cap
6	3	BB Ball Bearing
7	2	Bearing Ball

Listelenen kayıtların sıralamasına dikkat edin. A'dan Z'ye doğru bir sıralama gerçekleştirildi. Bu, SQL Server'ın varsayılan sıralama algoritmasıdır.

SQL Server'da iki tip sıralama yöntemi vardır. Bunlar;

- **ASC** (*Ascending*)

SQL Server için varsayılan sıralama yöntemidir. Artan sırada listeleme yapar. Metinsel veri için A'dan Z'ye, nümerik veri için ise 0'dan 9'a doğru sıralama gerçekleştirir.

ASC tipinde sıralama yapmak için Order By'dan sonra özellikle bir belirtim yapmaya gerek yoktur. Ancak kod okunabilirliği açısından kullanılmasında fayda vardır.

- **DESC** (*Descending*)

ASC'nin tam tersidir. Azalan sırada listeleme yapar. Metinsel veri için Z'den A'ya, nümerik veri için ise 9'dan 0'a doğru sıralama gerçekleştirir.

`Production.Product` tablosundaki kayıtları **`ProductID`** sütununa göre artan şekilde sıralayalım.

```
SELECT
 ProductID, Name
FROM
 Production.Product
ORDER BY
 ProductID ASC;
```

	ProductID	Name
1	1	Adjustable Race
2	2	Bearing Ball
3	3	BB Ball Bearing
4	4	Headset Ball Bearings
5	316	Blade
6	317	LL Crankarm
7	318	ML Crankarm

`Production.Product` tablosundaki kayıtları **`ProductID`** sütununa göre azalan şekilde sıralayalım.

```
SELECT
 ProductID, Name
FROM
 Production.Product
ORDER BY
 ProductID DESC;
```

	ProductID	Name
1	1004	% 20 indirimli ürün
2	999	Road-750 Black, 52
3	998	Road-750 Black, 48
4	997	Road-750 Black, 44
5	996	HL Bottom Bracket
6	995	ML Bottom Bracket
7	994	LL Bottom Bracket

Order By ile birden fazla sütuna göre de sıralama gerçekleştirebiliriz.

```
SELECT
 ReorderPoint, Name,
 ProductID
FROM
 Production.Product
ORDER BY
 ReorderPoint, Name ASC;
```

	ReorderPoint	Name	ProductID
1	3	All-Purpose Bike Stand	879
2	3	AWC Logo Cap	712
3	3	Bike Wash - Dissolver	877
4	3	Cable Lock	843
5	3	Classic Vest, L	866
6	3	Classic Vest, M	865
7	3	Classic Vest, S	864

Birden fazla sıralama sütunu belirttiğimizde durum biraz karışık gibi görülebilir. Aslında gayet basittir. İlk sıralanması istediğimiz sütunda aynı olan kayıtlar olduğu durumlarda, ikinci sıradaki sıralama sütununa göre sıralama işlemi gerçekleşir.

Yukarıdaki örneğimizde **ReorderPoint** sütunu değerinin aynı olması halinde, **Name** sütununa göre sıralama işlemi gerçekleşecektir.

Sıralama işlemi için sütun isimlerini yazmak mecburiyetinde değiliz. Bunun yerine yukarıda **SELECT** sorgusunda belirtilen sütunların sıra numarasını belirterek de bir sıralama gerçekleştirilebilir.

Sütun ismine göre hazırladığımız sorguyu sütun sırasına göre tekrar çalıştıralım.

```
SELECT
 ReorderPoint, Name,
 ProductID
FROM
 Production.Product
ORDER BY
 3 ASC;
```

	ReorderPoint	Name	ProductID
1	750	Adjustable Race	1
2	750	Bearing Ball	2
3	600	BB Ball Bearing	3
4	600	Headset Ball Bearings	4
5	600	Blade	316
6	375	LL Crankarm	317
7	375	ML Crankarm	318

`Order By 3`, aslında **`Order By ProductID`** ile aynı anlama gelmektedir. 3 yerine 2 kullanılırsa, Name sütununa denk gelecektir ve sıralamayı Name sütununa göre gerçekleştirecektir.

`Order By` ile büyükten küçüğe ve küçükten büyüğe sıralama yapılabileceği rastgele sıralama da yapılabilir. Bu işlem için bir sistem fonksiyonu olan **`NEWID()`** kullanılabilir.

```
SELECT
 ProductID, Name
FROM
 Production.Product
ORDER BY
 NEWID();
```

	ProductID	Name
1	932	ML Road Tire
2	892	HL Touring Frame - Blue, 54
3	831	ML Mountain Frame - Black, 44
4	810	HL Mountain Handlebars
5	359	Thin-Jam Hex Nut 9
6	482	Metal Sheet 2
7	902	LL Touring Frame - Yellow, 58

TOP Operatörü

Yüksek miktarda kayıt içeren tablolarda belirli sayıda kayıt listelemek için kullanılır. SQL standartlarında yer alan bir komuttur. Listelenmesi istenen kayıtlar sayı olarak belirtilebileceği gibi dönen kayıtların yüzde ile oransal listelemesi de yapılabilir.

Söz Dizimi:

```
SELECT TOP number|percent sutunlar
FROM tablo_isim
WHERE [condition]
```

Production.Product tablomuzdan dönen ilk 5 kaydı listeleyelim.

```
SELECT TOP 5 * FROM Production.Product;
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00

TOP operatörü ile yüzde kullanarak dönen kayıtların yüzde ile belirtilen kısmını da listeleyebiliriz.

```
SELECT TOP 1 PERCENT * FROM Production.Product;
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00
6	317	LL Crankarm	CA-5965	0	0	Black	500	375	0,00	0,00

TOP 1 PERCENT bölümü, tüm ürünlerin **%1**'i kadarını listelemeyi belirtir.

Aynı sorgu yöntemi ile bir filtre de oluşturulabilir. Bu sayede istenilen kriterlerdeki kayıtlar arasında bir **TOP** işlemi gerçekleştirilir.

'c' ile başlayan ürünleri bularak dönen kayıtlardan 5 tanesini listeleyelim.

```
SELECT TOP 5 * FROM Production.Product WHERE Name LIKE 'C%';
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	320	Chainring Bolts	CB-2903	0	0	Silver	1000	750	0,00	0,00
2	321	Chainring Nut	CN-6137	0	0	Silver	1000	750	0,00	0,00
3	322	Chainring	CR-7833	0	0	Black	1000	750	0,00	0,00
4	323	Crown Race	CR-9981	0	0	NULL	1000	750	0,00	0,00
5	324	Chain Stays	CS-2812	1	0	NULL	1000	750	0,00	0,00

TOP Fonksiyonu

T-SQL fonksiyonlarından biri olan **TOP** fonksiyonu, **TOP** operatörü ile benzer şekilde çalışır. **Order By** ile birlikte kullanılır.

Söz Dizimi:

```
TOP (expression) PERCENT WITH TIES
```

Production.Product tablosundaki kayıtların ilk 5 tanesini, **ProductID** sütununa göre listeleyerek sıralayalım.

```
SELECT
 TOP(5) WITH TIES *
FROM
 Production.Product
ORDER BY
 ProductID;
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00

Production.Product tablosundaki kayıtların ilk **%5**'lik kısmını **ProductID** sütununa göre listeleyelim.

```
SELECT
 TOP(5) PERCENT WITH TIES *
FROM
 Production.Product
ORDER BY
 ProductID;
```

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	StandardCost	ListPrice
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00	0,00
4	4	Headset Ball Bearings	BE-2908	0	0	NULL	800	600	0,00	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00	0,00
6	317	LL Crankarm	CA-5965	0	0	Black	500	375	0,00	0,00
7	318	ML Crankarm	CA-6738	0	0	Black	500	375	0,00	0,00

Bu sorguda bize yüzde özelliğini kazandıran **PERCENT** komutudur.

Tabloları Birleştirmek

İlişkisel verilerde farklı tablolarda yer alan ve birbirini tamamlayan kayıtların tek bir tabloymuş gibi sorgu sonucunda birleştirerek listeleme işlemleri **JOIN** ifadeleri ile gerçekleştirilir.

Bir tablo kendisi ile birleştirilebileceği gibi farklı birden fazla tablo ile de birleştirilebilir. Birleştirme işleminde istenen işleme göre farklı yaklaşımlar mevcuttur. Bu yaklaşımların tamamını karşılamak için **JOIN**'ler kendi içerisinde farklı isimlendirmeler ve farklı komutlara sahiptir.

Örneğin; e-ticaret sisteminde, müşterilerin siparişlerini tutan tablo ile müşteri tablosu birleştirilerek müşterinin hangi ürünleri sipariş ettiğini, hatta hangi kategorilerdeki ürünlere karşı daha fazla satın alma işlemini gerçekleştirdiği gibi bilgileri listelemek için kullanılabilir.

İleri seviye veritabanı programlama için **JOIN**'ler vazgeçilmez ve önem arz eden konulardan biridir. Genel olarak veritabanı programlamada kavram açısından karıştırılan bir konu olması nedeni ile bu konuya geniş yer ayırarak detaylı örnek ve analizler gerçekleştireceğiz.

JOIN'ler, her ne kadar karmaşık yapılara sahip olabilse de temel söz dizimi şu şekildedir.

```
SELECT sutun_ismi1[, sutun_ismi2, ...]
FROM tablo_ismi1 [, tablo_ismi2, ...]
WHERE tablo_ismi1.sutun_ismi1 = tablo_ismi2.sutun_ismi2
...
```

Bu **JOIN** kullanımı, klasik **JOIN** modelidir. Karmaşık olmayan, ancak ileri seviye uygulamalarda çok kullanılmayan temel bir **JOIN** yapısıdır. **ANSI** standartlarında bir **JOIN** söz dizimi olduğu için SQL Server'da desteklemektedir.

Klasik JOIN

WHERE cümleciği ile gerçekleştirilen bu **JOIN**, **ANSI** standartlarında ve yukarıda belirttiğimiz söz dizimine sahip olan **JOIN** yöntemidir. SQL Server klasik **JOIN**'i bir **INNER JOIN** olarak işleme alır.

Satılan ürünleri listeleyelim.

```
SELECT SOD.ProductID, P.Name
FROM Sales.SalesOrderDetail AS SOD, Production.Product AS P
WHERE SOD.ProductID = P.ProductID;
```

	ProductID	Name
1	707	Sport-100 Helmet, Red
2	707	Sport-100 Helmet, Red
3	707	Sport-100 Helmet, Red
4	707	Sport-100 Helmet, Red
5	707	Sport-100 Helmet, Red
6	707	Sport-100 Helmet, Red
7	707	Sport-100 Helmet, Red

Bu sorgu ile satılan tüm ürünler listelenir. Ürün birden fazla satılmış da olsa satıldığı kadar kayıt listelenecektir.

Geliştirilmesi en kolay **JOIN** yöntemidir. Temel kullanımı kavrayabilmek için birleştirmek istediğiniz iki tabloyu da inceleyebilirsiniz.

Örneğin, yukarıdaki sorguda **Sales.SalesOrderDetail** ve **Production.Product** tabloları kullanıldı. Bu işlemde belirlememiz gereken öncelik hangi sütunların birleştirileceğidir.

İki tabloyu da alt alta tek seferde çalıştırarak tablo yapısını inceleyebilirsiniz.

```
SELECT * FROM Sales.SalesOrderDetail
SELECT * FROM Production.Product
```

	SalesOrderID	SalesOrderDetailID	CarrierTrackingNumber	OrderQty	ProductID	SpecialOfferID	UnitPrice	UnitPriceDiscount
1	43659	1	4911-403C-98	1	776	1	2024,994	0,00
2	43659	2	4911-403C-98	3	777	1	2024,994	0,00
3	43659	3	4911-403C-98	1	778	1	2024,994	0,00

	ProductID	Name	ProductNumber	MakeFlag	FinishedGoodsFlag	Color	SafetyStockLevel	ReorderPoint	Standarc
1	1	Adjustable Race	AR-5381	0	0	NULL	1000	750	0,00
2	2	Bearing Ball	BA-8327	0	0	NULL	1000	750	0,00
3	3	BB Ball Bearing	BE-2349	1	0	NULL	800	600	0,00
4	4	Headset Ball B...	BE-2908	0	0	NULL	800	600	0,00
5	316	Blade	BL-2036	1	0	NULL	800	600	0,00

Bu sorgu sonucunda iki tablodaki verilerde alt alta listelenecektir. Birleştirmek istenen sütunlar belirlenerek sorgu şu şekilde yazılabilir.

- Birleştirmek istenen sütunlar belirlenir.

  Sales.SalesOrderDetail tablosundaki **ProductID** sütunu ile **Production.Product** içerisindeki **Name** sütunu birleştirilecek.

  Yani aşağıdaki gibi, şema_ismi.tablo_ismi.sütun_ismi olarak nitelendirebiliriz.

```
Sales.SalesOrderDetail.ProductID  -  Production.Product.Name
```

- Tablo isimlendirmesi uzun olacağı için basit bir sorgu bile çok karmaşık hal alabilir. Bu nedenle tabloları takma ad ile isimlendirin.

```
...
FROM Sales.SalesOrderDetail AS SOD, Production.Product AS P
...
```

  Artık bir önceki uzun isimlendirme şu şekilde olacaktır.

```
SOD.ProductID    -    P.Name
```

- Son olarak belirlenen ve kısaltılan isimler ile birlikte WHERE cümleciğinde tabloları eşitleyin.

```
...
WHERE SOD.ProductID = P.ProductID
...
```

JOIN işleminin temel mantığı bundan ibarettir. **JOIN** işlemini zorlaştıran şey sorgu yapısı değil, birleştirilecek tabloların çokluğu ve listelenmek istenen verinin karmaşık özelliklere sahip olmasıdır.

SQL Server'da JOIN Mimarisi

SQL Server performans ve sonuç olarak en iyi işlemi gerçekleştirecek **JOIN**'i kendisi belirler.

SQL Server, **JOIN** sorgularını oluşturmak için şu alt seviye **JOIN** yöntemlerini kullanır.

- **Merge Join:** Birinci tablodan bir kayda karşılık ikinci tablodan alınan kaydın ilişkili olması şartı ile çalışır.
- **Hash Join:** Tablolardan her ikisinde de uygun bir indeks bulunamazsa ya da WHERE cümleciği yer alıyorsa kullanılır.
- **Nested Loop Join:** Birleştirilecek iki tablodan az kaydı olan tablo dışarıdaki, çok olan tablo içerideki döngü gibi çalışır. Adından da anlaşılacağı gibi iç içe iki döngü gibi çalışır.

SQL Server'da `JOIN` işlemlerini bu 3 yöntemden birine göre çalıştırmak için özel ayar belirtilebilir.

Ayar belirtme işlemi için `JOIN` sorgusunun altına şu şekilde bildirim yapılır.

```
OPTION (MERGE JOIN)
```

Örnek:

```
SELECT S.CountryRegionCode, S.StateProvinceCode,
       T.TaxType, T.TaxRate
FROM Person.StateProvince AS S
LEFT OUTER JOIN Sales.SalesTaxRate AS T
ON S.StateProvinceID = T.StateProvinceID
OPTION (MERGE JOIN)
```

	CountryRegionCode	StateProvinceCode	TaxType	TaxRate
1	CA	AB	1	14,00
2	CA	AB	2	7,00
3	US	AK	NULL	NULL
4	US	AL	NULL	NULL
5	US	AR	NULL	NULL
6	AS	AS	NULL	NULL
7	US	AZ	1	7,75

Diğer yöntemlerde aynı şekilde kullanılabilir.

```
OPTION (LOOP JOIN)
```

ya da

```
OPTION (HASH JOIN)
```

INNER JOIN

İki tablo arasında birleştirme yaparken, her iki tabloda da yer alan değerler seçilir. Tablolardan sadece birinde yer alıp, diğerinde ilişkili değer bulunmayan kayıtlar seçilmez. **JOIN** yöntemlerinde en çok kullanılan tablo birleştirme yöntemidir.

Personellerin ad, soyad, personel türü ve telefon numarası bilgilerini listeleyelim.

```
SELECT P.FirstName, P.LastName, P.PersonType, PP.PhoneNumber
FROM Person.Person AS P
INNER JOIN Person.PersonPhone AS PP
ON P.BusinessEntityID = PP.BusinessEntityID;
```

	FirstName	LastName	PersonType	PhoneNumber
1	Ken	Sánchez	EM	697-555-0142
2	Terri	Duffy	EM	819-555-0175
3	Roberto	Tamburello	EM	212-555-0187
4	Rob	Walters	EM	612-555-0100
5	Gail	Erickson	EM	849-555-0139
6	Jossef	Goldberg	EM	122-555-0189
7	Dylan	Miller	EM	181-555-0156

Bu tablo birleştirme işleminde **Person.Person** ve **Person.PersonPhone** tablolarında ortak sütun olan **BusinessEntityID** sütunları üzerinden birleştirme işlemi gerçekleştirerek, her iki sütunda da birbiri ile eşleşen kayıtları listeledik. İki sütunu da birleştirdiğimiz için, her iki sütundan da istediğimiz sütunları takma isimlerini kullanarak sonuç listemize ekledik.

JOIN işleminde bir ya da birden fazla tablo birleştirilebilir. SQL Server aynı anda 256 tabloyu **JOIN** ile birleştirebilir.

Genel kullanılan **JOIN** türü **INNER JOIN** olsa da, bazı sorgu örneklerinde, sadece **JOIN** komutunun kullanıldığını görmüş olabilirsiniz. **INNER** yazılmadan, sadece **JOIN** kullanılan birleştirme işlemlerinde SQL Server, **INNER JOIN** işlemi gerçekleştirir. Yani **JOIN** ile **INNER JOIN** aynı yöntem ile çalışır.

OUTER JOIN

Aynı anda her iki tabloda da yer almayan kayıtları listelemek için kullanılır.

OUTER JOIN kendi içerisinde farklı özellikler için kullanılan 3 yan özelliğe sahiptir. Bu kullanımlar ile iki tablodan biri ya da her ikisi üzerinde bu işlemi gerçekleştirebilir.

OUTER JOIN sorgusunda kullanılan ilk tablo soldaki tablo, ikinci tablo ise sağdaki tablo olarak kabul edilir.

OUTER JOIN Tipleri

- **LEFT:** Soldaki tabloda yer alan kayıtlar, sağdaki tabloda karşılıkları olmasa bile getirilirler.
- **RIGHT:** Sağdaki tabloda yer alan kayıtlar, soldaki tabloda karşılıkları olmasa bile getirilirler.
- **FULL :** Soldaki ve sağdaki tabloda karşılıklı olarak eşit satırı olmayan kayıtlar getirilir.

LEFT OUTER JOIN

Birleştirilen iki tablodan soldaki (birinci tablo) tabloda bulunan tüm kayıtlar getirilir. Sağdaki (ikinci tablo) tabloda ise soldaki tablo ile ilişkili ve eşleşen kayıtlar getirilir.

```
SELECT S.CountryRegionCode, S.StateProvinceCode,
       T.TaxType, T.TaxRate
FROM Person.StateProvince AS S
LEFT OUTER JOIN Sales.SalesTaxRate AS T
ON S.StateProvinceID = T.StateProvinceID;
```

	CountryRegionCode	StateProvinceCode	TaxType	TaxRate
1	CA	AB	1	14,00
2	CA	AB	2	7,00
3	US	AK	NULL	NULL
4	US	AL	NULL	NULL
5	US	AR	NULL	NULL
6	AS	AS	NULL	NULL
7	US	AZ	1	7,75

LEFT OUTER JOIN ile gerçekleştirdiğimiz bu sorguyu **INNER JOIN** ile hazırlasaydık, **TaxType** ve **TaxRate** sütunlarında **NULL** olan kayıtlar listelenmeyecekti.

Yukarıdaki ile aynı sorguyu, sadece **JOIN** kısmını değiştirerek **INNER JOIN**'e dönüştürebiliriz.

```
SELECT S.CountryRegionCode, S.StateProvinceCode,
      T.TaxType, T.TaxRate
FROM Person.StateProvince AS S
INNER JOIN Sales.SalesTaxRate AS T
ON S.StateProvinceID = T.StateProvinceID;
```

	CountryRegionCode	StateProvinceCode	TaxType	TaxRate
1	CA	AB	1	14,00
2	CA	ON	1	14,25
3	CA	QC	1	14,25
4	CA	AB	2	7,00
5	CA	ON	2	7,00
6	CA	QC	2	7,00
7	CA	BC	3	7,00

LEFT OUTER JOIN işlemlerinde **OUTER** anahtar kelimesini zorunlu değildir. Sadece **LEFT JOIN** yazarak da kullanılabilir.

RIGHT OUTER JOIN

Birleştirilen iki tablodan sağdaki (ikinci tablo) tabloda bulunan tüm kayıtlar getirilir. Soldaki (birinci tablo) tabloda ise sağdaki tablo ile ilişkili ve eşleşen kayıtlar getirilir.

LEFT JOIN işleminde kullandığımız sorgudaü sadece **LEFT** kısmını **RIGHT** ile değiştirerek çalıştıralım.

```
SELECT S.CountryRegionCode, S.StateProvinceCode,
      T.TaxType, T.TaxRate
FROM Person.StateProvince AS S
RIGHT OUTER JOIN Sales.SalesTaxRate AS T
ON S.StateProvinceID = T.StateProvinceID;
```

	CountryRegionCode	StateProvinceCode	TaxType	TaxRate
1	CA	AB	1	14,00
2	CA	ON	1	14,25
3	CA	QC	1	14,25
4	CA	AB	2	7,00
5	CA	ON	2	7,00
6	CA	QC	2	7,00
7	CA	BC	3	7,00

Sonuç olarak 29 kayıt listelenmiştir. Fark ettiyseniz **LEFT JOIN** işleminden sonra kullandığımız **INNER JOIN** sorgusu ile aynı sonucu getirmiştir.

FULL OUTER JOIN

İki tablo üzerinde hem **RIGHT JOIN** hem de **LEFT JOIN** işlemi gerçekleştirir.

İlk olarak soldaki tabloda bulunan tüm kayıtlar listelenir ve bu kayıtların sağdaki tabloda karşılıkları varsa, onlar listelenir ve karşılığı olmayanlar **NULL** olarak listelenir. Daha sonra sadece sağdaki tabloda olup, soldaki tabloda olmayan kayıtlarda listelenir.

```
SELECT S.CountryRegionCode, S.StateProvinceCode,
       T.TaxType, T.TaxRate
FROM Person.StateProvince AS S
FULL OUTER JOIN Sales.SalesTaxRate AS T
ON S.StateProvinceID = T.StateProvinceID;
```

	CountryRegionCode	StateProvinceCode	TaxType	TaxRate
1	CA	AB	1	14,00
2	CA	AB	2	7,00
3	US	AK	NULL	NULL
4	US	AL	NULL	NULL
5	US	AR	NULL	NULL
6	AS	AS	NULL	NULL
7	US	AZ	1	7,75

FULL OUTER JOIN kullanırken **OUTER** yazılmak zorunda değildir. Sadece **FULL JOIN** olarak kullanılabilir.

CROSS JOIN

İki tabloda yer alan kayıtları çaprazlamak için kullanılır. Bu işleme matematiksel olarak kartezyen çarpımı denir. İki tablo arasında herhangi bir ilişkilendirme olmasına gerek yoktur.

5 kayıt içeren **A** tablosu ile 3 kayıt içeren B isimli iki tabloda **CROSS JOIN** işlemi uygulandığında, sonuç listesi **5*3** kayıttan oluşacaktır.

```
SELECT p.BusinessEntityID, t.Name AS Territory
FROM Sales.SalesPerson p
CROSS JOIN Sales.SalesTerritory t
ORDER BY p.BusinessEntityID;
```

	BusinessEntityID	Territory
1	274	Northwest
2	274	Northeast
3	274	Central
4	274	Southwest
5	274	Southeast
6	274	Canada
7	274	France

Sorgu sonucunda 170 kayıt listelenecektir. **CROSS JOIN** işlemi uygulanan iki tablonun toplam kayıt sayılarına bakalım.

```
SELECT COUNT(BusinessEntityID) FROM Sales.SalesPerson;
```

	(No column name)
1	17

```
SELECT COUNT(TerritoryID) FROM Sales.SalesTerritory;
```

	(No column name)
1	10

```
17*10 = 170
```

CROSS JOIN, birinci tabloda yer alan her bir kaydı ikinci tabloda yer alan her bir kayıt ile ilişkilendirerek satırlar türetmede kullanılır. CROSS JOIN sıklıkla kullanılan bir JOIN işlemi değildir. Ancak bazı satır türetme isteklerinde kullanılabilir.