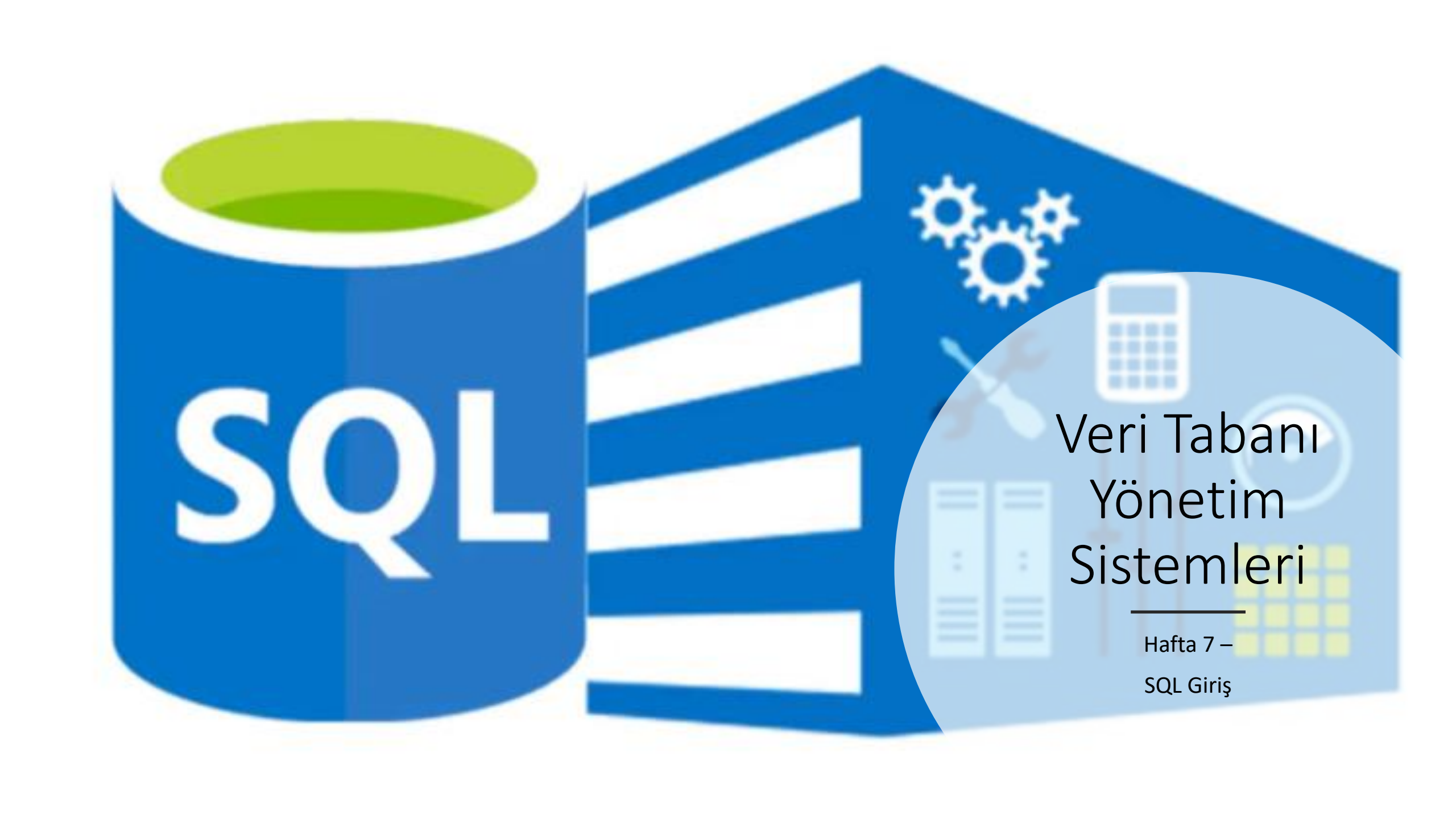


Veri Tabanı Yönetim Sistemleri

Hafta 7 –

SQL Giriş

Structured Query Language (SQL)

- Veri tabanı dili;
  - Veri tabanı ve tablo yapıları oluşturulmasına, temel veri yönetimi işlerinin gerçekleştirilmesine (ekleme, silme ve değiştirme) ve ham verileri yararlı bilgilere dönüştürmek için tasarlanmış karmaşık sorgular yazılmasına olanak tanır.
  - Minimum kullanıcı çabasıyla bu tür temel işlevleri yerine getirmelidir. Komut yapısı ve sözdiziminin öğrenilmesi kolay olmalıdır.
  - Taşınabilir olması gerekir; yani, bir kişinin bir RDBMS'den diğerine geçerken temel bilgileri yeniden öğrenmesine gerek kalmaması için bazı temel standartlara uyması gerekir.
- SQL, bu ideal veritabanı dili gereksinimlerini iyi bir şekilde karşılar.

SQL işlevleri iki geniş kategoriye ayrılır:

- ***Veri Tanımlama Dili (data definition language -DDL):*** SQL; tablolar, indeksler ve view gibi veri tabanı nesneleri oluşturmaya yönelik komutların yanı sıra bu veritabanı nesnelerine erişim haklarını tanımlama komutlarını içerir.

- ***Veri İşleme Dili(data manipulation language -DML).*** SQL; veri tabanı tabloları içinde veri eklemek, güncellemek, silmek ve almak için komutlar içerir.

TABLE 7.1

SQL DATA DEFINITION COMMANDS

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Validates data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows and columns from one or more tables (see Chapter 8, Advanced SQL)
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

TABLE 7.2

SQL DATA MANIPULATION COMMANDS

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to its original values
Comparison operators	
=, <, >, <=, >=, <>, !=	Used in conditional expressions
Logical operators	
AND/OR/NOT	Used in conditional expressions
Special operators	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values
Aggregate functions	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

SELECT

- *AdventureWork2019 örnek veri tabanı kullanılacaktır.

- ***Q: İnsan Kaynakları Departmanındaki Personelin NationalIDNumber, LoginID ve JobTitle alanlarını seçelim.***

- ***Q: İnsan Kaynakları Departmanındaki tüm alanları seçelim.***

- **NOT:** Çalıştığımız veri tabanını değiştirmek için Use ifadesiyle birlikte sorgulama yapacağımız veri tabanının adını girmemiz gerekir.
- **NOT:** Noktadan önceki ilk kısım Schema(Şema) adını belirtirken noktadan sonraki kısım ise tablonun adını belirtmektedir. Schema kavramı SQL 2005 ile birlikte gelmiş bir kavram olup mantıksal olarak veri tabanı nesnelerini gruplamak için kullanılır. Nesnelerin gruplandırılması yönetim açısından büyük kolaylık sağlar. Örneğin şirketinizde muhasebe işlemlerinde kullanılan tüm nesneleri gruplandırmak için Muhasebe adında bir Şema oluşturabilir ve muhasebe nesneleri ile ilgili tek tek yetki vermek yerine Muhasebe şeması için yetkilendirme yapabilirsiniz.
- **NOT:** Kolay kullanımı olmasına rağmen * yerine kolon isimleri kullanılması önerilir. Eğer tablolarda bir değişiklik yapılırsa (örneğin iki kolon arasına yeni bir kolon) sıralı erişim yapan uygulamalarda sorun çıkabilir. Ayrıca sorgu performansını düşürecektir. SQL Server'in gereğinden fazla veri okumasına sebep olup okuma performansını düşürecektir. Bununla birlikte sonuçlar bir Client'a gönderilecekse Server-Client arasındaki network trafiğini de arttıracaktır.

AS

- ***Alias Kavramı:** sorgu sonucu oluşan tablodaki sütun isimleri veri kaynağından farklı olarak gösterilmek istenebilir. Bu şekilde oluşturulan yeni isimlere «Alias» denir.

- ***Q: İnsan kaynakları departmanındaki personel bilgilerinin PersonelNo(BusinessEntityID), KullanıcıAdı(LoginID), Doğum Tarihi(BirthDate), Cinsiyet (Gender) ve Medeni Durumu(MaritalStatus) başlıkları halinde getirilmesini sağlayın.***

- **NOT:** AS operatörünün zorunlu olmadığı görülse de sorgunun anlaşılabilir olması açısından kullanılması önerilir.

- **NOT:** aslias isimlerinde özel ifade, boşluk gibi özel karakterler kullanmak istiyorsak [] veya ‘ ’ kullanabiliriz. Hepsinden farklı olarak önce Alias isminin verildiği son sorguya dikkat ediniz.

- **NOT:** Bu kadar farklı kullanıma rağmen hepsinin bilinip bir tanesinin alışkanlık haline getirilmesi faydalı olacaktır.

WHERE

- Sorgu sonuçlarının koşul ile sınırlandırılmasını sağlar.

```
SELECT select_listesi
FROM tablo_listesi
[WHERE arama şartı]
```

  - Where kullanılırken tablodaki kayıtlar TRUE(koşula uygun), FALSE(koşula uygun değil) veya UNKNOWN(erişilen veri NULL) olarak değerlendirilirler.
  - NULL değer boşluk veya 0 gibi değerler olarak **karıştırılmamalıdır.** UNKNOWN olarak değerlendirilir ve bilinmezlik ifade ederler. NULL olan iki içerik eşit mi değil mi diye karşılaştırılamaz.

Q: Kişiler tablosundan BusinessEntityID değeri 5 olan kişinin ID, Title, First Name ve Last Name bilgilerini getirelim.

WHERE

Q: Soy ismi 'Adams' olan kişilerin BusinessEntityID, FirstName, LastName, MiddleName bilgilerini getirelim.

- *tek tırnağı kaldırıp çalıştıralım!

Tek tırnak kullanmadığımız metinsel ifadeler SQL Server tarafından tablo veya kolon ismi gibi SQL Server nesneleri olarak algılanır.

```
Msg 207, Level 16, State 1, Line 3
Invalid column name 'Adams'.
```

AND – OR - NOT

- ***Q: Title Alanı 'Mr.' Olan ve FirstName değeri 'Robert' olan kişilerin BusinessEntityID, Title, FirstName, LastName, MiddleName alanlarını listeleyelim.***
- ***Q: Title Alanı 'Mr.' Olan veya FirstName değeri 'Robert' olan kişilerin BusinessEntityID, Title, FirstName, LastName, MiddleName alanlarını listeleyelim.***
- ***Q: Title bilgisi 'Mr.' olmayan kişilerin BusinessEntityID, Title, FirstName, LastName, MiddleName alanlarını listeleyelim.***

- NOT: Where ifadesi ile kullanılan koşul operatörleri arasında Öncelikle parantez içerisindeki işlemler, ardından NOT operatörü, ardından varsa AND ve son olarak da OR işlemi görülür.

- Sonuçlar sizce ne olur?

```
Select Title,FirstName,LastName,MiddleName
FROM Person.Person
WHERE Title = 'Ms.' AND FirstName = 'Catherine' OR LastName = 'Adams'
```

```
Select Title,FirstName,LastName,MiddleName
FROM Person.Person
WHERE Title = 'Ms.' AND (FirstName = 'Catherine' OR LastName = 'Adams')
```

!= <>	**İki ifadenin eşit olup olmadığını kontrol eder**
<	Değer küçük mü kontrol eder
<=	Değer küçük veya eşit mi kontrol eder
=	İki ifade arasındaki eşitliği kontrol eder
>	Değer büyük mü kontrol eder
>=	Değer büyük veya eşit mi kontrol eder
ALL	Karşılaştırma operatörü veya alt sorgu ile kullanıldığı zaman alınan değerlerin tümü arama koşulunu sağlayan satırları alır
BETWEEN	Dahil olan bir aralığı belirtir. AND ile birlikte kullanılır.
CONTAINS	Sözcükler ve ifadeler için belirsiz bir arama yapar
EXIST	Alt sorgu ile kullanıldığı zaman satır dönüyor mu kontrol eder
IN	Arama koşulunu sağlayan değerleri içeren liste sağlar
IS NOT NULL	Değerin boş olmadığını kontrol eder
NULL	Değer boş mu kontrol eder
LIKE	Verilen örnek değerle eşleşmeyi kontrol eder
NOT BETWEEN	Verilen aralıkta yer almayan değerler
NOT IN	Satır döndürmeyen değerlerin listesi
NOT LIKE	verilen örnekle eşleşmeyen değerler

- ***Liste fiyatı 200 ile 1000 arasında olan veya rengi Silver olan ürünlerin ProductID, Name, Color, ListPrice alanlarını listeleyelim.***

```
SELECT ProductID,Name,Color,ListPrice
 FROM Production.Product
 WHERE Color='Silver' OR (ListPrice > 200 AND ListPrice <=1000 )
```

LIKE

- ***İsmi B ile başlayan ürünlerin ProductID,Name bilgilerini listeleyelim.***

%	**Sıfır veya daha fazla karakter seti**
_	Sadece bir karakter
[]	Özel bir karakter
[^]	Özel bir karakterin olmadığı

LIKE 'AD%'	**AD ile başlayan tüm kayıtlar**
LIKE '%sin'	'sin' ile biten tüm kayıtlar
LIKE '%in%'	'in' ifadesini içeren tüm kayıtlar
LIKE '_en'	'en' ile biten tüm 3 karakterli kayıtlar
LIKE '[LS]%'	'L' veya 'S' ile başlayan tüm kayıtlar
LIKE '[A-D]nan'	Tüm 4 karakterli ve ilk karakteri 'A' ile 'D' arasında olan ve 'nan' ile biten tüm kayıtlar
LIKE 'M[^c]%'	'M' ile başlayan ve ikinci karakteri 'c' olmayan tüm kayıtlar

LIKE

- ***Q: İkinci harfi j olan kişilerin BusinessEntityID, FirstName, LastName alanlarını listeleyelim.***

- ***Q: İlk harfi a,b,c veya d olan kişilerin BusinessEntityID, FirstName, LastName alanlarını listeleyelim.***

- ***Yada?***
- ***Yada?***

- ***Q: Joker karakterlerden biri aranmak isteniyorsa?***

1.yol Escape karakteri kullanmak
Select * from dbo.TestTablo where aciklama like '%/%%' escape '/'
2.yol köşeli parantezi kullanmak:
Select * from dbo.testTablo where aciklama like '%[%]%'

- ***Q: İsmi B ile başlamayan ürünlerin ProductID, Name alanlarını listeleyiniz.***

Between.. And...

- Önce küçük sonra büyük değer yazılmalıdır.
- Her iki değerde koşula dahildir.

- ***Q: VacationHours değeri 15-25 arası olan İnsan Kaynakları personelinin BusinessEntityID, JobTitle, VacationHours alanlarını listeleyiniz.***

- **Q: 15-25 arası olmayanları listeleyiniz.**

IN

- Bir sütunun birden fazla değer alabileceği durumlarda kullanılmaktadır.
- OR ile aynı işlevde ama kullanımı daha esnek ve kolay

- ***Q: ProductID alanı 320,321,322 olan ürünlerin bilgilerini listeleyiniz.***

- ***Q: 320,321,322 olmayan ürünleri listeleyiniz.***
- (Aynı şekilde NOT IN yapısı mevcut…)

- Tersi bir kullanım da söz konusudur yani bir değeri içeren kolonları bulma:

```
SELECT *
FROM Production.Product
WHERE 'Chainring' in (Name,Color)
```

IS NULL – ISNULL() – COALESCE()

- ***Color değeri Null olan ürünlerin ProductID, Name ve Color alanlarını listeleyelim.***

(Özellikle raporlamalarda NULL bilgisi görülmesi hoş karşılanmayabilir.)

- ISNULL() 2 parametre alır. İlki NULL olma durumunu kontrol eder ve eğer NULL ise ikinci parametreyi döndürür.

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

- COALESCE() birden fazla parametre olarak ifade alır ve NULL olmayan ilk ifadeyi döndürür. ISNULL() dan en büyük farkı aynı anda birden fazla ifadeyi karşılaştırmasıdır.

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

NOT: COALESCE bir ANSI standardı olup ORACLE, MySQL gibi diğer veri tabanları yönetim sistemlerince desteklenmektedir. ISNULL() MS SQL SERVER'da geçerlidir.

Order BY

- Sorgulama sonuçlarının belirttiğimiz şekilde sıralanmasını sağlar.
- ***Ürünleri Ad,Ürün numarası ve yeniden sipariş alanlarını ad kolonuna göre sıralayalım***

- ***Ürünleri Ad,Ürün numarası ve yeniden sipariş alanlarını öncelikle yeniden sipariş ardından ad kolonuna göre sıralayalım***

- ***Ürünleri Ad,Ürün numarası ve yeniden sipariş alanlarını ad kolonuna göre küçükten büyüğe doğru sıralayalım***

- ***Ürünleri Ad,Ürün numarası ve yeniden sipariş alanlarını ad kolonuna büyükten küçüğe doğru sıralayalım***

- ***Her defasında farklı rastgele sıralanmış sonuçlar üretelim***

- Order by kullanmazsak sql verilere en hızlı şekilde nasıl erişiyorsa o şekilde bize gösterecektir.

Diğer bir ifadeyle fiziksel sıralamaya göre sıralanacaktır.

- ASC: küçükten büyüğe doğru sıralanacaktır (varsayılan)
- Bazı durumlarda bize rastgele sırada kayıt dönmesini isteyebiliriz.

Bu gibi durumlarda bir sistem fonksiyonu olan New_ID() kullanabiliriz.

TOP

- TOP ifadesi sorgu sonucu oluşan kayıt setinin sadece bir kısmını görüntülemeyi sağlar.
  - TOP(expression) PERCENT WITH TIES

  - Expression: numerik bir ifadedir. Kayıt setinin ne kadarlık diliminin gösterileceğini belirtir.
  - Percent: belli bir yüzdelik dilimin gösterilmesini sağlar.

- ***Ürün tablosundan 5 adet ürünün ProductID, Name, Color, ListPrice özelliklerini listeleyelim.***
- ***Tüm kayıtların yüzde 2 lik kısmını ekranda lisleteylim.***
- ***En pahalı 10 ürünün yukarıdaki özelliklerini listeleyelim.***
- ***En pahalı ürünün yukarıdaki özelliklerini listeleylim***
- ***En pahalı olan tüm ürünlerin özelliklerini listleylim!***

GROUP BY

- Bir grup veriyi belli bir özelliğe göre istatistiksel analiz yapmak için gruplar.
- Örneğin en yüksek maaşı alan personeli bulmak için built-in fonksiyon olan MAX fonksiyonu kullanılabilir. Ancak her departmanda en yüksek maaşı alan personeli bulamak için personel tablosunu departmanlara göre gruplamamız gerekir. Bir veri grubunu alt gruplara ayırmak için GROUP BY operatörü kullanılır.
- GROUP BY operatörü sıklıkla SQL Server Aggregate fonksiyonları ile birlikte kullanılır.

FONKSİYON	VERİ TİPİ	AÇIKLAMA
SUM()	Numerik	Sütundaki NULL olmayan değerlerin tamamını toplar
AVG()	Numerik	Sütundaki NULL olmayan tümünün ortalamasını alır.
MIN()	Numerik,metin,tarih	Sütundaki en küçük sayıyı,tarihi veya metni döndürür.
MAX()	Numerik,metin,tarih	Sütundaki en büyük sayıyı,tarihi veya metni döndürür.
COUNT()	Satır tabanlı herhangi bir veri tipi	Tablodaki NULL olmayan değerlerin sayısını döndürür.

GROUP BY

- ***Ürünler tablosundaki toplam kayıt sayısını, toplam fiyat ve ortalama fiyatı gösterelim.***
- Aggregate fonksiyonlarda aksi belirtilmedikçe NULL değerler göz ardı edilir. Bu yönde bir uyarı mesajı alınabilir(NULL değerler içeriyor ise)

```
Select COUNT(*) CntToplam, COUNT(ProductID) CntProductID, COUNT(Color) CntColor, COUNT(DISTINCT Color) CntColorDist
FROM Production.Product
```

  - CntToplam: herhangi bir kolonun değerini NULL olup olmadığını kontrol etmeden getirir.
  - CntProductID: ProductID değeri NULL olmayan değerleri sayarak getirir.
  - CntColor: Color değeri NULL olmayan değerleri sayarak getirir.
  - CntColorDist: Color alanındaki NULL olmayan ve birbirinden farklı değerleri sayarak getirir.

- AVG(Fiyat) = SUM(Fiyat)/Count(*) her zaman eşit olmayabilir! SUM ile dikkate alınmayan bazı satırlar Count ile dikkate alınacaktır !
- ***Her bir üründen kaçar tane satıldığını satılan ürün sayısına göre azalan sırada listeleyelim***
- Sadece bir kolona göre gruplamak zorunda değiliz.
- ***Her bir üründen hangi şehirde kaçar tane satıldığını satılan ürün sayısına göre azalan sırada listeleyelim***

HAVING

- Group by ifadesiyle gruplanan sorgular üzerinde koşul yazmamızı sağlayan diğer bir yardımcı operatördür.

Where ifadesi ile belirtilen koşullar, veriye fiziksel olarak erişilirken kontrol edilir. Örneğin ProductID<710 gibi. Fiziksel olarak veriyi okurken değeri 710 dan küçük olan satırları okur.

Having ise fiziksel olarak okunmuş ve gruplanmış veriler üzerinde koşul uygulanır. Örneğin toplam satılan ürün sayısı 500 den büyük olanları listelemek. Önce tüm kayıtlar okunup hangi üründen kaçar tane satıldığı bulunması gerekir ardından gruplanan veri üzerinde şarta uyanlar listelenir.

- **Toplam satılan ürün adedi 500 den fazla olan ürünlerin ürünID ve toplam satışını listeleyelim.**

- Aggregate fonksiyonu Where ile yazarsak sonuç ne olur?

- Having ile Aggregate olmayan bir koşul yazılabilir. Ama bu durum tavsiye edilmez çünkü veri fiziksel olarak okunup gruplanır daha sonra koşul uygulanır. Aynı işlemi Where ile yapalım?

```
Select ProductID, COUNT(ProductID) [Toplam Satılan Ürün]
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING ProductID=708
```

Case When

- Sorgularımızdaki Kolonların değerine göre farklı işlemler yapmamıza olanak sağlar.
- CASE
  - WHEN Boolean expression THEN result expression [...n]
  - ELSE else_result expression]

  END

Ürünlerimizin Ürün ID, Ürün Numarası, Adı, Ürün Hattı, Kategori olarak listeleyelim. Ürün Hattı R olanların Kategori alanında Road, M olanların Mountain, T olanların Touring, S olanların Other sale items ve bunların dışında bir değer ise Not for sale yazmalıdır. Ayrıca ÜrünNumarasına göre sıralayalım.

Ürünlerimizin ÜrünID, Adı, ListeFiyatı ve Fiyat Açıklaması olarak listeleyelim. Liste fiyatı 100 den küçük olanlarda Ucuz,100-500 arası olanlarda orta, üzeri olanlarda pahalı yazmalıdır.

DISTINCT - ALL

- Sorgu sonucunda oluşan tekrar eden verileri engeller. Belirtilen ifadelerin tekrar edilmemesini sağlar.
- ***20 den fazla sipariş verilmiş ürünleri listeleyelim.***
- ***Aynı ürün adı tekrar etmeyecek şekilde listeleyelim.***
- ***Birden fazla kolonda listelenebilir. Her bir kolon bilgisi aynı olan kayıtlar bir defa listelenecektir.***
- ***Toplam satış adedimiz ile kaç adet farklı satış yaptığımızı gösterelim***

- ALL ise DISTINCT tersi olarak tüm satırların listelenmesini sağlar. Select ifadelerinde varsayılan olandır.

UNION - UNION ALL

- Bazı durumlarda 2 veya daha fazla sonuç seti birleştirilmek istenebilir. Özellikle büyük firmalarda sürekli erişilen veriye hızlı ulaşmak için nadir erişilen veriler (arşiv vb.) farklı yerde tutulur. Örneğin Siparişler ile Teslim Edilmiş Siparişler vb.

- ***Mevcut ve geçmiş tüm satış görevlisi tablolarından satış kotası 0 dan büyük olanların satış görevlisi ID, satış kotası ve düzenlenme tarihi bilgilerini ID değerine göre ve kota bilgilerine göre azalan sırada listeleyelim***.

- Aslında sonuç setlerini birleştirirken ilk sorguya distinct ekler. 2 tabloda da aynı olan kayıtlar listelenmez. Ekleme yapılırken verilerin ilk sonuç setinde olup olmadığı kontrol edilir.

- UNION-UNION ALL arasındaki fark DISTINC –ALL arasındaki fark gibidir. UNION ALL kullanılırsa ikinci sonuç seti kontrol yapılmaksızın ilkinin sonuna eklenir.

- Tekrar eden kayıt olmadığına eminseniz veya tekrar durumu önemli değilse çok miktarda veri içeren sonuç setleri için her defasında tekrarlı kayıt olup olmadığını kontrol etmeyerek performans kazanmak adına UNION ALL kullanılabilir.

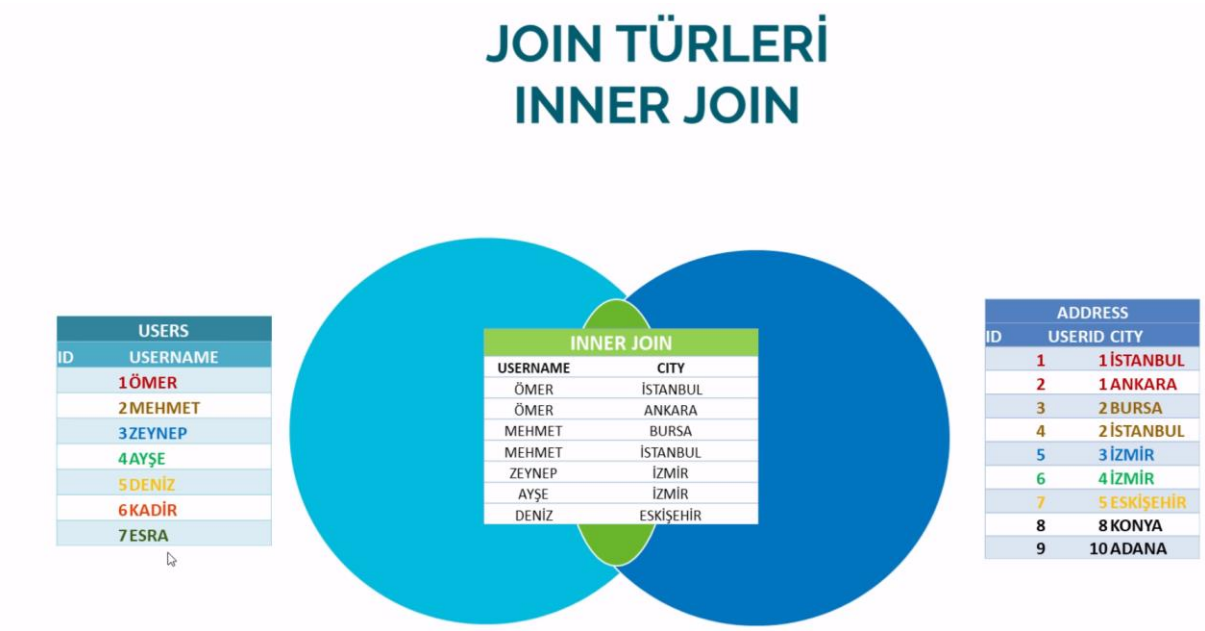
BIRDEN FAZLA TABLONUN BERABER SORGULANMASI

- İlişkisel veri tabanlarının en önemli özelliklerinin başında verilerin büyük bir tabloda tutulması yerine normalize edilerek birkaç tabloda tutulması gelir. Normalizasyonun temel amaçları arasında başlıca olarak tekrar eden kayıtları en aza indirgemek, verilerin kapladığı alanı azaltmak ve performansı arttırmak sayılabilir.
  - Örneğin Siparişler tablosunu düşünürsek bir müşteri 100 sipariş vermişse 100 defa adı, soyadı, adresi bilgisi girilmelidir. Ad işin 50, Soyad için 50, Adres için 200 karakter düşünülürse bir müşteri için 300 * 100 byte alan gerekecek ayrıca aynı bilgiler de tekrar edecektir. Bunun yerine müşteriler diye bir tablo açmak ve müşteri ID ile bu tabloları ilişkilendirmek temel normalizasyon yaklaşımıdır. (SiparisYeni,Musteriler)
  - Başka bir avantajı ise müşteriye ait yeni bilgileri rahatlıkla ekleyebiliriz.

- 2 tablonun böylece birleştirilip tek tablo sorgulanması JOIN işlemi olarak bilinmektedir. En çok kullanılan JOIN çeşitleri:
  - Inner JOIN
  - Outer JOIN
  - Cross JOIN

INNER JOIN

- En çok kullanılan JOIN türü.
- Her tabloda diğer tablodaki kayıtları referans eden bir alan bulunmalıdır.
- Bire bir eşleştirme yapar.
- Kümelerle ifade edilecek olursa iki kümenin kesişimine eşittir. Yani her iki tabloda olup, verilen koşula uyan kayıtlar listelenir.
  - Select <select list> // görüntülemek istediğimiz kolonlar
  - From <first table> // ilk tablo
  - <join type> <second table> // join tipi
  - [ON <join condition] // her iki tablodaki ortak bulunan değerler üzerindeki koşul



INNER JOIN

- `SELECT *`

`FROM ORDERS inner join USERS on ORDERS.USERID = USERS.ID`

*UserID kolonunun 1 defa listelenmesi için * yerine kolon isimleri yazılmalıdır.

- Aşağıdaki Hatanın sebebi ne olabilir?

```
Select Person.BusinessEntity.*,BusinessEntityID
From Person.BusinessEntity
INNER JOIN HumanResources.Employee
ON Person.BusinessEntity.BusinessEntityID = HumanResources.Employee.BusinessEntityID
```

INNER JOIN

- ***Person.BusinessEntity tablosundaki kişilerin tüm bilgileriyle aynı kişilerin HumanResources.Employee tablosundaki LoginID bilgilerin listeleyelim?***

- LoginID kolonu BusinessID olarak değiştirildiğindeki hatanın sebebi sizce ne olabilir?

```
Msg 209, Level 16, State 1, Line 1
Ambiguous column name 'BusinessEntityID'.
```

- ***Alias kullanarak gerçekleştirelim.***

- Alias tanımlandığı halde kullanılmaması hataya sebep olacaktır.

INNER JOIN

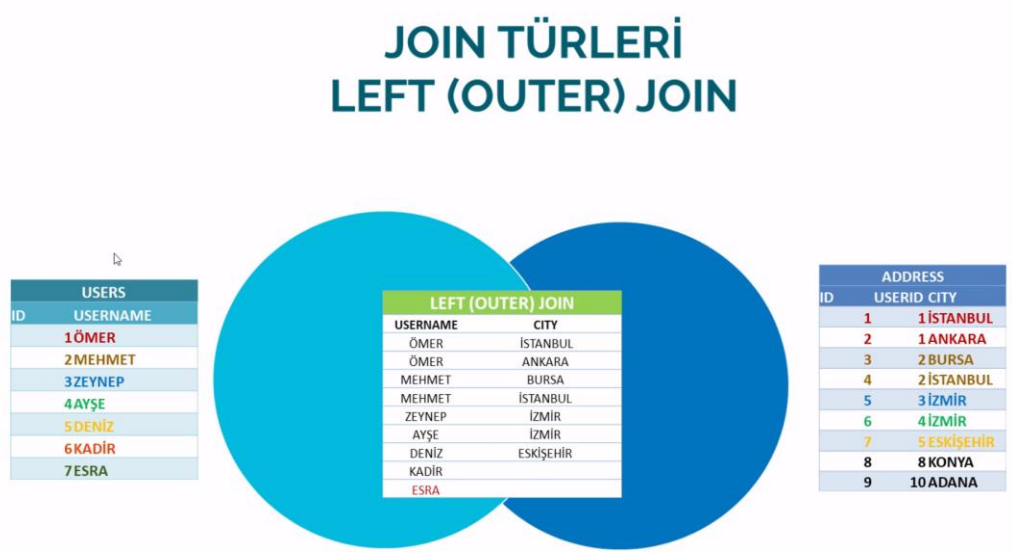
- ***İş tanımı : Engineering Manager olan kişilerin business entityID, JobTitle, FirstName ve LastName bilgilerini getirelim.***

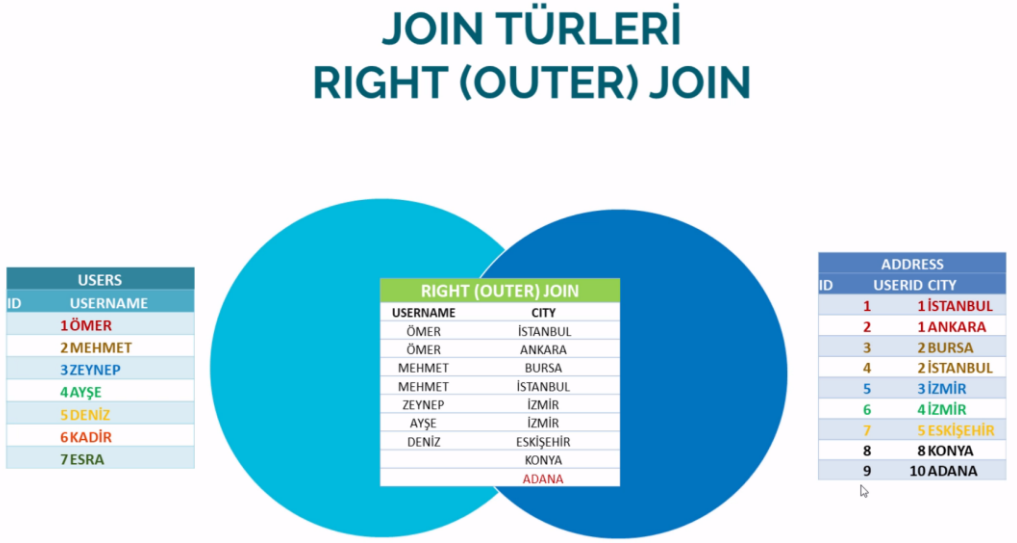
- Sadece JOIN yazmakta geçerlidir.

OUTER JOIN

- Her iki tabloda da olma koşulu gerekmediğinde kullanılır.
- Left Outer Join, Right Outer Join, Full Outer Join olmak üzere 3 e ayrılır.

  - Select <Select listesi>
  - From< The table you want to be the «Left»table
  - <LEFT|RIGHT> [OUTER] JOIN <table you want to be the «RIGHT»table
  - ON <Join condition>





LEFT OUTER JOIN

- Outer çeşitleri içerisinde en sık kullanılandır.
- Referans olarak seçilen tablodaki kayıtların tümü listelenir ve eğer diğer tabloda birleştirme şartına uyan karşılığı varsa onlar listelenir yoksa NULL olarak gösterilir.
- ***Person.StateProvince tablosunda CountryRegionCode, StateProvinceCode alanlarıyla Sales.SalesTaxRate tablosundan TaxType ve Tax Rate alanlarını listeleyelim.***

(29 kayıt listelendi)

- ***Vergi karşılığı olmayan ülke ve bölgelerde listelensin!***

(184 kayıt listelendi)

Sol taraftaki tablodaki (Person.StateProvince) kayıtlar referans alındı ve bunlara karşılık gelen değerler listelendi. Karşılığı olmayanlar ilse NULL olarak gösterildi.

Outer yazımı zorunluluk değildir. Left Join olarak da yazılabilir.

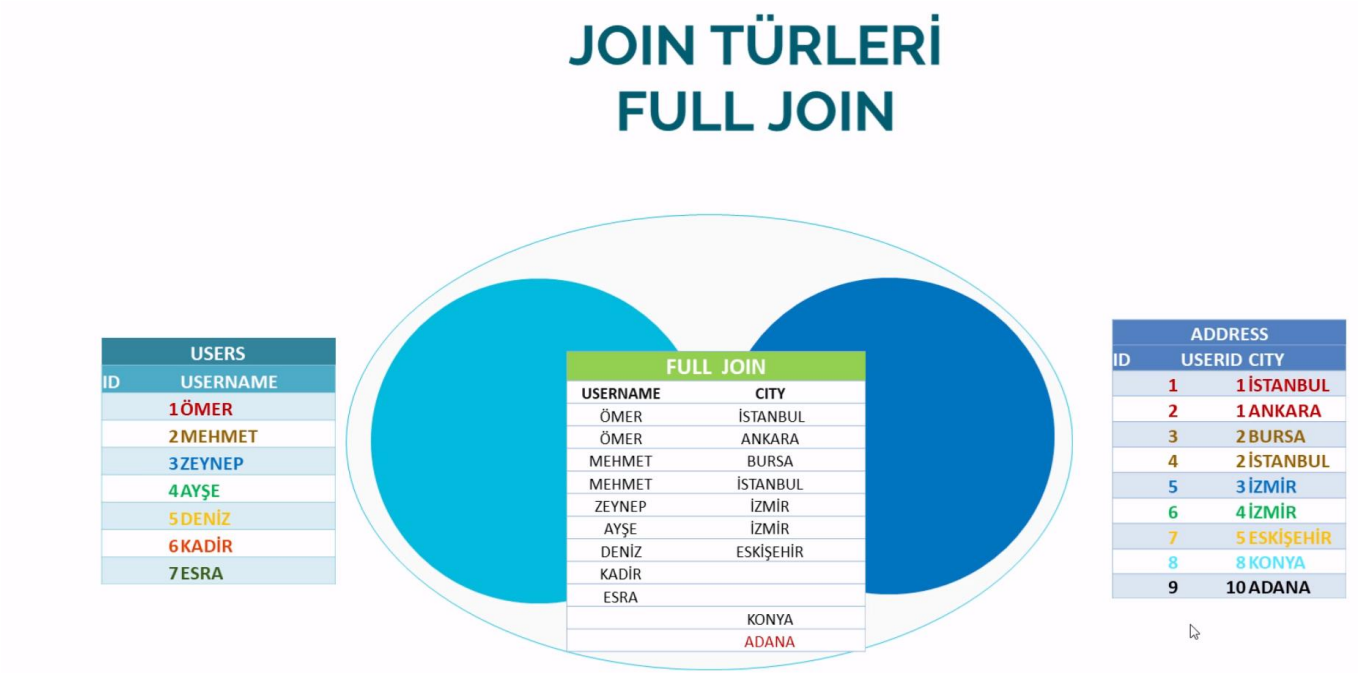
RIGHT OUTER JOIN

- Sağ tarafa yazılan tablo referans alınır. Sol taraftaki tabloda koşul karşılığı alanlar varsa listelenir yoksa NULL olarak gösterilir.

- ***Yukarıdaki örnekleri RIGHT OUTER JOIN olarak listeleyelim.***

- Left Join işlemi yapılan iki tablo sorgular aynı kalmak koşuluyla tablo yerleri değiştirilerek Right Join işlemine dönüştürülebilir.
- ***Yukarıdaki örnekler üzerinde deneyerek görelim!***

FULL OUTER JOIN



- Her iki tabloyu da referans olarak alır. Yani her iki tabloyu hem Left Join hem Right Join ile birleştirmiş gibi davranır. Daha açık bir ifadeyle sol taraftaki tablo referans alınarak listelenir karşılığı olmayanlar Null olarak gösterilir. Ardından sağ tabloda olup sol tabloda olmayan kayıtlar da listelenerek Full Join elde edilir.
- SQL Server 2000 ve öncesinde Left Join için (*=) , Right Join için (=*) kullanılabiliyordu. Bu özellik 2005 ve sonrası için desteklenmemekte.

CROSS JOIN

- Bu join işleminde referans alınan tablolar arasında herhangi bir bağlantı aranmaz. Yani ON anahtar sözcüğüne gerek yoktur. Cross Join bir koşul belirtilmedikçe her iki tablodan birini referans alır ve her bir kayıt için diğer tablodaki kayıt sayısı kadar kayıt listelenir. (Matematikte Kartezyen Çarpım gibi düşünülebilir)

- ***Person.StateProvince tablomuzla Sales.SalesTaxrate tablolarını StateProvinceID, CountryregionCode, TaxType ve TaxRate alanları üzerinden Cross Join yapalım***

Referanslar

- «SQL Server 2017» Abaküs yayınları – İsmail Adar
- Carlos Coronel, Steven Morris, and Peter Rob, Database Systems: Design, Implementation, and Management, Cengage Learning.
- BTK AKADEMİ – «Uygulamalarla SQL Öğreniyorum» - Ömer Çolakoğlu

- Resim:
  - https://teknoloji.org/sql-nedir-sql-nerelerde-kullanilir/