Sütunların Birleştirilmesi

İki yada daha fazla sütunu tek bir sütunda göstermek

(Ör: Ad Soyad) için kullanılır.

Oracle veritabanlarında birleştirme işlemi "||" karakteri ile

MS Sql Server veritabanında "+" karakteri ile ve

MySql veritabanında ise CONCAT fonksiyonu kullanılır.

Veri Tipleri:

Ms Sql Server Kullanımı

Select per_id,(ad +" "+soyad) AS adısoyadı,sehir From Personel

Oracle Kullanımı

Select per_id,(ad ||" "||soyad) AS adısoyadı,sehir From Personel

MySql Kullanımı

Select per_id,CONCAT(ad ," ",soyad) AS adısoyadı,sehir From Personel

Alanlar:Gösterilecek alanlar bu bölüme yazılır ve birden fazla alan eklencekse virgül ile ayrılır.*(yıldız) kullanıldıgında ise tablodaki tüm alanları gösterir.

table_adi:Seçim yapılacak tablonun adı yazılır.

Where:Koşullu kayıtların gösterilmesi için kullanılır.Koşul birden fazla ise AND veya OR deyimleri ile birbirine bağlanır.

Ayrıca Where komutu ile birlikte kullanılan LİKE ve IN ifadeleride vardır.

SQL KOMUTLARI

Select Komutu:

Select (Sutün İsmi) From (Tablo İsmi) şeklinde Kullanılır.

Select * from Personel =Personel Tablosundaki Tüm Kayıtları listeler. (*) İsareti tüm Sutundaki kayıtları getirir.

Select Komutu MS SQL Server,MySQL,Oracle veritabanları için geçerlidir.

İnsert Komutu:

1.Kullanım: INSERT INTO tablo VALUES (deger1,deger2,deger3,...)

2.Kullanım: INSERT INTO tableadı (sutun1,sutun2,sutun3,...) Values (deger1,deger2,deger3,...)

TABLOLARIN YEDEĞİNİ ALMA

```
-- Tabloyu yedeklemek için into yazıp NewEmp yazılır.
select EmployeeID,
FirstName,
BirthDate
into NewEmp from Employees

select * from NewEmp
```

Update Sorgusu ile Kayıt Güncelleme

Kullanış Şekli:

Update TabloAdi

SET alan1=değer1, alan2=deger 2 …

Where Koşulumuz

Örn:

Update urunler Set fiyati=500 where urunId=311

Where Kullanımı:Koşullu kayıtların gösterilmesi için kullanılır.Koşul birden fazla ise AND veya OR deyimleri ile birbirine bağlanır.AND komutu tüm koşulların sağlanması durumunda çalışır. OR komutu ise koşulların bir tanesi çalışması durumunda sorgu çalıştırır.

Ayrıca Where komutu ile birlikte kullanılan LİKE ve IN ifadeleride vardır.

Örnekler:

Select * From personel Where ad='Seref' bu dizin personel tablosundaki adı seref olan tüm kişileri listeler.

```
--TEKRARLI KAYITLARI BULMA KODU
SELECT PERSONID FROM PERSON GROUP BY PERSONID
HAVING COUNT(*) > 1
```

And ve Or Kullanımı:

a)AND komutu tüm koşulların sağlanması durumunda çalışır.

Select *From personel Where görevi='Mühendis' and sehir='Ordu' bu dizin personel tablosundaki görevi mühendis ve sehiri ordu olan kayıtları listeler.

b)OR komutu ise koşulların bir tanesi çalışması durumunda sorgu çalıştırır.

Select *From personel Where görevi='Mühendis' OR sehir='Ordu' bu dizin personel tablosundaki görevi mühendis veya sehiri ordu olan kayıtları listeler.

AND VE OR 2.TANIM VE ÖRNEKLER

TANIM:

AND operatörü

Birinci durumla beraber ikinci durumunda olduğu zaman kullanılır.

OR operatörü ise birinci durum veya ikinci durumun gerçekleşmesi durumunda kullanılır.

```
SELECT  alan_adi1,alan_adi2
FROM tablo_adi
WHERE
alan_adi1=sorgu_degeri AND alan_adi2=sorgu_degeri
```

```
SELECT  alan_adi1,alan_adi2
FROM tablo_adi
WHERE alan_adi1=sorgu_degeri OR alan_adi2=sorgu_degeri
```

id	Adi_soyadi	Sehir	Bolum	Meslek_Kodu
1	Salih ESKİOĞLU	İstanbul	Bilgi İşlem Sorumlusu	1234567
2	Ayhan ÇETİNKAYA	Kocaeli	İdari İşler Yöneticisi	2345678
3	Serkan ÖZGÜREL	Erzincan	Muhasebe	3456789
4	İlhan ÖZLÜ	İstanbul	BİIgi İşlem Sorumlusu	2345678

AND

ÖRN1:

SELECT * FROM Personel WHERE Sehir='İstanbul' AND Bolum='Bilgi İşlem Sorumlusu'

id	Adi_soyadi	Sehir	Bolum	Meslek_Kodu
1	Salih ESKİOĞLU	İstanbul	Bilgi İşlem Sorumlusu	1234567
4	İlhan ÖZLÜ	İstanbul	BİIgi İşlem Sorumlusu	2345678

Açıklama=Şehir ismi= İstanbul ve

Bölümü= 'Bilgi işlem Sorumlusu' olan kayıtları getirir.

OR

ÖRN1=

id	Adi_soyadi	Sehir	Bolum	Meslek_Kodu
1	Salih ESKİOĞLU	İstanbul	Bilgi İşlem Sorumlusu	1234567
2	Ayhan ÇETİNKAYA	Kocaeli	İdari İşler Yöneticisi	2345678
4	İlhan ÖZLÜ	İstanbul	BİIgi İşlem Sorumlusu	2345678

Açıklama=Şehir ismi İstanbul ve Kocaeli olan kayıtları ayrı ayrı getirdi.

IN Kullanımı:

a)Bu dizin personel tablosundaki şehir sütunun içindeki Ordu ve İzmir olan kayıtları getirir.

-Select *From personel Where sehir IN('Ordu','İzmir')

b)Bu dizin personel tablosundaki sehiri ordu veya izmir olan kayıtları listeler.

Select *From personel Where sehir IN('Ordu','İzmir')

NOT IN KULLANIMI:

Select *From personel Where sehir NOT IN('Ordu','İzmir')

Bu dizin personel tablosundaki şehir sütunun içindeki Ordu ve İzmir haricindeki tüm şehir kayıtlarını getirir.

UNİON(Birleştirmek)Kullanımı

--Asagıdaki işlemde veri türü aynı olmadıgı için Birleştirme işlemi gerçekleşmedi.Categoryname String degerde Shipvia ise integer degerdedir.

--UNION OLMASI İÇİN 2 TABLODAKİ SÜTUNDAKİ VERİLERİN BİREBİR AYNI OLMASI GEREKİR.

```
select CategoryName from Categories
union

select  ShipVia from Orders
```

Doğru işlem Şu Şekildedir.AuB Şeklindedir.A birlesim B diye okunur.

```
select CategoryName from Categories
Union
select ProductName from Products
```

--Buradaki işlemde veri türleri aynı oldugu için birleştirme işlemi gerçekleşir.

Tekrarlı kayıtların(Unique) yani benzersiz degerlerin ekrana gelmesini istiyorsak **UNION ALL kullanırız.**

Kullanım Şekli:

```
Select alan_ad(ları)_1   from tabloadı_1

UNION ALL
```

Select alan_ad(ları)_2 from tabloadı_2

EXCEPT KOMUTU(Dışında,Haricinde)

EXCEPT KOMUTU: Except komutu iki farklı sql sorgusundan birinin sonuç kümesinde olup diğerinin sonuç kümesinde olmayan kayıtları listeler. Örneğin A ve B adında iki farklı küme düşünelim. **Except** komutu A kümesinde bulunup B bulunmayan veyahut B kümesinde bulunup A da bulunmaya kayıtları listeleyecektir.Yani matematiksel gösterimle A/B şeklinde Gösterilir.

Kullanım Şekli:

Select Sütun_adı from tablo_adı

Except

Select Sütun_adı from tablo_adı

Örn: select CategoryName from Categories
except

select ProductName from Products

Açıklama=Burda Sadece Product Tablosunda olan Category Tablosunda olmayan kayıtları getirir.

INTERSECT KOMUTU(Kesişim) Intersect komutu ise iki farklı sql sorgusunun kesişim noktalarını bulur. Yine küme

örneğine dönecek olursak **intersect** komutu A ve B kümelerinin ikisinde de bulunan kayıtları listeleyecektir.Matematiksel Gösterim AnB şeklindedir.

Kullanım Şekli:

Select Sütun_adı from tablo_adı

Intersect

Select Sütun_adı from tablo_adı

Örn:

```
select CategoryName from Categories
intersect
select ProductName from Products
```

Açıklama:Burda ise Category Tablosundaki Category Name ile Product tablosundaki Product Name 'da ortak olan yani kesişen kayıtları getirir.

Like Kullanım:

Bu dizin personel tablosundaki adı alanında A harfi ile başlayan tüm kayıtları getirir.

--Herhangi bir kelimede **A ifadesi** Geçiyorsa %A% Kullanılır.

Örn:Select *From personel Where sehir ad LIKE '%A%

-- MARKASI A ile baslayan sütunlar

--select * from TBLPRODUCT where BRAND like 'A%'

-- MARKASI K ile biten sütunlar

--select * from TBLPRODUCT where BRAND like '%K'

--Herhangi bir karakterle başlayan ve ardından "ondon" ile başlayan bir Şehre sahip tüm müşterileri seçer.

```
SELECT * FROM Customers
WHERE City LIKE '_ondon';
```

--Aşağıdaki SQL ifadesi, "b", "s" veya "p" ile başlayan Şehri olan tüm müşterileri seçer.

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

--Aşağıdaki SQL ifadesi, "a", "b" veya "c" ile başlayan Şehri olan tüm müşterileri seçer.

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

As Komutu Kullanımı

Kayıtlar listelendiğinde sütun başlıkları veritabanındaki başlık ile aynı görünecektir. Listeleme sırasında görünecek başlıkları değiştirmek istersek As komutu kullanabiliriz. Bu kullanım tablodaki sütun başlığını değiştirmez, sadece listelerken farklı görünmesini sağlar.Takma isim olarakta düşünebiliriz.

```
SELECT adi AS Adı, soyadi AS Soyadı FROM personel
```

Sql dilindeki diğer joker karakterler:

_ (Alt tire): Tek bir karakter gelebilir.

[bsz]= Belirtilen karakterlerden biri gelebilir (bsz)

[a-d]= a ile d harfleri arasındaki tüm harflerden biri gelebilir (a, b, c, d)

[^a-k]=a ile k arasındakiler olmayacak.Diğerleri olucak.

Is Null Kullanımı

Belirtilen alanın boş olduğu kayıtları seçmek için **is null**:

Örn: SELECT * FROM personel WHERE diplomanotu **Is Null**

Boş olmayanları seçmek içinse **is not null** şeklinde kullanabiliriz

Örn:

SELECT * FROM personel WHERE diplomanotu **Is Not Null**

Order By Komutu:

a)asc= **Küçükten büyüğe** doğru sıralar.Ve A'dan Z'ye kadar Gösterir.

b)desc=**Büyükten Küçüğe** doğru sıralar.Z'den A'ya kadar gösterir.

Asc veya desc belirtilmezse, varsayılan olarak Asc kabul edilir ve artan sıralama yapılır.

SELECT TOP KOMUTLARI

-Select TOP * from Products

a)Product Tablosundaki Tüm Sütunları Gösterir.

Select Top 10 * from Products

b)Product Sutündaki Tüm sutunlardaki ilk 10 Kayıdı Gösterir.

Select Top 20 * from Employees **order by 1 asc**

c)Burda ise Top komutu Employees tablosunda ilk 20 kayıdı getirir ve orderby komutu ile 1.sütuna göre sıralar.

DISTINCT:

Sorgu sonucu tekrarlı satırları önlemek için DISTINCT ifadesi kullanılır.

Distinct ifadesi Select ifadesinden sonra Sütün İsimleri gelir.From ve Tablo ismi şeklinde ifade edilir.Unique(tekil) ifadeleri gösterir.

Örn:Kullanım Şekli

SELECT DISTINCT * FROM tablo_adi

GROUP BY KULLANIMI:

GROUP BY komutu belirtilen alandaki değerleri gruplandırarak, her grup için istenen bilgiyi elde etmemizi sağlar.

Birden fazla group by kullandığımız zaman exceldeki pivot table gibi davranır.

HAVING KULLANIM:

SQL de Having Kullanımı

Group by dan sonra kümeleme fonksiyonlarını filtreler.

--Kümeleme fonksiyonlarında where yerine having kullanılır.

Kümeleme fonksiyonları Avg(),Sum(),count(),max(),min()'dir

Select Productname,sum(UnitPrice) as ToplamFiyat from Products Group by ProductName having SUM(UnitPrice)>100

JOİNLER NEDİR?

(INNER) JOIN=iki tablodaki **eşleşen kayıtlar için kullanılır.**

Select P.ProductName,C.CategoryName, c.Description,C.Picture from Products P join Categories C ON p.CategoryID=C.CategoryID

LEFT(OUTER)JOİN=iki tablodaki eşleşen kayıtlar ve eşleşmeyen sol kayıtlar için kullanılır.

Yani **Soldaki tablonun tümünü ve sagdaki tabloda ki eşleşen kayıtları döndürür.**

Burda ise **Product tablosundaki Categoryıd ile Categori tablosundaki Categori ID esitlenir.Sonrasında join işlemi uygulanıp left joinin sol tarafına yazdıgımız Region tablosundaki tüm kayıtları ve Territories tablosundaki eşleşen kayıtları getirir.**

Select T.TerritoryDescription,R.RegionDescription from Territories T left join Region R ON T.RegionID=R.RegionID

RİGHT(OUTER)JOİN=iki tablodaki **eşleşen kayıtlar ve eşleşmeyen sağ kayıtlar için kullanılır.**

--Burda ise Product tablosundaki Categoryıd ile Categori tablosundaki Categori ID esitlenir.Sonrasında join işlemi uygulanıp right joinin sag tarafına yazdıgımız Region tablosundaki tüm kayıtları ve Territories tablosundaki eşleşen kayıtları getirir.

Select T.TerritoryDescription,R.RegionDescription from Territories T right join Region R ON T.RegionID=R.RegionID

FULL(OUTER)JOIN=**iki tablodaki eşleşen kayıtlar ve eşleşmeyen sol ve sağ kayıtlar için kullanılır.LEFT ve RIGHT JOIN birleşimidir.**

SQLDE VİEW KULLANIMI:

Create View : **Yeni view oluşturmak için kullanılır.**

Alter View : Daha önceden oluşturulmuş bir view üzerinde değişiklik yapılmak için kullanılır.

Drop View : Daha önceden oluşturulmuş bir view'i veritabanından silmek için kullanılır.

With Encryption : View i şifrelemek için kullanılır.

Viewların görevi

Create View TakmaİSim as select [sutun ismi] from tabloismi

Sql View in İçerisinde Yapılamayacak İşlemler Nelerdir?

1-İsimsiz bir kolon kullanılamaz. (sql aggregate fonksiyon kullanımından sonra kolon ismi boş gelir. Sql aggregate fonksiyonları görmek için tıklayın.)

2-Sql stored procedure yapısı gibi, view içerisinde parametre gönderilemez.

3-View yapısı içerisinde dml kodları (insert into, update, delete) kullanılamaz.

4-View yapısı içerisinde, sadece select ile başlayan ifadeler kullanılabilir.

5-View içerisinde order by (sıralama) fonksiyonu kullanılamaz.

Viewlar

Viewlar tabloları daha düzenli hale getirip ayrı ayrı tablolara bölmemizi sağlar.
Kullanıcıların bazı kritik tabloların sadece belli sütunlarını veya satırlarını göstermesi gerektiği durumlarda kullanılabilir.Viewler select komutuyla çalışır.

```
CREATE VIEW view_adi
AS
SELECT sütun_adları
FROM base _tablo_adı
```

Örn: Create View İsmiAhmetOlanlar
As
Select FirstName,LastName from Customers where FirstName='ahmet'

select * from İsmiAhmetOlanlar şeklinde Çağırılır.

FirstName	LastName
ahmet	Alexander
ahmet	Bryant
ahmet	Butler
ahmet	Coleman
ahmet	Con
ahmet	Edwards
ahmet	Flores
ahmet	Foster
ahmet	Griffin
ahmet	Henderson
ahmet	Hernandez
ahmet	Hill
ahmet	Jai
ahmet	Jenkins
ahmet	Lal
ahmet	Li
ahmet	Nicholls
ahmet	Perry
ahmet	Powell
ahmet	Roberts
ahmet	Russell
ahmet	Shan
ahmet	Simmons
ahmet	Allen

Store Procedure Kullanımı

1-Store prosedure fonksiyonlara benzer.

2-Performans acısından verimlidir.

Kullanım:

CREATE PROCEDURE or CREATE PROC prosedür_adı

AS

Begin

//SQL Cümlecigi

örn:

Select * from Categoriler v.b

end

Cagırırkende= Exec veya execute prosedur_adı olarak cagırılır.

Yaptıgımız birşeyi prosedur üzerinden degisiklik yapmak istiyorsak Alter Procedure Kullanılır.

ALTER PROCEDURE or ALTER PROC prosedür_adı

AS

Begin

//SQL Cümlecigi

örn:Select * from Categoriler v.b

End

Yaptıgımız birşeyi prosedur üzerinden Silmek yapmak istiyorsak Drop Procedure Kullanılır.

DROP PROCEDURE or CREATE PROC prosedür_adı

AS

Begin

//SQL Cümlecigi

Örn:Select * from Categoriler v.b

End

Parametreli Prosedür Oluşturma:

ALTER proc [dbo].[ParametreliVeriler]

@kullaniciadi as varchar(100),

@id as int

as

begin

Select ProductName,ProductID from Products where ProductID=**@id** and ProductName=**@kullaniciadi**

End

Çalıstırmak için:

exec ParametreliVeriler 'Chang','2'

ProductName ProductID

Chang 2

Çıktısı Yukarıdaki Gibidir.

Prosedurlerde parametreli oldugunda varsayılan degerde alabilir.

örn:

ALTER proc [dbo].[ParametreliVeriler]

@kullaniciadi as

varchar(100)='ismail',@id as int=14

SQLDE STRİNG İŞLEMLERİ

1)Substring Fonksiyonu

Örn: select SUBSTRING('ısmail akgul',1,2)

--Substringde 1.karakterden başlayarak 2 karakter getir anlamına gelir.

Çıktısı=is şeklinde sonuç döndürür.

2)Concat Fonksiyonu

2 şekilde yazılır=

1)select 'ısmaıl'+'akgul'+'1234'

Cıktı=ısmaılakgul1234 şeklinde yazılmıştır.

//Artı işaretiyle koyduğumuzda string ifadeleri birleştirir.

2) select CONCAT(FirstName,' ',LastName) as AdSoyad from Customers

Örn:

	AdSoyad
1	ahmet Alexander
2	ahmet Bryant
3	ahmet Butler
4	ismail desouza
5	ahmet Coleman
6	ahmet Con
7	ahmet Edwards
8	ahmet Flores
9	ahmet Foster
10	ahmet Gonzales
11	ahmet Griffin

3) *Left Fonksiyonu:*

Select LEFT('Ismaıl',3)

Left komutu soldan üç harfi alır.

4)Rıght Fonksiyonu:

select Rıght('Ismaıl',3)

Rıght komutu **sağdan üç harfi alır.**

5)Trim Kullanımı

trim() fonksiyonu **string'in başındaki ve sonundaki boşlukları veya silinmesini istediğiniz karakterleri siler.**

1. **trim**(string)
2. **trim**(string,charlist)

örn:

a)Rtrim Fonksiyonu

string'in başındaki boşluğu siler.

rtrim(string)
rtrim(string,charlist)

select RTRIM('ismail Akgul ')

b)Ltrim Fonksiyonu

string'in sonundaki boşluğu siler.

select LTRIM(' ismail Akgul')

ltrim(string)
ltrim(string,charlist)

TRIGGERLAR

Kullanım Amacı:
Tablo üzerinde bir işlem gerçekleştiğinde (insert, update, delete) başka bir işlem daha yapılmak istendiği zaman kullanılır.Yani insert yapıldığı anda başka bir veri silinsin v.b

Mantıgı:
Trigger'lar DML (Delete, Insert ve Update işlemleri) işlemleri üzerinde çalışırlar.
Select ile bir bağlantısı yoktur.
DML işlemi gerçekleştiğinde deleted, inserted yada her iki sanal tablolaya kayıtlar eklenir.
Trigger tetiklenerek, trigger içindeki sorgular gerçekleştirilir. Sonra da kayıtlar gerçek tabloya(rollback ile sorguyu iptal edebiliriz) eklenerek sorgu işlemi bitirilir.

Create Trigger Trigger_adi

ON tablo_adi

{FOR,AFTER,İNSTEAD OF} {İNSERT DELETE UPDATE}

AS
BEGİN
--Sql cümlecikleri

END
trigger_adi: oluşturacağımız trigger adını veriyoruz.
tablo_adi: Hangi tabloda çalışacağını belirtiyoruz
After yada Instead of:
After yapılırsa işlem yapıldıktan sonra,
İnstead of yapıldığında yapmak istediğimiz işlemin yerine çalıştırılacağını belirtiyoruz
Insert,Update, Delete: Tabloda hangi sql işlemi sırasında çalıştırılacağını belirtiyoruz.

Inserted / Deleted Kavramları

Trigger'ın çalışmasına göre, hangi tablo üzerinde etkinse tablo üzerine eklenilen veya güncellenilen kayıtları "inserted" üzerinde, silinen kayıtları ise "deleted" üzerinde kaydediyor. Bunlar sanal tablolardır (virtual table).
Insert: Bir insert işlemi gerçekleştirdiğimizde, eklediğimiz kayıt ilk olarak **inserted** tablosuna eklenir.
Delete: Bir delete işlemi gerçekleştirdiğimizde sildiğimiz kayıt ilk olarak **deleted** tablosuna eklenir.
Update: Update işleminde ise eski kayıt
önce **deleted** tablosuna, güncellenen(yeni) kayıt
ise **inserted** tablosuna eklenir.

Trigger enable/disable yapma

Disable yapma:

1-Yöntem

Disable trigger trigger_adı On tablo_adı

--2.yöntem
alter table tablo_adı disable trigger trigger_adı

Trigger Kaldırma

Drop Trigger trigger_adı

Tüm Triggerları Pasifleştirme / Aktifleştirme

enable trigger all On tablo_adı --aktif

disable trigger all On tablo_adı --Pasif

After **/Instead** Of Kavramı
After Triggerler:
Insert,Update yada Delete işlemi gerçekleştirildiğinde tetiklenir.
Not: Sadece tablolar için tanımlanabilir.
Instead Of Triggerler:
Belirlenen işlem gerçekleşirken devreye girip, SQL sorgusu yerine çalıştırılır.
After tetikleyicileri sadece tablolar için tanımlanabilirken Instead Of tetikleyicileri hem tablolar için hem de view tabloları ile de kullanılabilir.

Instead Of:

Bir INSERT, UPDATE veya DELETE i şlemi bir tabloya uygulandığında bu tablo üzerinde , sırasıyla
bir Instead Of INSERT, Instead Of UPDATE veya Instead Of DELETE tetikleyici varsa bu işlem tablo üzerinde gerçekle şmez.
Onun yerine tetikleyici içinde yazılı kodlar yapılır.

İNSTEAD OF

ÖRNEK:

```
Create trigger EklerkenSilmeTrigger2

ON Bölme
--Bölme tablosu
INSTEAD OF INSERT
 --Burada ise Bölme işleminde İnsert into yazdıgımızda ekleme işlemi gerçekleştirmez.
--Yalnızca begin end arasındaki blogun içindeki kod devreye girer.
AS
BEGIN

insert into Bölme2 values('TELEVİZYON')
--Yalnızca Bölme 2 tablosundaki insert into işlemi tetiklenir. Yukarıdaki Bölme tablosundaki ekleme işlemi tetiklenmez.

END
```

AFTER ÖRNEK:

```
Create trigger EklerkenSilmeTrigger

ON Bölme
--Bölme tablosu
AFTER INSERT
--İnsert işlemi yaparken
AS
BEGIN

İnsert into Bölme2 values('TELEVİZYON')
--Bölme 2 tablosundaki insert into işlemide tetiklenir.
--Yani bir işlem yapıyorsak diger işlemide tetiklemiş
oluyoruz.Bölme tablosuna veri eklendiğinde Bölme2
tablosunada Televizyon eklenir.
END
```

ÖRNEK2:

```
CREATE TRIGGER ARTTIR
 TBTPRODUCT Tablosundaki ekleme işlemi yaptıktan sonra
deneme tablosunda Güncelleme işlemi yap.
ON TBLPRODUCT –Tablo isim
AFTER INSERT
AS
UPDATE Deneme set Toplam=toplam +1
```

Örn:

```
Create trigger
Categori_Tablosunda_Veri_Ekledikten_Sonra_Product_Tabl
osunu_Goster
on  Categories
after insert
as
begin

select * from Products

end
```

Açıklama=Başlangıçta Categori tablosunda veri ekledikten sonra Product tablosunu göster anlamına gelir.

DCL'de ise,

kullanıcıya yetki tanımlama için

GRANT,

kullanıcı yetkilerini engellemek için

DENY

ve

Daha önce yapılmış olan yetki ve izinleri kaldırmak için REVOKE komutları kullanılır.

Bir veri girişi sırasında bazı bilgilerin zorunlu dolması gerekiyorsa bu bilgilere karşılık gelen veritabanı özniteliklerinin de uygun şekilde tanımlanması gerekir. SQL, öznitelik değeri olarak NULL kabul edebilir. Veri tanımlaması gerekli ise ilgili öznitelik için NOT NULL kısıtının tanımlanması gerekir.

1-CAST

Örn: CAST Sorgu esnasında veri türünün dönüştürülmesini sağlar.

(Veritabanı yapısını değiştirmez)

Cast ([Tutar] as nvarchar (20))+ 'TL dir.' Sonuç: 20.00TL dir.

CONVERT Cast ile aynı işlevi gerçekleştirir.

Convert (nvarchar (20),[Tutar])+ 'TL dir.'

2-PARSE

PARSE Bir ifadedeki bilgili ilgili dile göre metin içerisinden çıkarır.

Parse('19 Haziran 1982' AS datetime USING 'Tr-TR')

Sonuç: 1982-06-19 00:00:00 dir.

3-LOWER

LOWER Metinleri küçük harfe çevirir.

Lower ('KÜÇÜK HARF')

Sonuç: küçük harf

4-UPPER

UPPER Metinleri büyük harfe çevirir. Upper('büyük harf') Sonuç: BÜYÜK HARF ROUND Ondalıklı bir sayının istenen basamağa kadar yuvarlanmasını sağlar.

5-ROUND

ROUND(123.4545, 2); Sonuç: 123.4500

6-GETDATE

GETDATE SQL Server yazılımının çalıştığı bilgisayarın sistem tarihini döndürür. GetDate() Sonuç: 2017-06-01 11:00:12.98 DAY, MONTH, YEAR Tarih veri türündeki değişkenlerin sırasıyla gün, ay ve yıl verisine dönüştüren işlevlerdir. Day('2018/5/17')

Sonuç: 17 Month(FaturaTarihi)

Sonuç: 05

Year(GetDate())Sonuç: 2017

LTRIM, RTRIM Karakter türündeki değişkenlerin başındaki (LTRIM) ya da sonundaki (RTRIM) boşluk karakterlerini kaldırır. LTRIM(' TÜRKİYE') Sonuç: 'TÜRKİYE'

SUBQUERY KULLANIMI

1-Kullanım

-Tablo yerine kullanılabilir.

```
select * from
 (select COUNTRY,CITY from LG_317_CLCARD where
COUNTRY='Türkiye') iller
```

2-KULLANIM

-JOIN GİBİ 2 TABLOYU BİRLEŞTİRMEMİZİ SAGLAR.

--Aşagıdaki subquery ifadesinde select ifadesinden sonra istedigimiz ACCRISKLIMIT sütununu yazarız.
--Ana tabloya baglı çalışır.
Joındeki karsılıgı LEFT JOIN'dir.
CLCARD ana tablo oldugu için joine göre left joın olarak ifade edilir.

--ÇALIŞMA MANTIĞI

--Çalışma mantıgı ise

--Aşagıdaki subquery ifadesinde select ifadesinden sonra istedigimiz ACCRISKLIMIT sütununu yazarız.
--Sonrasında where ile LG_117_01_CLRNUMS tablosundaki CLCARDREF ile LG_117_CLCARD tablosundaki C.[LOGICALREF] ifadesi join gibi ilişkiye girer.
Ana tabloyla subquery içindeki tablo daima left join mantığıyla çalışır.
Yani oradaki where joindeki On işlevini üstlenir.
--Burada CLCARD tablosu ana tablo oldugu için CLCARD tablosunun tamamını getirir.
LG_117_01_CLRNUMS tablosundaki eşleşenleri getirir.
Eşleşmeyen kayıtlar ise null olarak döner.

--Ana tabloya baglı çalışır.
Joındeki karsılıgı LEFT JOIN'dir.
Clcard ana tablo oldugu için joine göre left joın olarak ifade edilir.

Çeşitleri:

1-Tek Sonuç Döndüren Subqueryler

Bu tür ana sorguda (=, >, >=, <, <=, <>) gibi oparatörler kullanılıyorsa subquery den **tek sonuç dönmelidir.** Tek sonucu bütün satırlarda gösterir.

Örn:

```
SELECT * FROM
(SELECT * FROM EMPLOYEES WHERE COUNTRY = 'UK')
AS İSCİLER;
```

```
--SUBQUERY KULLANIM
Select

  (SELECT ACCRISKLIMIT FROM
TENTERPRISE..LG_117_01_CLRNUMS WITH (NOLOCK)
WHERE CLCARDREF=C.[LOGICALREF]) RISK_LIMITI

   From TENTERPRISE..LG_117_CLCARD C
   where CARDTYPE not in(4,22)

--JOIN KULLANIM
SELECT ACCRISKLIMIT
```

```
FROM TENTERPRISE..LG_117_CLCARD  C

WITH (NOLOCK)

LEFT JOIN LG_117_01_CLRNUMS CLR ON
C.LOGICALREF=CLR.CLCARDREF


 where CARDTYPE Not in(4,22)
```

Açıklama:Çıktı olarak aynı çıktıyı verir.

BOSLUK,NULL VE STRİNG SAYIYLA SÜTUN OLUŞTURMA

```
SELECT '' as Bosluk,

'317' as StringYazıHaliyle,

NULL AS NullİfadesiFROM LG_317_CLCARD
```

Açıklama:

--CLCARD TABLOSU KAÇ SATIR İSE O KADAR SATIR DÖNDÜRÜR.
--UNION ALL İFADELERİNDE KULLANILIR.

--YANİ TABLODA 1200 SATIR VARSA 1200 SATIR BOSLUK,317 VE NULL DEGERİ DÖNDÜRÜR.
--ÇÜNKÜ FARKLI HER İKİ TABLODA SÜTUN İSİMLERİ AYNI OLMAK ZORUNDA VE TABLONUN ÇALIŞMASI İÇİN SÜTUN OLARAK GÖZÜKMESİNİ SAGLIYORUZ.

Çıktısı:

	Bosluk	StringYazıHaliyle	Nullifadesi
103		317	NULL
104		317	NULL
105		317	NULL
106		317	NULL
107		317	NULL
108		317	NULL
109		317	NULL
110		317	NULL
111		317	NULL
112		317	NULL
113		317	NULL
114		317	NULL
115		317	NULL
116		317	NULL
117		317	NULL
118		317	NULL
119		317	NULL
120		317	NULL
121		317	NULL

TRANSACTİON

Bir veya birden fazla SQL ifadesi arka arkaya tek bir işlem gibi çalıştırılmak istenildiği zaman TRANSACTION yapısı kullanılır.

TRANSACTION özellikle ardı ardına gelen ve birbiriyle bağımlı birden fazla SQL komutu tek bir SQL komutu olarak kullanılmasını sağlar.

Bu sayede bir TRANSACTION kullanılarak yazılan SQL komutlarının ya tamamını gerçekleştirir veya hiçbiri gerçekleştirilmez.

TRANSACTION işlemleri tek bir işlem olarak ele alacağı için herhangi birisi gerçekleşmediği zaman diğer gerçekleşen işlemleri de yok sayacaktır.

ROLBACK VE COMMIT KAVRAMI

ROLLBACK: Transaction işlemindeki SQL komutu tarafından yapılmış olan tüm değişikliklerin geri almak için kullanılmaktadır.

Yazılan SQL komutlarının herhangi birinde hata meydana gelmesi hâlinde bir sorun karşısında ROLLBACK işlemi ile kayıtları ilk hâline (transaction başladığı duruma) geri getirir.

COMMIT: Oluşturulan TRANSACTION işlemi başarılı bir şekilde gerçekleştirildiğinde veritabanında tablolar üzerinde yapılan işlemlerin (örneğin INSERT, UPDATE, DELETE) tablolara kalıcı olarak aktarılmasıdır.

SAVE TRANSACTION:Rollback durumunda transaction içinde yapılan işlemlerin en baştan tekrarlanması yerine kaldığı noktayı işaretler ve oradan işleme devam eder.Kaldıgı yerden devam eder.

@@ERROR:Son çalıştırılan SQL ifadesinin sonucunda oluşan hata numarasını tutar.İfade başarılı ise 0 degerini alır.

@@TRANCOUNT=Kullanılan bağlantı üzerinde kaç tane aktif transaction olduğunu belirtir.

ÖRN:

BEGIN TRANSACTION INSERT INTO [Stok Hareket Türleri] VALUES (5,'Sipariş Verildi') INSERT INTO [Stok Hareket Türleri] VALUES (6,'Sipariş İptal Edildi') ROLLBACK

Eğer aynı TRANSACTION yapısı aşağıdaki örnekte görüldüğü gibi ROLLBACK yerine

COMMIT ifadesi ile yazılsaydı kayıtlar Bolge tablosuna kalıcı olarak aktarıldığı alttaki şekilde görünmektedir.

ÖRN:

BEGIN TRANSACTION INSERT INTO [Stok Hareket Türleri]

VALUES (5,'Sipariş Verildi') INSERT INTO [Stok Hareket Türleri]

VALUES (6,'Sipariş İptal Edildi')

COMMIT

BAĞIMLILIK

FONKSİYONEL BAĞIMLILIK Normalleştirme fonksiyonel bağımlılıkların analizine dayalı olarak yapılır. Fonksiyonel bağımlık iki set öznitelik arasındaki kısıtlardır. Bir veritabanındaki herhangi bir tablo T ve bu tablodaki iki öznitelik A ve B olsun.

Eğer A özniteliğinin değeri B özniteliğinin değerini belirliyorsa B özniteliğinin A özniteliğine bağımlı olduğu söylenir.

B'nin A'ya fonksiyonel bağımlılığı ok işareti ile A→B şeklinde gösterilir.

Kısmi Bağımlılık (Partial Dependence) Bir tabloda birincil anahtar bir veya daha fazla öznitelikten oluşabilir. Eğer birincil anahtar iki veya daha fazla öznitelikten oluşuyorsa bu tür birincil anahtarlara birleşik anahtar (composite key) adı verilir. Bu tür tablolarda anahtar olmayan öznitelik, birleşik anahtarın sadece bir kısmı ile belirleniyorsa buna kısmi bağımlılık denir.

Bir veritabanında yer alan tabloda A, B, C ve D gibi toplam dört özniteliğin olduğunu varsayalım. Bu tablonun birincil anahtarı da (A, B) olsun. Bu durumda AB→CD fonksiyonel bağımlılığı yazılabilir. Bu tabloda A→C fonksiyonel bağımlılığını da varsayalım. Bu durumda C özniteliği birleşik anahtarın sadece bir kısmı olan A özniteliğine bağımlıdır. Yani C özniteliği A'ya kısmi bağımlıdır.

Geçişli Bağımlılık (Transitive Dependence) Bir tabloda yer alan bazı özniteliklerin başka bir öznitelik aracılığıyla üçüncü bir özniteliğe bağımlı olması geçişli bağımlılık olarak adlandırılır.

Tam Fonksiyonel Bağımlılık (Full Functional Dependence) Bir özniteliğin değeri bağlı olduğu anahtar ile benzersiz olarak belirleniyorsa bu ilişki tam bağımlılık olarak adlandırılır.

Çok Değerli Bağımlılık (Multiple Valued Dependence) Tabloda bir alandaki değerler virgülle ayrılarak oluşturulan liste veya dizi değerlerinden oluşuyorsa çok değerli bağımlılık vardır.

Döngüsel Bağımlılık (Cyclic Dependency) Döngüsel bağımlılıkta döngü kapalı halka, tekrar etme anlamında kullanılmaktadır. Veritabanında bir öznitelik A, diğer öznitelik B'ye bağımlı iken aynı zamanda B özniteliği de A'ya bağımlı ise döngüsel bağımlılık olarak adlandırılır.

BCNF, 4NF ve 5NF Günümüzde modern ilişkisel veritabanı modellerinde genelde 3. normal formdan sonrası genelde uygulanmaz.

3NF'de Hesaplanmış Değerleri Saklayan Alanların Durumu 3NF'de aynı tablodan hesaplanarak bulunan değerler birincil anahtara geçişli olarak bağımlı olduklarından kaldırılır.

Birinci Normal Form (1NF) Birinci normal formda çok değerli ve çok parçalı öznitelikler ve tekrar eden gruplar kaldırılır.

İkinci Normal Form (2NF) 2NF'de tekrar eden değerler yeni tablolara taşınmaktadır. 2NF kuralları aşağıdadır: • Tablolar 1NF olmalıdır. • Tüm anahtar olmayan alanlar birincil anahtara tam fonksiyonel bağımlı olmalıdır. • Kısmi bağımlılıklar kaldırılmalıdır. Önceden de bahsedildiği gibi kısmi bağımlılık fonksiyonel bağımlılığın bir özel durumu olup bir alanın birleşik anahtarın bir parçasına tam bağımlı olması

demektir. Kısaca birincil anahtara kısmi bağımlı alanlar yeni bir tablo oluşturarak taşınmalıdır.

IF-ELSE Yapısı

IF-Else yapısı komutların belli bir yapıya göre çalışmasını sağlar. İşlenecek komut satırı birden fazla ise

BEGIN – END yapısı kullanılır.
Örn: IF(Şart1)

```
BEGIN
   İşlemlerimiz
END
ELSE IF(Şart2)
BEGIN
   İşlemlerimiz
END
ELSE
BEGIN
   İşlemlerimiz
END
```

Örn:

```
Declare @Stokmiktarı as int

--declare diyerek @degiskenİsmi yazdık
select @Stokmiktarı=Sum(UnitsInStock)  from Products

if(@Stokmiktarı>15)

begin

print ' 15 ten büyük'
end

else
begin

print '15 ten kücük'

end
```

ELSE İF KULLANIMI:

-İf leri gezer.
-Sağladığı ilk if koşulun çıktısını verir.
Hiçbir koşulu sağlamadıgı taktirde else çalışır.
Örn:

```
declare @s1 int=1
declare @s2 int=2
declare @s3 int=3

if(@s1=0)
-- 1 0 a esit degil alt kosula geçer
```

```
print 'dogru'

 else if(@s2=0)

--2 0 'a esit degil yazmadı alt if'e geçer.

print 'dogru'

else  if(@s3=30)

--3 0 a esit degil o yüzden koşulu sağlamadı.
print 'dogru'

else
 print  'Yanlış'
```

Çıktısı:

Hiçbir koşul sağlamadıgı için Yanlış sonucunu döndürür.

```
Messages
    Yanlis
```

Örn2:

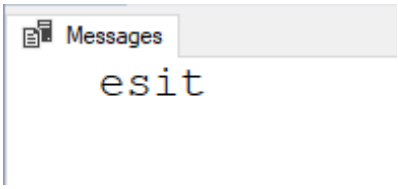
```
if(@s1=1)

print 'esit'
```

--Burada ise 1=1 e esit oldugu için öteki koşullara bakmadan dogru olan if kosulunun çıktısını getirdi.

Çıktı olarak eşit döner.

```
 else if(@s2=0)

print 'esit degil'

else  if(@s3=30)

print 'esit degil'

else
 print  'Yanlış'
```

Çıktısı:



İÇİÇE İF KULLANIMI

```
if(@s1=1)
    begin

        --Tüm if kosullarını dolasır.
```

Koşulu saglayan tüm if kayıtlarını getirir.

Koşulu sağlamayan koşulları getirmez.

```
-- 1 0 dan büyük dogruyu yazdı

print 'dogru'
end

if(@s2=3)
begin
--0 2 den büyük degil yazmadı alt if'e geçer.

print 'yanlıs'

end

if(@s3=3)
begin

--30 3 ten büyük o yüzden dogru yazdı.
print 'dogru'

end
```

Çıktı:Çıktı olarak koşulu sağlayan değerleri getirmiş olduk.

```
Messages
dogru
dogru
```

WHİLE DÖNGÜSÜ KULLANIMI

```
 declare   @Sayac int =1
--Sayac default olarak 1 verilir.
    while  @Sayac<=7
    --While sayac diyerek 1 den 7 dahil'e kadar tekrarla
demiş oluyoruz.
    begin
    -- 7 kere eklememizi saglar.
    insert into ForMantıgı
Values(@Sayac,DATEADD(MONTH,1,GETDATE()))

    set @sayac=@Sayac+1
    --7 kere eklemiş oluruz.

    end
```

Continue Komutu:

Bazı durumlar için bazen işlem yapmadan döngüyü devam ettirmek isteriz.

Bu gibi durumlarda da CONTINUE deyimini kullanırız. CONTINUE kendisinden sonraki gelen ifadeyi yok sayar.

Örnek: 1 - 5 arasında ekrana yazdırır

fakat 2 sayısına geldiğinde yazıyı yazdırmadan

geçer.

```
DECLARE @Sayi int=0
```

yada set @sayi=0 şeklindede yazılabilir.

```
WHILE (@Sayi<=4)
BEGIN
   SET @Sayi=@Sayi+1
   IF(@Sayi=2)
     CONTINUE
```

Burada sayı 2 ye geldiği zaman if deyimine girer @sayi1=2 ye eşit olduğu zaman continue deyimiyle 2 sayısını yazdırmaz.

```
   PRINT @Sayi
END
PRINT 'Döngüden Çıkıldı..'
```

Çıktısı:

1
3
4
5
Döngüden Çikildi..

Burada görüldüğü gibi 2 haricindeki 1 den 5 kadar olan sayıları yazdırdı.

Break Komutu:

Şart yerine getirildiğinde döngüden çıkmak için kullanılır.Yani şartı sagladıgı anda döngüden çıkar.

```
DECLARE @Sayac INT
--İlk önce declare terimiyle degisken tanımlarız.
1 2 3 4
SET @Sayac=0
--Sonrasında ise set ifadesiyle birlikte degiskene deger
veririz.
WHILE (@Sayac<10)
--While @sayac=0 degeri select
 ifadesinden sonra 1 arttırır.
 BEGIN
SELECT @Sayac=@Sayac+1

 IF(@Sayac*@Sayac>10)
 --Sayac 1 arttırır.
10 degerini gectikten sonra print mesajıyla sayı 10 degerini
gecti mesajını verir.
```

```
    BEGIN
        PRINT 'Sayı 10 değerini geçti.'
        BREAK --döngüden cık
    END
END
```

SWİCH CASE KULLANIMI

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than
30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText


FROM OrderDetails;
```

Örn:

```
Select Ay,
--İlk önce Select yazıp sonra işlem yapmak istediğimiz
sütunun ismini yazıyoruz.
Case
-- Sonrasında Case yazıp Ay sütununda işlem yapacağımız
için Ay sütununu alıyoruz.
  Ay

when 1 then 'Ocak'
-- when den sonra AyNo su 1 olan yere Ocak yaz
anlamına geliyo
when 2 then  'Şubat'
when 3 then  'Mart'
when 4 then  'Nisan'
when 5 then  'Mayıs'
when 6 then  'Haziran'
when 7 then  'Temmuz'
when 8 then  'Agustos'
when 9 then  'eylül'
when 10 then  'Ekim'
when 11 then  'Kasım'
when 12 then  'Aralık'
--Koşul haricinde yazdıgımız degerlere ise digerleri
yazarak işlemi tamamlıyoruz.
else 'digerleri'
--Daha sonrasında yeni bir tablo oluşur.
--end as ifadesini yazdıktan sonra yeni oluşacak tablonun
ismini yazarız.
```

```
end as Ayisimleri5

from Aylar2
```

Çıktısı:

Ay	Ayisimleri5
1	Ocak
2	Subat
3	Mart
4	Nisan
5	Mayis
6	Haziran
7	Temmuz
8	Agustos
9	eylül
10	Ekim
11	Kasim
12	Aralik
13	digerleri
14	digerleri

SQL KULLANICI OLUŞTURMA VE KULLANICI ROLLERİ VE YETKİLERİ

Bulkadmin=Bulk insert yapma yetkisine sahiptir.Bir dosyadan veritabanına kayıt eklemek için kullanılır.

Genellikle excelden veri çekmek için kullanılır.

DbCreator=Herhangi bir veritabanını oluşturma,düzenleme ve kaldırma yetkisi vardır.

ProcessAdmin=sql processlerini görme ve sonlandırma yetkisine sahiptir.Tüm çalışan processleri görürler.

Kill komutu ile istenen process veya processleri sonlandırabilirler.

SecurityAdmin=Sql hesabı oluşturup silebilirler.Server rolü oluşturma yetkileri yoktur.

ServerAdmin=Sql server instance'nın özelliklerini değiştirebilir,yeniden başlatabilir veya hizmeti durdurabilirler.

SetupAdmin=Linked server oluşturma yetkileri vardır.

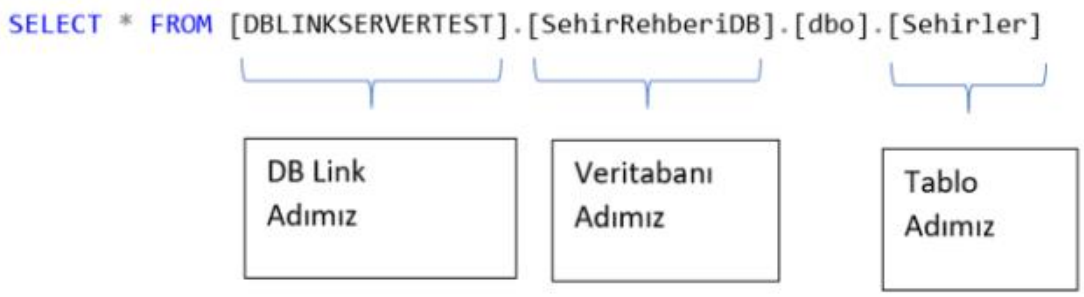
SysAdmin=Tüm yönetici yetkilerine sahip roldür.

LİNKED SERVER

SQL Server'ın önemli özelliklerinden biri olan **Linked Server SQL Server'ın OLEDB/ODBC** veri kaynakları aracılığıyla farklı kaynaklar (Oracle ,Access,Excel vb..) üzerindeki verilere erişmemizi ve bu sunucular üzerinde komutları çalıştırmamızı sağlar.

Linked Server : istediğiniz herhangi bir bağlantı ismi
Provider : SQL bağlantısı için Microsoft OLE DB Provider for SQL Server
Product Name : SQL
Data Source : Bağlantı kuracağınız SQL Server ismi olabilir , IP adresi olabilir . Soldaki işaretlediğim yerde nokta görünüyor çünkü SQL Serveri açarken . ile giriş yapmıştım locale girmek için , sizde Server ismi yazıyordur.
Catalog : Bağlantı kuracağınız Database ismi.
Sonra Security Sayfasına geçiyoruz ve "**Be made using this security context**" i işaretleyip , bağlanılacak Sql serverdaki yetkili kullanıcı bilgilerini veriyoruz.
2 türlü yazımı vardır;

1-



2-

```
select *
 from DBLINKTESTSERVER
(LINKEDSERVER, 'select * from users')
```

3- Select * from Openquery(LINKEDSERVER, 'select * from users')

LOGİN YETKİLERİ

DB_accessadmin=Veritabanına erişim için yetki verir.

DB_backupoperator=Veritabanının yedek alması için yetki verir.Sistem yöneticilerine yedek alınan 3.parti bir yazılımda kullanılmak üzere bu yetki verilebilir.

DB_Datareader=Tablolardaki verileri okuma yetkisi verir.Sadece select sorgusu kullanan ve self service raporlama yapan son kullanıcılara bu yetki verilebilir.

DB_datawriter=Tablolardaki verileri düzenleme(update) ,silme(delete) veya ekleme(insert)

yetkisi verir.Server bazında yetki kısıtlanarak farklı programlardaki farklı kullanıcılara bu yetki verilebilir.

DB_ddladmin=Veritabanı üzerinde DDL(Data Definition language) komutlarını çalıştırma izni verir.

DB_denydatareader=Veritabanı üzerinde okuma işlemi yetkisini engeller.

DB_denydatawriter= Veritabanı üzerinde yazma işlemi yetkisini engeller.

Db_owner=Veritabanı üzerindeki tüm yapılandırma ve bakım yetkilerini verir.Veritabanını drop etme yetkisine sahiptir.Owner,sahip almanına gelmektedir.

Db_Securityadmin=Veritabanı üzerindeki üyelikleri ve izinleri yönetme yetkisidir.

NOLOCK ÖZELLİGİ

Sql server'da herhangi bir tabloda kayıt üzerinde yapılan işlem sonucunda işlem sonlanana kadar server sql server tarafından 'Locking' kilitlenir ve bu işlem sonlanana kadar başka bir kullanıcının bu tablo üzerinde işlem yapılması engellenir.

Kullanımı:

```
SELECT * FROM CUSTOMERS WITH(NOLOCK)
```

TABLO ÖZELLİKLERİ

Sql tablodan yeni sütun ekleme:

```
ALTER TABLE Tablo_adi
Add kolon_adi veri_tipi(int,varchar,nvarchar);
```

Sql tablodan sütun silme:

```
ALTER TABLE tablo_adi
DROP COLUMN Kolon_adi;
```

Veri tipini Değiştirme

```
ALTER TABLE ogrenci
ALTER COLUMN DogumGunu year;
```

Sql tablodan sütun güncelleme:

ALTER TABLE tablo_adi

ALTER COLUMN Kolon_adi veri_tipi;

Tablodaki sütunun adını değiştirme

EXEC sp_rename Personel.[TCKimlikNo], TcNo ,'COLUMN'

INSERT İNTO

SELECT,SQL de tablodaki verileri başka tabloya aktarma-kopyalama

Insert into select komutu ile varolan tablodaki verilerimizi,hedef tablomuza(varolan)taşıyabiliriz.

Kullanım:

Insert into hedef_tablo(alanadi1,alanadi2,alanadi3…)

Select alanadi1,alanadi2,alanadi3 from tablo

Açıklama=Databasedan databaseyede sütun kopyalayabiliriz.

Yukarıdaki SQL ile tablomuzdaki bütün verileri hedef tablomuza kopyalamış oluruz.

Eğer belli şartları sağlayan verilerimizi taşıyacaksak sqlimize WHERE şartı eklemeliyiz.

Kullanım şekli 3 tanedir;

1-Belirli kolonları Insert edebiliriz.

```
INSERT İNTO InsertEdilecekTabloAdi

(Kolon1,Kolon2,kolon3)

Select kolon1,kolon2,kolon3 from TabloAdi
```

2-Tüm tabloyu Insert edebiliriz.

```
INSERT İNTO InsertEdilecekTabloAdi

Select * from TabloAdi
```

3-Sorguya WHERE ile filtleyerek INSERT edebiliriz.

```
INSERT İNTO InsertEdilecekTabloAdi(kolon1)

Select kolon1 from TabloAdi

Where KOLON1='Veriler'
```

Örn:

INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;

Aşağıdaki SQL ifadesi "Tedarikçiler" i "Müşteriler" e kopyalar.

Yeni oluşturduğumuz tablodaki veri tipleriyle,

Verileri alacağımız tablodaki veri tiplerinin aynı olması gerekir.Yani Customers tablosundaki sütunların veri tipleriyle Suppliers tablosundaki sütunların veri tipleri aynı olmak zorundadır.

Neden Kullanırız:

Örneğin kapsamlı bir rapor oluşturmanız gerekiyor. Firma ve yıl bazlı Alış tutarları ve miktarları, Satış tutarları ve

miktarları, toplam adetleri, kalan stoklar, stokların max min miktarları kaç fatura kesilmiş vs bir çok tablodan verileri toplayıp tek raporda göstermeniz gerekiyor. Bu gibi işlemlerde karmakarışık sorgular yerine bunu kullanabiliriz.

Örn: CREATE TABLE YILLIK_RAPOR (CariAdi [nvarchar](190) NULL, SatisTutari [float] NULL, AlisTutarı [float] NULL, FaturaAdedi [int] NULL,)

Açıklama=İlk önce tablo oluşturuyoruz.

Veri tiplerinin aynı olması ve sırasının aynı olması gerekiyor.Create Table parantezinin yanındaki sütun değeriyle select sorgusundaki sütun değerinin aynı olması gerekir.

INSERT INTO YILLIK_RAPOR (CariAdi)

SELECT CariAdlari FROM CARILER WHERE Aktif=1 order by CariKodu asc

INSERT INTO YILLIK_RAPOR (SatisTutari) SELECT SUM (Tutar) FROM SATISLAR

Group by CariKodu order by CariKodu asc;

INSERT INTO YILLIK_RAPOR (AlisTutarı)

SELECT SUM (Tutar) FROM ALISLAR group by CariKodu order by CariKodu asc;

--Ayrı tablolardan sütunlar çekilir.

INSERT INTO YILLIK_RAPOR (FaturaAdedi)

SELECT COUNT (Tutar) FROM FATURALAR Group by CariKodu order by CariKodu asc; select CariAdi, SatisTutari, AlisTutarı, FaturaAdedi from YILLIK_RAPOR

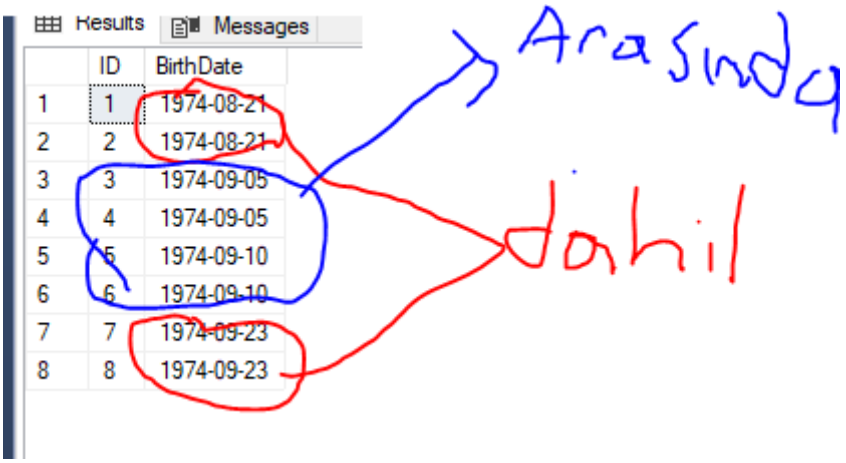
BETWEEN KULLANIMI

Örn:

Stline tablosunda 20170101 tarihinden 20170105 tarihleri arasındakileri al diyoruz. 20170101 ve 20170105 arasındaki 20170101 ve 20170105 degerleride dahil olan değerleri al demiş oluyoruz.

```
select ROW_NUMBER() OVER(order by BirthDate asc) as ID,BirthDate from DogumGunu where BirthDate between '1974-08-21' and '1974-09-23'
```

Çıktısı:



Açıklama:Yukarıdaki resimde görüldüğü üzere '1974-08-21' ve '1974-09-23' dahil arasındaki değerleri al demiş oluyoruz.Kırmızı yazılan ifadeler dahil getir demiş oluyoruz.

Değişken Tablolar (Table Variables)

Değişken tablolarda aslında geçici tablolara benzemektedir. Farkı oluşturduğumuz tablonun bir kısmı tempdb de bir kısmıda Sunucu olarak kullanıdığımız SQL Server belleğinde tutulmaktadır. Oluşturduğumuz tablo isminin başına **"@"** işareti koyarız. Değişken tablolara erişim temporary tablolara göre daha hızlıdır.

Kullanımı=

```
DECLARE @TabloAdı TABLE (kolon1 veritipi, kolon2 veritipi)

DECLARE @Musteriler TABLE (MusteriID NCHAR(5),
MusteriAdSoyad NVARCHAR(30), MusteriSehir
NVARCHAR(15))

INSERT INTO
@Musteriler(MusteriID,MusteriAdSoyad,MusteriSehir)

SELECT Customers.CustomerID, ContactName, City

FROM Customers
```

SQL REPLACE() Kullanımı

REPLACE(alan_adi,degisecek_veri,yeni_veri)

SQL FORMAT() Kullanımı

SELECT

FORMAT(alan_adi,gosterim_formati) FROM tablo_adi

Örn:

Select Format(dogum_tarihi,'DD.MM.YYYY') From Personel

Açıklama=

Yukarıdaki örnekte, dogum_tarihi alanındaki veri **Gün.Ay.Yıl** şeklinde ekrana verilmektedir. Burada dikkat edilecek nokta, formatımızı belirlerken İngilizce yapısına göre yazmamız gerekmektedir. **DD'nin anlamı DayDay** 'dir. Yani gün bilgisini çeker. **MM 'nin anlamı MonthMonth** demektir. Yani ay bilgisini çeker. **YYYY ise YearYearYearYear** demektir ve yıl bilgisini çeker.

RANKİNG SIRALAMA FONKSİYONLARI

Ranking(Sıralama) Fonksiyonları bir window fonksiyonu tipidir. Tüm pencere fonksiyonları gibi OVER() ifadesi ile kullanılırlar.

RANKING_FUNCTIONS () –buraya fonksiyon adı gelir.

OVER (

[PARTITION BY alan1,alan2,..] –isteğe bağlı

ORDER BY alan1,alan2,.

)

ROW_NUMBER=

Row_Number() fonksiyonu OVER ifadesi ile kullanılır.

OVER'dan sonra parantez içinde gruplanacak alanlar ya da sıralamayı belirleyecek alanlar eklenir.

Sıralama fonksiyonları için PARTITION BY(Sütun gruplaması yapar) kullanılması mecburi değildir. Partition by kullanılarak veri kümesi küçük gruplara bölünebilir ve sıralama bu gruplara özel oluşturulabilir. Kullanılmadığı zaman veri kümesini tek bir grup olarak algılar.

RANK=

1den başlayarak satırlara özel sıra numarası verilmesini sağlar. Çoklayan satırlarda aynı numara üzerinden devam eder. Çoklayan satırlardan sonra sıra numarasında boşluklar oluşur çünkü çoklayan kayıt sayısı kadar numara atlar.

RANK()

1den başlayarak satırlara özel sıra numarası verilmesini sağlar. Çoklayan satırlarda aynı numara üzerinden devam eder. Çoklayan satırlardan sonra sıra numarasında boşluklar oluşur çünkü çoklayan kayıt sayısı kadar numara atlar.

Aynı örnek tipi üzerinden devam edelim. Aşağıdaki örnekte şehire göre artan(şehir metin olduğu için alfabetik) bir sıralama yaptı. USA tekrarladığı için 3 satırı da 5.sıra olarak atadı. Çoklayan kayıt sayısı kadar numara atladığı için USA'den sonra gelen kayıt 6 değil 8 olarak devam etti.

```
SELECT b.ShipCountry,RANK() OVER (ORDER BY b.ShipCountry ASC) AS RN

FROM [Order Details] A

INNER JOIN Orders B on a.OrderID=b.OrderID

where ProductID='1' and EmployeeID='3'
```

ÇIKTI:

	ShipCountry	RN
1	Canada	1
2	France	2
3	Mexico	3
4	Spain	4
5	USA	5
6	USA	5
7	USA	5
8	Venezuela	8

DENSE_RANK()

1den başlayarak satıralara özel sıra numarası verilmesini sağlar. Çoklayan satırlarda aynı numara üzerinden devam eder. RANK ile karşılaştırıldığında tek fark çoklayan satırlardan sonra kaldığı numara üzerinden devam eder.

```
SELECT b.ShipCountry,DENSE_RANK() OVER (ORDER BY b.ShipCountry ASC) AS RN

FROM [Order Details] A

INNER JOIN Orders B on a.OrderID=b.OrderID

where ProductID='1' and EmployeeID='3'
```

ÇIKTI:

	ShipCountry	RN
1	Canada	1
2	France	2
3	Mexico	3
4	Spain	4
5	USA	5
6	USA	5
7	USA	5
8	Venezuela	6

NULL TANIMLARI

1-NullIf Nedir? Nasıl ve Nerede Kullanılır?

Bazende gelen verilerdeki istediğimiz değerleri Null ifadesine çevirmek isteyebiliriz. Böyle bir ihtiyacımız doğduğunda ise NullIf deyimi yeterli olacaktır.

2-Coalesce Nedir? Nasıl ve Nerede Kullanılır?

Birden fazla null koşuluna göre değer atamak istiyorsak Coalesce deyimini kullanabiliriz.

İsnullda 1 tane sütunda işlem yapabilirken Coalescede birden fazla sütunda nullmu değilmi nullsa ne yazıyım mantığıyla çalışır.

Coalesce aslında Case mantığında çalışır ve **birden fazla kolon** arasında kontrol sağlayabilirsiniz. Bir koşul gerçekleşmez ise diğerine bakar, oda gerçekleşmez ise bir sonraki. Deyim bitene kadar. **Coalesce** deyiminde koşul kriterini değiştiremessiniz. Sadece koşulun null olması ile ilgilenir.

3-IsNull Nedir? Nasıl ve Nerede Kullanılır?

Sql server sorgu sonucunda bize gelen null verileri değiştirmek isteyebiliriz. Buna ihtiyacımız olduğunda IsNull ifadesini kullanabiliriz.

SQL LENGTH FONKSİYONU

Karakter sayısını ifade eder.

Yazılışı=Len()

Örn:

```
SELECT Len('ismail')
```

Çıktısı: 6 sonucunu verir.

```
SELECT LEN(COUNTRY) AS KARAKTERSAYISI,COUNT(COUNTRY) AS SATIRSAYISI,COUNTRY FROM LG_317_CLCARD GROUP BY COUNTRY
```

DATALENGTH FONKSİYONU

Veri tipinin Byte miktarını verir.

Çıktısı=

Results | Messages

	KARAKTERSAYISI	SATIRSAYISI	COUNTRY
1	6	7	KOSOVA
2	7	7	ALMANYA
3	7	2	ESTONYA
4	8	1	LITVANYA
5	6	4	KUVEYT
6	7	1	EKVATOR
7	6	13	FRANSA
8	6	8	ITALYA

Açıklama=

Lenght sütundaki karakter sayısını

Count ise sütundaki satır sayısını ifade eder.

Format Fonksiyonu

--ISO Formatında 2 türlü şekilde yazılır.

```
--  1-  SELECT FORMAT(GETDATE(),'yyyyMMdd')

--  2- Convert(nvarchar(10),GetDate(), 112)
```

Açıklama=Sondaki parametre 112 koddur.

Getdate Tarih degerini varchar'a dönüştürüp 112 koduylada iso formatına dönüştürür.

2-Kullanım

SQL Server Offset Fetch kullanımı

SELECT deyimi ORDER BY deyimi ile birlikte kullanılır.

Dönecek başlangıç satırını OFFSET değeri ve

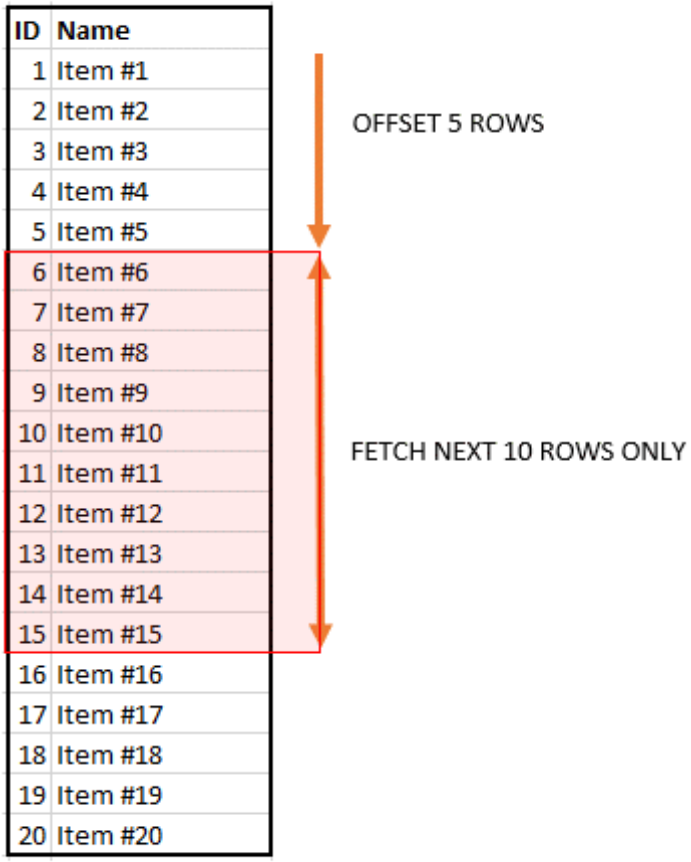
Bu noktadan döndürülecek maksimum satır sayısı FETCH tarafından belirlenir.

OFFSET=Atlanacak satır sayısı

FETCH=Döndürülecek satır sayısı

- **OFFSET** deyimi, sorgudan satır döndürmeye başlamadan önce atlanacak satır sayısını belirtir.
- **offset_row_count** , sıfıra eşit veya daha büyük olan bir sabit değişken veya parametre olabilir.
- **FETCH** deyimi, OFFSET deyimi işlendikten sonra döndürülecek satır sayısını belirtir.
- **offset_row_count** bir'e eşit veya daha büyük olan sabit, değişken veya skaler olabilir.
- FIRST ve NEXT sırasıyla eşanlamlıdır,

Böylece bunları birbirlerinin yerine kullanabilirsiniz. Benzer şekilde, **FIRST**ve **NEXT** değişmeli olarak kullanabilirsiniz.



Offset=5 satırı çıkar.

Fetch=10 satırı al anlamına gelir.

Örn:

```
SELECT
product_name,     list_price FROM     production.products
ORDER BY
list_price,product_name asc
OFFSET 10 ROWS;
```

Açıklama=İlk 10 ürünü atlayıp geri kalanını iade etmek
için, OFFSET kullanılır.

Örn:

En pahalı 10 ürünü almak için hem OFFSET hem
de FETCH cümlelerini kullanırsınız:

Offset 0=Hiçbir satır atlamadan en pahalı 10 ürünü alır.

```
SELECT
product_name,list_price FROM    production.products
ORDER BY
list_price,product_name Desc
OFFSET 0 ROWS –Hiçbir satır atlama

--En pahalı 10 ürünü getir.
FETCH FIRST 10 ROWS ONLY;
```

Örn:

İlk 10 ürünü atlayıp sonraki 10 ürünü seçmek için,
hem OFFSET hem de FETCH yan tümcelerini kullanılır.

```
SELECT
product_name,    list_price FROM    production.products
ORDER BY
list_price,
product_name
OFFSET 10 ROWS      --ilk 10 ürünü atla
FETCH NEXT 10 ROWS ONLY; --sonraki 10 ürün seçmek
için
```

TRUNCATE TABLE=SQL tabloların içindeki verileri siler.Ancak tablonun kendisini silmez.

SQL Date Data Types

- DATE - format : YYYY-MM-DD
- DATETIME - format:YYYY-MM-DD HH:MI:SS
- SMALLDATETIME - format: YYYY-MM-DD HH:MI:SS
- TIMESTAMP - format: a unique number

```
--Güncel Zaman
Select
 Day(CURRENT_TIMESTAMP),
MONTH(CURRENT_TIMESTAMP),
YEAR(CURRENT_TIMESTAMP),
--iso formatına yazdırmak için 112 kod numarası kullanılır.
CONVERT(char(30), CURRENT_TIMESTAMP, 112)
```

SQLDE WİTH TİES KULLANIMI

Top ifadesiyle beraber kullanılırlar.

- WITH TIES, ORDER BY ile sıralanan sonuçlarda
- **Son kayıt ile aynı değerde olan kayıtların da listelenmesini sağlar.**
- Bu durumda sonuç belirtilen n sayısından daha fazla olabilir.
- WITH TIES sadece ORDER BY ile birlikte kullanılmaktadır.

SELECT TOP 3 WITH TIES OrderID, ProductID, quantity
FROM [order details] ORDER BY quantity DESC
(Sipariş detayları tablosundan (order details)

en yüksek siparişi verilen 3 ürünü listeleyelim.)

Örn: SELECT TOP 1 WITH TIES * FROM
dbo.TBL_NOTLAR WHERE DERS='MATEMATİK'

ORDER BY PUAN DESC

"TOP 1" dememize rağmen aynı şartı sağlayan 2 kayıt olduğu için 2 kaydı da kolayca getirmiş olduk.

	ID	AD	DERS	PUAN
1	1	MESUT GÜNEŞ	MATEMATİK	98
2	2	MUSTAFA DEMİRCİOĞLU	MATEMATİK	98

Kısaca:

WİTH TİES kullanırken ORDER BY kullanmak zorundayız.

TARİH FONKSİYONLARI

1-Day(Tarih)

2-Getdate(Tarih)=Güncel tarihi buluruz.

3-Month(Tarih)

4-year(Tarih)

5-Datepart(Quarter, Getdate())=çeyrek'i buluruz.

6-Datediff(year,başlangıç,bitiş tarihi)=İki tarih arasındaki farkı bulmamızı sağlar.

Bitişten başlangıcı çıkarırız.

1.parametre year,month,day alabilir.

7-DateName

Açıklama=Yıl,ay,günün ismini getirir.Tüm satırda getirir.

DATENAME(month,OrderDate) AS "Sipariş Yılın Hangi Ayı",
DATENAME(day,OrderDate) AS "Sipariş Ayın Kaçıncı Günü",
DATENAME(weekday,OrderDate) AS "Sipariş Haftanın Hangi Günü"

8-DATEPART
Ay,hafta,Gün,Tarih sayı formatında gösterir.
8-DATEPART(Month,OrderDate) AS "Sipariş Yılın Kaçıncı Ay'ı",

DATEPART(Day,OrderDate) AS "Sipariş Ayın Kaçıncı Günü",

DATEPART(weekday,OrderDate) AS "Sipariş Haftanın Kaçıncı Günü"

DATEADD Fonksiyonu:

DATEADD =Ay eklememizi sağlar.

TABLONUN YEDEGİNİ ALMA

Kullanım:

Select *

into new_table_name

From

old_table_name

CTE(COMMON TABLE EXPRESSİON)

Ortak Tablo İfadeleri

Sorguların yürütülmesi anında elde edilmiş olan geçici sorgulardır.

Sadece çalıştığı sorgu bloğunda geçerlidir.

- CTE kullanımıyla üzerinde çalışacağımız veri setini küçülterek gereksiz yüklerden kurtulmuş olmak, okunabilirlik.
  - **Rekürsif sorgularda**, aynı tablo üzerindeki tekrarlı joinler için (**hiyerarşik verileri sorgulamak** gibi)

Kullanım:

Örn:

```
WITH Sales_CTE (SalesPersonID, SalesOrderID,
SalesYear)
AS
```

```
-- sorguyu tanımlıyoruz.
(
   SELECT SalesPersonID, SalesOrderID,
YEAR(OrderDate) AS SalesYear
   FROM Sales.SalesOrderHeader
   WHERE SalesPersonID IS NOT NULL
)
--sorguyu calıstıkmak icin alttaki selecti yazıyoruz.
-- Tümünü çalıştırmak gerekir.Alttaki select'le çalıştırırsan
hata alırsın.Tümünü çalıştırman gerekiyor.
SELECT
SalesPersonID,
COUNT(SalesOrderID) AS TotalSales,
SalesYear
--Toplam Satış
FROM Sales_CTE
--Yukarıdaki CTE tablosunun ismi
GROUP BY SalesYear, SalesPersonID
ORDER BY SalesPersonID, SalesYear;
```

Tek bir sorguda birden çok CTE tanımı kullanma

```
WITH Sales_CTE (SalesPersonID, TotalSales, SalesYear)

AS

 (

   SELECT SalesPersonID, SUM(TotalDue) AS TotalSales,
YEAR(OrderDate) AS SalesYear

   FROM Sales.SalesOrderHeader

   WHERE SalesPersonID IS NOT NULL
```

```
        GROUP BY SalesPersonID, YEAR(OrderDate)
)
Sales_Quota_CTE (BusinessEntityID, SalesQuota,
SalesQuotaYear)
AS
(
        SELECT BusinessEntityID, SUM(SalesQuota)AS
SalesQuota, YEAR(QuotaDate) AS SalesQuotaYear
        FROM Sales.SalesPersonQuotaHistory
        GROUP BY BusinessEntityID, YEAR(QuotaDate)
  )


SELECT
 SalesPersonID
  , SalesYear
  , FORMAT(TotalSales,'C','en-us') AS TotalSales
  , SalesQuotaYear
  , FORMAT (SalesQuota,'C','en-us') AS SalesQuota
  , FORMAT (TotalSales -SalesQuota, 'C','en-us') AS
Amt_Above_or_Below_Quota
```

```
FROM Sales_CTE

JOIN Sales_Quota_CTE ON
Sales_Quota_CTE.BusinessEntityID =
Sales_CTE.SalesPersonID  AND

Sales_CTE.SalesYear=Sales_Quota_CTE.SalesQuotaYear

ORDER BY SalesPersonID, SalesYear;

OPTİON(MAXRECURSION)

--
```

Sonsuz döngüye girmesini engellemek için MAXRECURSION KULLANILIR.

Yani kısır döngüye girdiği zaman döngüyü durdurur.

OVER KULLANIMI

PowerBI'ydaki earlier fonksiyonu gibi çalışır.

Yani satır satır işlem yapar.

1-KULLANIM:

1-Yalnızca OVER() şeklinde kullanılır.

PowerBI'ydaki ALL() fonksiyonuna benzer.

Yani Tüm toplamı tüm satırlarda gösterir.

Örn:

```
select Id,Kategori,Tutar,sum(Tutar)
OVER () as TümToplam from dbo.HesapOzeti
```

Results | Messages

	Id	Kategori	Tutar	TümToplam
1	1	Yakit	-100,00	-150,00
2	2	Alacak	200,00	-150,00
3	3	Kira Alacak	500,00	-150,00
4	4	Giyecek	-150,00	-150,00
5	5	Yakit	-400,00	-150,00
6	6	Yakit	-200,00	-150,00

Açıklama:Burada tüm toplamı satır satır gösterdi.

2-KULLANIM:

Order By ile kullanımı

1-Baştan sona kümülatif toplam

```
select Id,Kategori,Tutar,sum(Tutar) OVER (ORDER BY Id
asc) as kümülatiftoplam from dbo.HesapOzeti
```

Results | Messages

	Id	Kategori	Tutar	kümülatiftoplam
1	1	Yakit	-100,00	-100,00
2	2	Alacak	200,00	100,00
3	3	Kira Alacak	500,00	600,00
4	4	Giyecek	-150,00	450,00
5	5	Yakit	-400,00	50,00
6	6	Yakit	-200,00	-150,00

Açıklama:Burada Id'ye göre azalandan artana sıralama yapar.

Örn:1.satırda sadece 1.satırı alacağı için kendisi gelicektir.

Sonrasında 1.ve 2. Ve 3.satıra bakar toplamını 3.satıra yazar.

Sonrasında 1.ve 2.ve 3.ve 4.satıra bakar toplamını 4.satıra yazar.

Sonrasında 1.ve 2.satıra bakar toplamını 2.satıra yazar.

2.yol baştan sonra kümülatif Toplam

```
SELECT Id,Kategori,Tutar,SUM(Tutar) OVER (ORDER BY
Id ROWS BETWEEN CURRENT ROW AND UNBOUNDED
FOLLOWING) AS Degisim
FROM dbo.HesapOzeti;

--2.yöntem kümülatif toplam baştan Sona=
```

2-SONDAN BAŞA KÜMÜLATİF TOPLAM

```
Select Id,Kategori,Tutar,

Sum(Tutar) OVER (ORDER BY Id desc) as Kümülatiftoplam

 from dbo.HesapOzeti
```

	Id	Kategori	Tutar	TümToplam
1	6	Yakit	-200,00	-200,00
2	5	Yakit	-400,00	-600,00
3	4	Giyecek	-150,00	-750,00
4	3	Kira Alacak	500,00	-250,00
5	2	Alacak	200,00	-50,00
6	1	Yakit	-100,00	-150,00

2.Yol Sondan Başa Kümülatif Toplam

```
SELECT Id,Kategori,Tutar,SUM(Tutar) OVER (ORDER BY
Id ROWS BETWEEN UNBOUNDED FOLLOWING

AND CURRENT ROW) AS Degisim

FROM dbo.HesapOzeti;
```

3-PARTİON BY KULLANIMI:

PowerBI'ydaki AllExcept fonksiyonuna benzer.

Farklı şekillerde kullanılır.

Sütunsal toplama yapar.

Örn:

1-Yüzde bulmak için kullanabiliriz.

```
SELECT
Kategori,
Tutar,
(Tutar*100)/SUM(Tutar) OVER(PARTITION BY Kategori)
AS YuzdeTutar
```

Results | Messages

	Kategori	Tutar	YuzdeTutar
1	Alacak	200,00	100,00
2	Giyecek	-150,00	100,00
3	Kira Alacak	500,00	100,00
4	Yakit	-400,00	57,1428
5	Yakit	-200,00	28,5714
6	Yakit	-100,00	14,2857

Açıklama:Burada ilk önce Tutar*100 bildiğimiz kısım Sum(Tutar) ise yakıttakilerin yakıtta olanların toplamını bulur.

PARTITION İLE LAG VE LEAD KULLANIMI:

Önceki ve Sonraki yılları bulmak için kullanırız.

1-Önceki Yıl(LAG)

```
SELECT Musteri,
Yil,
Tutar,
LAG(Tutar, 1)
OVER (partition by Musteri order by Yil) AS OncekiYilTutar
FROM dbo.Ticaret
```

Musteri	Yil	Tutar	OncekiYilTutar
X	2001	50,00	NULL
X	2003	220,00	50,00
Y	2001	100,00	NULL
Y	2002	130,00	100,00
Y	2003	80,00	130,00

--Açıklama=
Müşteriye göre sütun bazlı gruplama yapar.
Tutar sütunundaki 100.00 üstünde bir rakam olmadıgı icin null döndü.
--lag() 1.parametre hangi sütunda işlem yapacagımız
2.parametre ise kaç hücre sonraki tutarı al anlamına gelir.
1 ise 1 sütun altındaki rakamı alır.2 ise 2 sütun altındaki degeri alır.
Yok ise null döner.
--Partition by diyerekte sütunca gruplama işlemi yaptık. Üst satırındaki degeri alır.

--B): Musterinin sonraki yil Tutar(LEAD)

```
SELECT Musteri,Yil,Tutar,LEAD(Tutar, 1) OVER
(PARTITION BY Musteri ORDER BY Yil asc) AS
SonrakiYilTutar
FROM dbo.Ticaret
```

--lead()

1.parametre hangi sütunda işlem yapacagımız

2.parametre ise kaç hücre sonraki tutarı al anlamına gelir.1 ise 1 sütun üstündeki rakamı alır.2 ise 2 sütun üstündeki değeri alır.

Yok ise null döner.

--lead ve lag fonksiyonlarını kullanmak icin Over(order by) kullanılmak zorundadır.

Musteri	Yil	Tutar	SonrakiYilTutar
X	2001	50,00	220,00
X	2003	220,00	NULL
Y	2001	100,00	130,00
Y	2002	130,00	80,00
Y	2003	80,00	NULL

SIRALAMA FONKSİYONLARI(RANK,DENSE_RANK,ROW_NUMBER)

--RANK fonksiyonu ile tekrar eden satırlara aynı numaralar verilir ve kullanılmayan numaralar geçilir.

--DENSE_RANK fonksiyonunda ise kullanılmayan numaralar geçilmez.

--ROW NUMBER:Tekrar eden kısımlarda Row_Number hiçbirşey ayırt etmeden 1 den başlayarak ardışık sayı döndürür.

```
SELECT
Yil,
 DENSE_RANK() OVER( ORDER BY Yil DESC) AS
Dense_Rankkullanım,
 --Burada Yıla göre sırala demiş oluyoruz.
Dense Rank'ta 2 tane 2003 1.sırada 2 tane 2002 2.sırada
yani düzen bozulmaz.
 RANK() OVER( ORDER BY Yil DESC) AS Rankkullanım,
 --Burada Yıla göre sırala demiş oluyoruz.
Rankta ise 2 tane 2003 sıralamaya riayet etmez. 1. ve
2.satırdan sonra normalde 2 olması gerekirken 2 tane 1 den
sonra
3.satırda 3 olarak gösterir.
5.satırda 5 olarak devam eder.
 ROW_NUMBER() OVER(ORDER BY Yil DESC) AS "Normal
Sıralama"
FROM dbo.Ticaret
```

	Tutar	D_Rankkullanım	Rankkullanım	Normal Sıralama
1	220,00	5	1	1
2	130,00	4	2	2
3	100,00	3	3	3
4	80,00	2	4	4
5	50,00	1	5	5

FARK:

DENSE_RANK:

SELECT Yil,
 DENSE_RANK() OVER(ORDER BY Yil DESC) AS D_Rankkullanım,
 --Burada Yıla göre sırala demiş oluyoruz.
Dense Rank'ta 2 tane 2003 1.sırada 2 tane 2002 2.sırada yani düzen bozulmaz.
 RANK() OVER(ORDER BY Yil DESC) AS Rankkullanım,
 --Burada Yıla göre sırala demiş oluyoruz.

RANK:

Rankta ise 2 tane 2003 sıralamaya riayet etmez.
 Sütundaki Tekrarlı verileri aynı gösterir.
1. ve 2.satırdan sonra normalde 2 olması gerekirken 2 tane 1 şeklinde gösterir.
Sonra
3.satırda 3 olarak gösterir.
5.satırda 5 olarak devam eder.

ROW_NUMBER:

 ROW_NUMBER() OVER(ORDER BY Yil DESC) AS "Normal Sıralama"
--Tekrar eden kısımlarda Row_Number hiçbirşey ayırt etmeden 1 den başlayarak ardışık sayı döndürür.
 FROM dbo.Ticaret

	Yil	D_Rankkullanım	Rankkullanım	Normal Sıralama
1	2003	1	1	1
2	2003	1	1	2
3	2002	2	3	3
4	2001	3	4	4
5	2001	3	4	5

SİSTEM TABLOLARI

1-MasterDB

Sistem konfigürasyonu,kullanıcılar,veritabanları,sistem dosyaları,Collation bilgisi gibi SQL Server sisteminin temel konfigürasyon bilgilerini tutar.

2-ModelDB

Şablon veritabanıdır.

Her bir oluşturulacak veritabanı ModelDB'nin bir kopyası olarak oluşturulur.

-Her veritabanında otomatik olarak olmasını istediğimiz tipler,fonksiyonlar,tablolar vs. varsa bu veritabanının içine konulabilir.

3-MSDB

SQL Server Agent servisinin kullanıldığı veritabanıdır.

Periyodik olarak çalıştırılan her türlü işlem(Joblar,schedule'lar,alertler) burada tutulur.

SYSJOBS=Yapılan jobları gösterir.

Sysjobhistory=İş geçmişini gösterir.

4-TempDB tabloları

2 Çeşit tabloda tutulur.

1-Geçici Tablolar(Temporary Tables)

Kullanımı: #TabloİSmi Şeklinde kullanılır.

Oturum kapatıldıktan sonra veya bir başka Query ekranı açıldığında bu tabloya erişilemez.

SQL Serverda geçici tablolara sadece bulundukları ortamlardan erişilebilir.

SQL kapandıktan sonra tablolar silinir.

2-Genel Geçici Tablolar(Global Temporary Tables)

1-##TabloİSmi şeklinde kullanılır.

Eğer temp table'a global olarak diğer ortamlardan da erişilmesini istiyorsak o zaman global temp table kullanmamız gerekmektedir.

Geçiçi Tabloların Özellikleri(Temprary Table)

- **Saklı yordam (Stored Procedure) içerisinde geçici tablo kullanılabilir**; ancak, kullanıcı tanımlı fonksiyon (User Defined Function - UDF) içerisinde geçici tablo yaratılamaz.
- Normal tablolarda olduğu gibi geçici tablolarda da indeks (index) oluşturabilir ve kimlik (identity) alan tanımlanabilir.
- Geçici tabloların farklı kullanıcılar tarafından aynı anda oluşturulma ihtimalleri vardır. Bu durumda sistem kendilerine benzersiz (unique) bir id ataması yaparak isim çakışmalarını engeller.

Kullanım Amacı:

1-Yani önemli tablolar üzerinde kritik sorgular çalıştırmak zorunda kalındığında ve sonuçlar tahmin edilemeyecek gibiyse, a)Geçici bir tablo oluşturulur ve kodlar bu geçici tablo üzerinde test edilir;

b)Daha sonra istenilen kodlar gerçek tabloya uygulanır.

2-Aynı zamanda geçici tablolar, karmaşık ve çok fazla bilgi olan tablolardan sadece belirli bir kısmı alarak üzerinde çalışmak için de kullanılabilir.

Local Temprorary Table ile *Global Temporary Table* arasındaki Farklar:

1-Yerel geçici tablolar(Local Temprorary Table),

Tabloyu oluşturan kişi SQL Server ile olan bağlantısını kapattığında yok edilir.

2-*Genel geçici tablo*(*Global Temporary Table*) *ise son aktif bağlantı kapatıldığı anda yok edilir.*

Yani geçici tabloyu oluşturan kişi ile birlikte o anda SQL Server'a birden fazla kişi bağlı bulunabilir.

Bu durumda tabloyu oluşturan kişi SQL Server ile olan bağlantısını kapattıktan sonra SQL Server'a bağlı bulunan kimse kalmayana kadar geçici tablo saklanır ve bağlı bulunan en son kişi bağlantısını sonlandırdığında tablo silinir.

Tablo oluşturmak için CREATE TABLE komutunu kullanacağız. Tablo içine veri aktarmak için de INSERT INTO komutunu kullanacağız. CREATE TABLE kullanımının kalıbı aşağıdaki gibidir:

```
CREATE TABLE #TabloAdı (kolon1 veritipi, kolon2 veritipi)
```

```
CREATE TABLE #Musteriler(MusteriID INT, MusteriAd VARCHAR(50),
MusteriSoyad VARCHAR(50));

INSERT INTO
#Musteriler(MusteriID,MusteriAd,MusteriSoyad)
```

```
SELECT Musteri.CustomerID, FirstName, LastName
FROM Person.Person AS Kisi
INNER JOIN Sales.Customer AS Musteri
ON Kisi.BusinessEntityID = Musteri.PersonID;
SELECT MusteriID, MusteriAd, MusteriSoyad FROM
#Musteriler;

DROP TABLE #Musteriler;
```

Ne zaman Temp Tablo Ne zaman Tablo Degiskeni Kullanmaliyiz?

Eger 100 kayittan daha az kayit ile çalisacaksaniz tablo degiskeni kullanin

Diğer türlü temp tablo kullanmalisiniz. Bunun sebebi SQL server tablo degiskeni için istatistik olusturmaz.

Eğer index olusturmaniz gerekiyor ise temp tablo kullanin.

Not: Yani geçici tabloların ömrü SQL Server kapatıldığında zaten sonra ererler.

MAİL GÖNDERME

```
EXEC msdb.dbo.sp_send_dbmail
@profile_name='SQLMAIL'
@recipients='sqlserver.egitim@gmail.com'
@body='hello world'
@query=sorgu kısım
@attach_query_result_as_file=1;
```

TRANSACTİON İŞLEMİ

Çoğu zaman bir transaction yalnızca bir türde işlem yapar, yani sadece veri silme, veri güncelleme veya veri ekleme gibi tek türde işlem yapar.

Transaction, çalışma yapısı olarak

ya bütün işlemleri gerçekleştirir ya da hiçbirini gerçekleştirmez. İşlemlerden biri başarısız olursa, hiçbir işlem gerçekleşmez;

ancak tüm işlemler başarılı olduğunda Transaction, içinde gerçekleşen tüm veri değişikliklerini onaylamış demektir.

ÖNEMLİ:

Hepsi Başarılı Olduğunda;

Transaction bloğundaki işlemlerin hepsi başarılı olduğunda

Transaction Commit (Onaylama) komutu çalışır ve değişiklikler veritabanında gerçekleşmiş olur.

Veritabanında Insert, Update, Delete gibi işlemler başarılı bir şekilde gerçekleştiğinde COMMIT komutu ile değişiklikler kaydedilir.

Herhangi Bir hata varsa;

ancak bir hata varsa işleyiş bozulur ve

Transaction Rollback (Geridönüş) komutu çalışır,

bu şekilde **tüm işlemler(sorgular) geri alınır ve en başa dönülür**. Böylece veri kaybına karşı bir çeşit koruma mekanizması oluşturulmuş olunur.

CAST İŞLEMİ

Veri tipinin değiştirilmesini sağlar.

CAST([KDV_ORANI] as varchar(50))

ERROR(Hata Terimleri)

Sql Server'da tanımlı olan hata mesajlarına sys.messages sistem tablosundan ulaşılabilir.

RaiseErrror Kullanımı:

Uygulamalarımızda meydana gelen hataları devreye almak için kullanılan bir fonksiyondur.

Kullanımı:

RAISERROR('Hata Mesajı', HataSeviyesi, HataDurumu)

HataSeviyesi=Mesajın kritiklik düzeyini ifade eder. 0-25 arası bir değer alabilir. 0-10 aralığı kullanıcının girdiği verilerden kaynaklı hataları, 11-16 aralığı kullanıcının düzeltebileceği türden hataları, tek başına 17 rakamı yetersiz kaynak, disk dolu, tablo okumaya korumalı gibi hatayı, 18 rakamı ölümcül olmayan dahili hatayı, 19 rakamı Sql Server'ın kısıtlarına takılma sonucu oluşan hatayı(bu

hatada WITH LOG özelliğinin kullanılması gerekir), 20-25 aralığı ölümcül(sadece admin'in ekleyebildiği) hataları ifade eder.Buraya genellikle 16 rakamı geçilir..
HataDurumu= 1 - 255 arası değer alabilir ancak raiserror'un hata üretebilmesi için gereken değer 1-127 aralığıdır.

DEĞİŞKEN KULLANIMI

1-Tablodan dönen değeri değişkene atmak için kullanılır.

Örn:

Banka tablosu olsun.

	MusteriNo	MusteriAd	Bakiye	SonİslemTarihi
1	1	ismail	100,00	2020-11-24
2	2	emre	300,00	2020-12-10
3	3	kerem	900,00	2020-03-12

MusteriNo,MusteriAd,Bakiye ve SonİslemTarihi sütunları bulunmaktadır.

Tablodan dönen değişkeni alarak işlem yapalım.

```
declare @Bakiye money
```

Açıklama:

@Bakiye diye Money veri türünde bir değişken tanımlarız.

```
SELECT  @Bakiye=Bakiye FROM BANKA   WHERE
MusteriNo =1
```

Sonrasında Banka tablosundaki müşteriNo=1 olanların bakiyelerini getirir.Birden fazla MusteriNo=1 olan bakiyesi olanlar var ise en son satırdaki değeri gösterir.Tek sütun ve tek satır değer döner.

Çıktısı:

	BAKİYE
1	100,00

Açıklama:Tek satırlık bir sütun değeri döner.

Tablo olarak yaparsak o satırın en son değerini gösterir.

```
Declare @Bakiye
      SELECT  @Bakiye=Bakiye FROM BANKA

         SELECT @Bakiye AS BAKİYE

         select * from BANKA
```

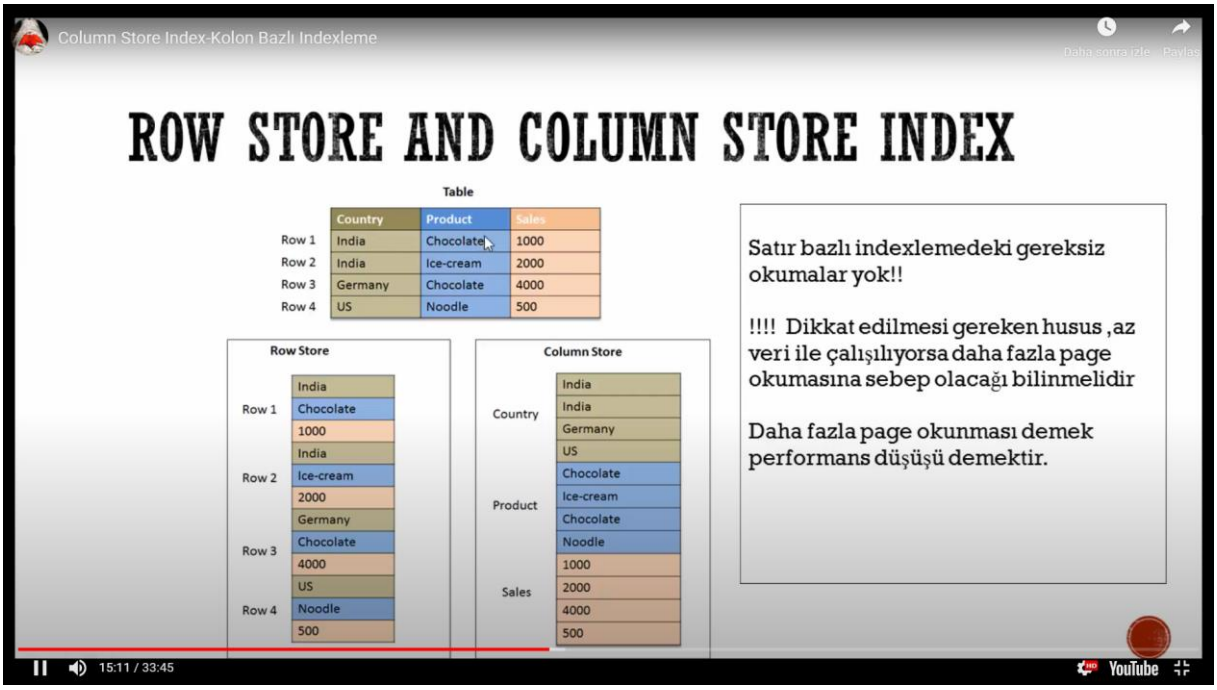
Çıktısı:

Burada Son satırdaki değeri gösterir.

Results | Messa

	BAKİYE
1	900,00

ROW STORE İLE COLUMN STORE FARKLARI



COLUMNSTORE INDEX ÇEŞİTLERİ

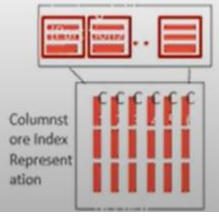
ColumnStore Index Çeşitleri (1/2)

Non-Clustered ColumnStore Index

- SQL Server **2012** versiyonunda duyuruldu.
- Tablo verisine ek alan harcar.
- Diğer indexlerle birlikte kullanılabilir.
- Belirli kolonlar seçilerek indexlenebilir.
- Constraintler ve triggerlar tanımlanabilir. PK,FK kullanılabilir.
- **Tanımlandığı tabloda yalnızca okuma işlemi yapılabilir.**

Clustered ColumnStore Index

- SQL Server **2014** versiyonunda duyuruldu.
- Tüm tablonun veri tutma modunu belirler.
- Diğer indexlerle birlikte kullanılamaz.
- Tüm tablo indexlenir.
- Constarint, Trigger, PK, FK tanımlanamaz.
- **Tanımlandığı tabloda INSERT,UPDATE,DELETE işlemi yapılabilir.**



Microsoft SQL Server 2016 sürümünden sonra

Clustered Indexlerde Primary Key ile Foreign Key tanımlanabilir.

Artık NonColumnIndexlerde

Yazma(insert,update,delete) işlemleri yapılabiliyor.

Örn:Indexlerde yalnızca 1 tane ColumnStoreIndex bulunur.

```
Create table IndexUygulama (
ID int not null ,
Ad varchar(20),
SoyAd varchar(20),
PRIMARY KEY NONCLUSTERED (ID asc)

)
```

Tabloya ekleriz.

CREATE CLUSTERED COLUMNSTORE INDEX IndexAdi ON tabloAdi

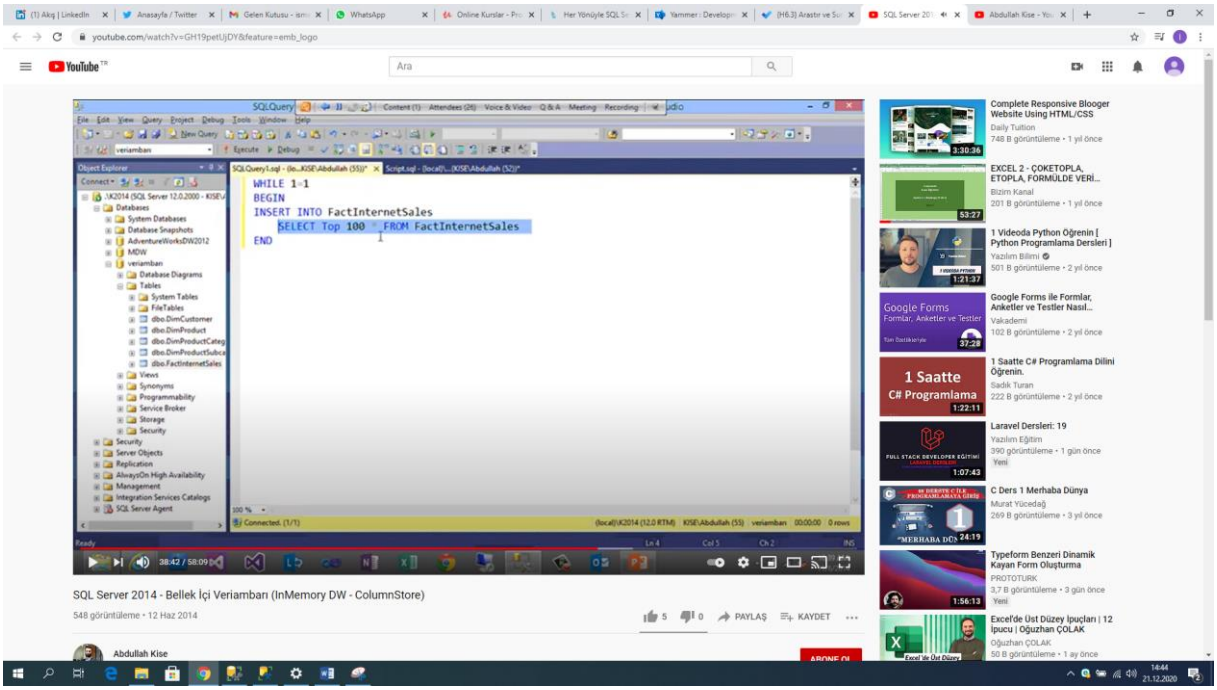
CLUSTERED INDEX=TABLOYU KOMPLE COLUMN STORE'A DÖNÜSTÜRÜYOR.

Verilerin çok olduğu fact'e karşılık gelir.O tip yapılarda yapılsa yeterlidir.

2- Batch Mode işleme:

SQL Server sorgu operatörleri geleneksel olarak **tek seferde bir satır** işlemektedir. Yani sürekli "**row mode**"da çalışmaktadır. Columnstore ile birlikte **tek seferde bir grup(batch) satır** işleyen yeni operatörler getirildi (genellikle 900 satıra kadar çıkar). Bu özellik yoğun hesap gerektiren işlemden geçecek binlerce satırın daha yüksek hızda elde edilmesini sağlamaktadır.
Scan, filter, Project, hash(inner) join ve (local)hash aggregation SQL Server 2012'de "batch mode"u destekleyen sorgu operatörleridir.

Query Optimizer row mode veya batch mode tercihini şu şekilde yapar; binlerce satır filtreleme, birleştirme, gruplama ve yoğun hesapların olduğu işlemlerden geçmesi gerektiğinde batch mode'u tercih eder,
daha az veride ve batch modu
desteklemeyen sorgu operatörleri kullanıldığında row mode'u tercih eder.



İF EXİST VE İF NOT EXİST KULLANIMLARI

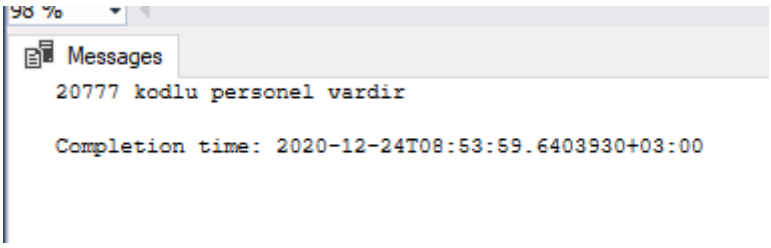
A)İF EXİST İFADESİ

--İF EXISTS İFADESİ if EXISTS(İFADE) içindeki ifadede işlem varmı diye kontrol eden bir yapıdır.

--Var ise True Yok ise False değer döndürür.

```
if EXISTS(select * from Person.Person where
BusinessEntityID='20777')

begin

print '20777 kodlu personel vardır'
end

else
begin

select '20777 kodlu personel yoktur'
end
```

ÇIKTISI:



Açıklama:

Yukarıdaki kod ifadesinde İF EXISTS(ifade) içindeki ifade Select ifadesindeki person tablosundaki BusinessEntityID='20777' ye ait bir kişi varmı demiş olduk

Var ise Print '20777 kodlu personel vardır' yazıcak

Yok ise Select '20777 kodlu personel yoktur' ibaresi yazacaktır.

B)İF NOT EXİST İFADESİ

--İF NOT EXISTS İFADESİ if NOT EXISTS(İFADE) içindeki ifadede işlem yokmu diye kontrol eden bir yapıdır.
--Yok ise True Var ise False değer döndürür.

Örn:

```
if Not EXISTS(select * from Person.Person where
BusinessEntityID='20778')

begin

print '20778 kodlu personel yoktur'
end

else
begin

select '20778 kodlu personel vardır'
end
```

Çıktısı:

```
20778 kodlu personel yoktur

Completion time: 2020-12-24T09:10:57.2610779+03:00
```

Açıklama:

Yukarıdaki kod ifadesinde İF NOT EXISTS(ifade) içindeki ifade Select ifadesindeki Person tablosundaki BusinessEntityID='20778' ye ait bir kişi Yokmu demiş olduk

Yok ise Print '20778 kodlu personel yoktur' yazıcak

Var ise Select '20778 kodlu personel vardır' ibaresi yazacaktır.

FONKSİYONLAR(Drop,Update,Create)

Fonksiyonlar tamamen işimizi kolaylaştırmak adına sürekli olarak tekrarladığımız Sql sorgularına tek bir noktadan erişmemizi sağlar.Satır satır işlem yaparlar.

Buda bize hızlı bir erişim imkanı,hızlı bir hata kontrol mekanizması, çabuk müdahale, sorgu tekrarlamama gibi imkanları verir.

Sql Serverda 4 çeşit fonksiyon kavramı vardır.

1- Scalar-Valued Function

Scalar = Sayısal değer döndüren bir fonksiyondur.

```
Create Function Fn_ToplamaYap(@sayi1 int,@sayi2 int)

Returns İnt  --Dönecek çıktının veri tipi

As

Begin

Declare @toplam int

Set @toplam = @sayi1+ @sayi2

Return @toplam    --Toplam değeri integer döndürür.

End
```

Bu şekilde kullanılan bir fonksiyonu herhangi bir tabloya join ile bağlayabiliriz veya direk select çekebiliriz.

Select dbo.Fn_ToplamaYap(4,7)

Açıklama:Tek bir sütunda tek bir değer olarak 11 çıktısını döndürecektir.

2- TABLE-VALUED FUNCTION

A)

Tablo Döndüren Fonksiyon(SQL de temp tablo yada değişken tablo kullanımı bazen performans sorunlarına yol açabilir. Bu durumda Büyük sorgularda kullanılan tablolar tablo döndüren fonksiyonlarla kullanıldığında performansta artmalara yol açabilmektedir.)

Kullanım:

```
CREATE FUNCTION fonksiyonAdi (varsaParametreler)

RETURNS TABLE

 AS

 BEGIN

RETURN

Select ifadesi
```

END

B)ÇOKLU İFADE İLE TABLO DÖNDÜREN FONKSİYONLAR (Multi-statement)

Bu türden fonksiyonlar bir öncekine benzer. Farkı dışarıya değer döndüren tablo tanımlanması gereklidir.

CREATE FUNCTION fonksiyonAdi (varsaparametreler)
RETURNS @deger TABLE (tablo tanımı)

AS

BEGIN

Sql deyimleri

INSERT INTO @deger selectIfadesi

INSERT INTO @deger selectIfadesi

INSERT INTO @deger selectIfadesi

RETURN END

Örn:

Create function fn_sipList1(@uid tinyint)

```
Returns table

as

Return
select * from siparis where @uid in(1,2)

 Alter function fn_sipList1(@uid tinyint)

returns table

 as

begin

Return select * from siparis where @uid IN(1,2)

END

--test etmek için

select * from dbo.fn_sipList1(1)
```

2- Table-Valued Function

3-Aggragate Function

4-System Function

Kullanıcı Odaklı Fonksiyonlar

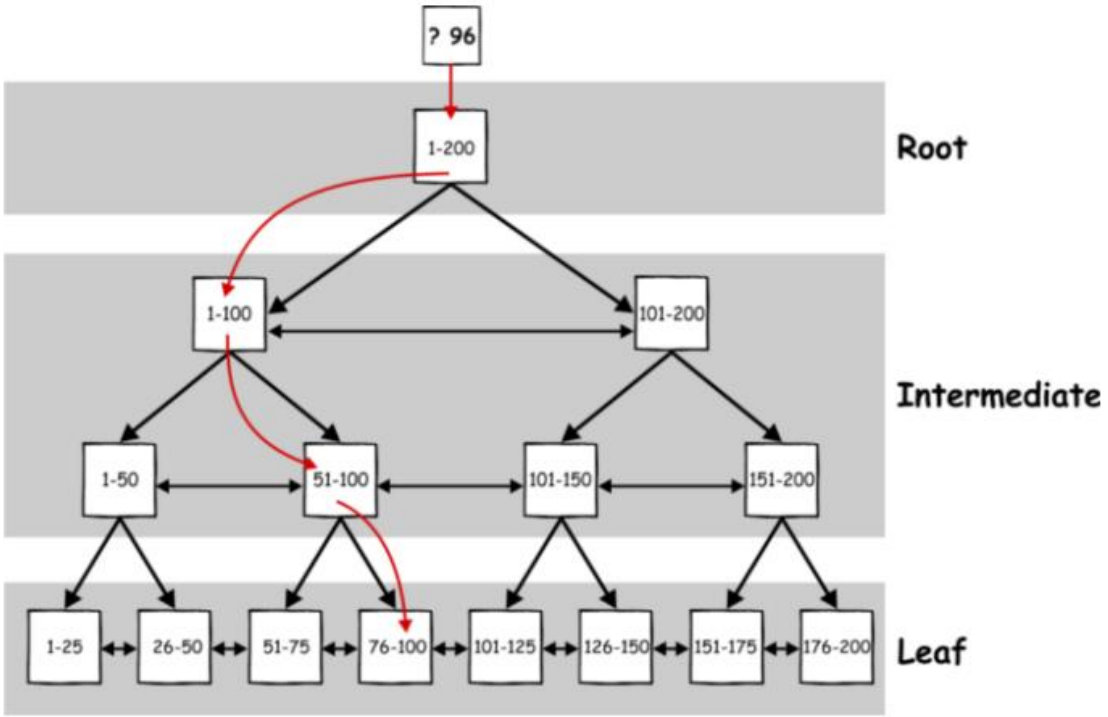
1-Scalar Valued Functions=

```
CREATE FUNCTION fonksiyon_adi
(
  -- Parametrelerin eklendiği yer
  @param1 veritürü,
  @param2 veritürü
)
RETURNS geri_dönecek_değerin_ veri türü(int,string vb.)
AS
BEGIN
  -- Önce Geri dönecek değer tanımlanır.
  DECLARE @donen veritürü
  -- Sql ifadeleri dönen parametreye değer aktarımı gibi
işlemler RETURN @donen
END
```

INDEX KULLANIMI

Sql Server bu dosyaları fiziksel olarak değil mantıksal olarak 8 KB'lık bloklara böler. Bu bloklara **page** denir. Bundan dolayı dosyanın ilk 8 KB'ı page bir sonraki 8 KB'ı page1 olur ve bu şekilde devam eder. Page'lerin içinde ise tablolardaki satırlara benzeyen ve adına **row** denilen yapılar bulunur.

Sql Server page'ler üzerinde başka bir mantıksal gruplama daha yapar; art arda 8 tane page'in bir araya gelmesiyle oluşan 64 KB büyüklüğündeki veri yapısına **extent** denir.



HEAP TABLE(İNDEXSİZ)

Bir tabloya heap denilmesi aslında onun üzerinde bir index tanımlı olup olmamasına bağlıdır.

Heap table üzerinde bir veri arandığında Sql Server tablonun kayıtlarına sırayla erişir ve aradığımız kayıtla eşleştirir.

Kayıt bulunsa bile eşleşebilecek başka kayıt var mı diye tüm kayıtlarda karşılaştırma işlemi yapar.

Tüm satırları gezer.Zaman kaybıdır.

Sql Server'ın yaptığı bu işleme **Table Scan** denir. Bu işlem maliyetli bir işlemdir.Tavsiye edilmez.

Clustered Table

Üzerinde clustered index tanımlı tablolara denir.

Sorgu index tanımlanmış kolonları kullanırsa, veriye çok hızlı erişim sağlanır. Data page'ler veriye hızlı erişim için birbirine bağlıdır.

Fark;

Heap table'lara göre **INSERT**, **UPDATE** ve **DELETE** işlemlerinde ekstra index bakım maliyeti vardır.

Clustered Index

1-Yukarıda da bahsettiğimiz gibi, Clustered index'ler tablodaki veriyi fiziksel olarak sıralar.

2-Bir tablo fiziksel olarak sıralandığından tablo üzerinde sadece bir tane clustered index tanımlanabilir.

3-Clustered index için seçilecek kolon veya kolonlar sorgulardaki en fazla kullanılan kolonlar olmalıdır.

4-Veriler, bu kolonlara göre fiziksel olarak sıralanacağından çok hızlı erişilir.

5-Ayrıca seçilen kolonun çok değiştirilmeyen bir alan olması gerekir.

KULLANIM:

CREATE CLUSTERED INDEX IX_IndexName ON TableName (Column1);

Non-Clustered Index

1-Non-Clustered Index veriyi fiziksel değil mantıksal olarak sıralar.

2-Bu index'lerin leaf node'larında verinin kendisi değil nerede olduğu bilgisi tutulur.

3-Tablo üzerinde en fazla 999 tane non-clustered index tanımlanabilir.

4-Non-clustered index'ler veriye doğrudan erişemez.

5-Heap(indexsiz) üzerinden ya da bir clustered index üzerinden erişebilir.

6-Bu index'i oluştururken sorgumuzun koşul kısmında sık kullandığımız kolonlardan oluşturulması gerekir.

CREATE NONCLUSTERED INDEX IX_IndexName ON TableName (Column1);

Özellikleri:

Bir tabloda en fazla bir tane clustered index, 999 tane de non-clustered olabilir.

Sql Server'da bir index en fazla 16 kolon içerebilir ve toplam boyutu 900 byte'ı aşmaması gerekir.

Ayrıca büyük boyutlu alanlar yani varchar(max), nvarchar(max), xml, text ve image türüne sahip kolonlar üzerinde herhangi bir index tanımlaması yapılamaz.

Maliyetleri de çok fazladır. Her index oluşturduğunuzda veri tabanınızdan bir alan işgal edilir.

Dezavantaj:

Index'lerin **insert**, **update** ve **delete** işlemlerinde tekrardan organize olması gerekir ve bu durum tablo performansını olumsuz etkiler.

Bir tabloda index oluşturulmaya başlandığında Sql Server tabloyu kitler ve erişimi engeller.

Index oluşturma işlemi tablodaki veri sayısına göre kısa veya uzun sürebilir.

Unique Index

1-Verinin tekilliğini sağlamak için kullanılır.

2-Veri tekrarını engeller ve tanımladığımız kolona göre veri çekmeyi hızlandırır.

3-Tablomuza bir primary key veya unique constraint tanımladığımız zaman otomatik unique index tanımlanır.

4-Bu index'i birden fazla kolona tanımladığımız zaman tekillik tek kolon üzerinden değil de tanımlandığı kolonlar üzerinden oluşuyor.

5- Tanımlandığı kolona sadece bir kere **null** değeri eklenebilir. Hem clustered hem de non-clustered index'ler unique olarak tanımlanabilir.

KULLANIM:

CREATE UNIQUE INDEX AK_IndexName ON TableName (Column1);

Filtered Index(NonClustered)

Bu index türünde ise tüm tabloya index tanımlamak yerine, belirlenen koşula uyan veriye index tanımlanır.

Hem performansı arttırır hem de index bakım maliyeti düşük olur. Normal bir non-clustered index'e göre daha az yer kaplar.

Kullanım:

CREATE NONCLUSTERED INDEX IX_IndexName ON IndexName (Column1, Column2) WHERE …

Composite Index(NonClustered)

Tablo üzerinde tanımlanan index tek kolon üzerinden değil de birden fazla kolon üzerinden tanımlandıysa bu index türüne composite index denir.

Bu index tanımında kolonların hangi sırada yazıldığı da çok önemlidir.

Index performansının artması için çeşitliliği fazla olan kolon başa yazılmalıdır.

Yani tablodaki verilere göre tekil(Unique) veri sayısı fazla olan kolon başa yazılır.

KULLANIM:

CREATE NONCLUSTERED INDEX IX_IndexName ON IndexName (Column1, Column2)

Covered Index

Ama index tanımından farklı bir kolon veya kolonları çekmek istediğimiz zaman, öncelikle index koşuluna uyan veriler çekilir ve key değeri belirlenir.

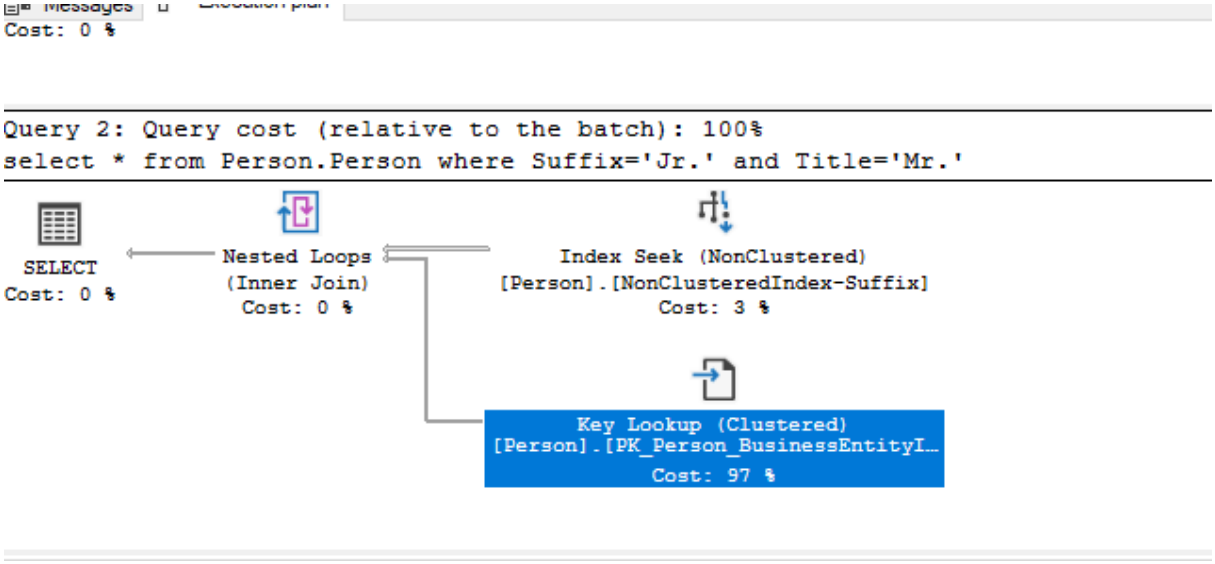
Daha sonra bu key değeri üzerinden index'te tanımlı olmayan kolonların değerlerine erişilir. Bu işleme **key lookup** denir.

Yani istediğimiz filtreledeki alanı indexleyip diğer alanlarda geldiği zaman key_lookup oluşur.

Composite index'te 16 kolon ve 900 byte sınırı olduğundan istediğimiz kadar kolon ekleyemeyiz.

KULLANIM:

CREATE NONCLUSTERED INDEX IX_IndexName ON TableName (Column1) INCLUDE (Column2, Column3);



Composite İndex Özellikleri:

Hatırlarsanız büyük boyutlu veri tutan varchar(max), nvarchar(max), xml, text ve image gibi alanları index tanımına ekleyemiyorduk.

INCLUDE seçeneği ile eklenebilir.

Ayrıca 16 kolon ve 900 byte kısıtı da ortadan kalkmış oluyor.

INCLUDE seçeneği sadece Non-clustered indexlerle tanımlanabilir.

İNDEX TASARIMI

1-Yoğun şekilde veri güncelleme işlemi olan tablolarda, index tanımında mümkün olduğunca az kolon seçmeliyiz.

2-Veri güncellemenin az olduğu tablolarda daha çok index tanımlayabiliriz.

3-Clustered index'i mümkün olduğunca az kolona tanımlamalıyız. İdeal tanımlanma biçiminde clustered index'imiz unique olan kolonda tanımlanmalı ve null değeri içermemeli.

4-Index tanımladığımız kolonda ne kadar tekrarlı veri varsa index performansımız düşük olacaktır.

5-Composite index'lerde kolonların sırasına dikkat etmeliyiz.

- Computed kolonlara da gereksinimleri karşıladıkça index tanımlanabilir. Yani compute edilen değerin deterministik olması gerekir.
- Depolama ve sıralama etkileri nedeniyle index tanımlarında kolonlar dikkatli seçilmelidir.
- Index tanımında kolon sayısı, yapılacak insert, update ve delete işlemlerinde performansı direkt olarak etkileyecektir.

Indexlerin pasif yapılması;

1-Index'i kısa süre için silip tekrar oluşturmamız gerekiyorsa bu seçeneği kullanabiliriz.

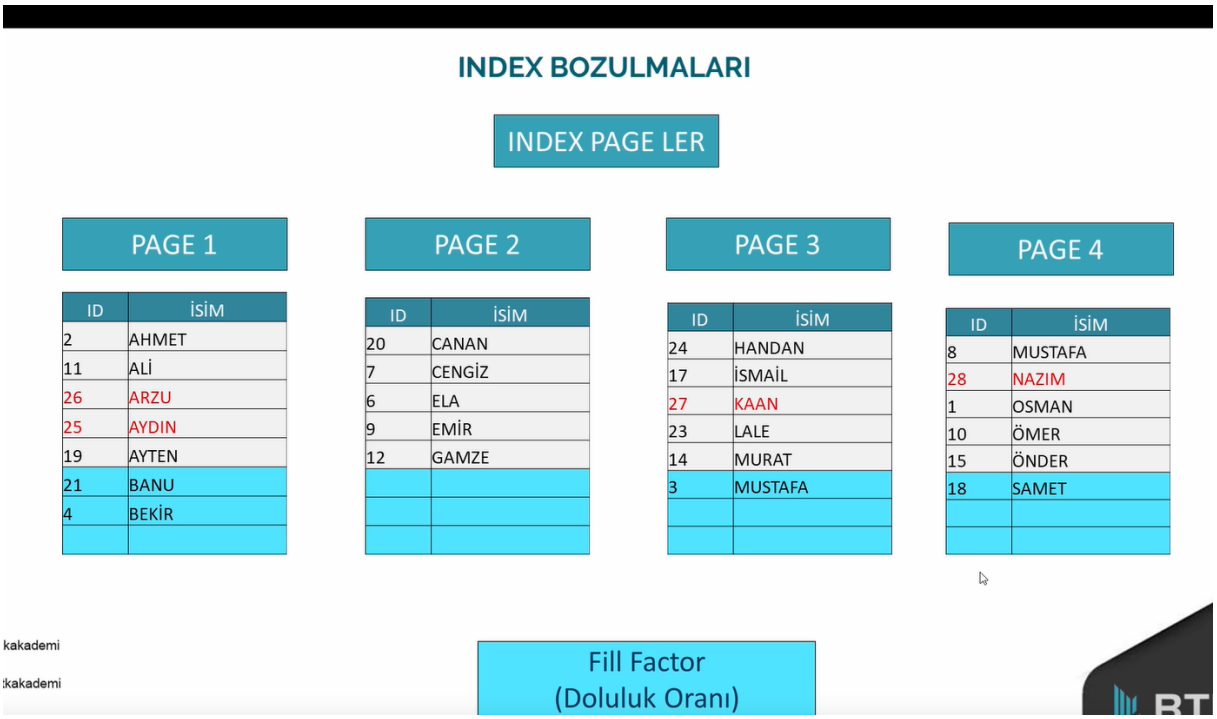
ALTER INDEX INDEX IX_IndexName ON TableName DISABLE

Indexlerin silinmesi; clustered index'ler silindiğinde leaf node'larda tutulan veri, sıralanmamış heap table'larda tutulmaya başlanır. Hem tanımla bilgisi hem de index'in verileri diskten silinir. Primary key olarak tanımlanan clustered index silinemez. İlk önce tablodaki bu constraint kaldırılmalıdır.

DROP INDEX INDEX IX_IndexName ON TableName

Index Seçenekleri

FILLFACTOR: Verilerin tutulduğu leaf node'lardaki data page'lerin yoğunluğunu ayarlamak için kullanılır. Gelen yeni kayıtlar page'lere yazılırken doluluk oranı kontrol edilir, yer varsa ilgili page'e yoksa page ikiye bölünür ve yeni gelecek veri için organize edilir.



Bu pagelerin doluluk oranını FILLFACTOR ile ayarlıyoruz. Varsayılan değer sıfır olup tüm data page'ler doldurulur. Fillfactor'ün büyük tutulması page sayısını arttıracaktır.

Verileri okurken daha çok veriye daha az page okuyarak ulaşırız. Bu değerin az tutulması yani page'lerin boş bırakılması ise insert işleminde bize performans kazandıracaktır.

Her iki durumda da olumlu ve olumsuz yanı olduğundan bu değer iyi hesaplanmalıdır.

PAD_INDEX: Fillfactor değeri leaf node'larda uygulandığını söylemiştik. Bu değerin Intermediate seviyesindeki node'lara da uygulamak istersek fillfactor seçeneği ile birlikte pad_index seçeneğini de kullanmalıyız.

SORT_IN_TEMPDB: Bu seçeneğin aktif edilmesi index işlemlerimizin veri tabanında değil de Tempdb sistem veri tabanında olacağı anlamına gelir.

IGNORE_DUP_KEY: Bu seçenekle unique index tanımlı tabloya aynı kaydı tekrar eklediğimizde hata mesajının seviyesini düşürerek uyarı verilmesini sağlar. Kayıt eklenmez ama işlem bir transaction içinde ise transaction sonlanmaz diğer işlemlere devam edilir.

DROP_EXISTING: Oluşturulmak istenen index adı ile aynı isimde yeni bir index oluşturulmak istediğinde kullanılır. Eski index'i silip yenisini oluşturur.

ONLINE: Index oluştururken tablonun kitlendiğini söylemiştik. Bu seçenek ile index oluşurken tablo kitlenmez ve veriye erişim sağlanır.

MAXDOP: Bu seçenek ile index oluşturma işlemi için sunucumuzdaki işlemcilerden kaç tanesini kullanacağını belirtebiliriz. Bu değer en fazla kullanılacak işlemci değerini belirtir.

DATA_COMPRESSION: Bu seçenek ile özellikle büyük boyutlu index'lerin oluşurken index'imize ait verilerin sıkıştırılmasını sağlar.

EXECUTİON PLAN:

İhtiyacımız olan verilere erişmenin en verimlini yolunu belirlemek için sorguyu analiz eder. Bir sorgu çalıştırıldığında hangi tablolara nasıl erişildiğini ve sorgu bitimine kadar meydana gelen bütün işlemleri ve bunların maliyetlerini gösterir.

Execution planlar **Estimated** ve **Actual** olmak üzere ikiye ayrılır.

1-Estimated execution plan tahmini planı temsil ederken,

2- Actual execution plan ise sorgu sonucunda oluşacak olan gerçek plandır.

Execution planlarla;

1-Hangi index kullanacağımızı,

2-Join operasyonlarının nasıl davranacağını, verinin nasıl sıralanacağı ve gruplanacağını, sorgudaki tabloların hangi sırada işleneceği vb. gibi soruların cevabını bulabiliriz.

Execution planı anlamak performans ayarı için bir ön koşuldur.

SET STATISTICS TIME ON parse, compile ve execute işlemleri için harcanan CPU süresi ve geçen süre hakkında bize bilgi verir.

SET STATISTICS IO ON ise SQL sorgusu tarafından oluşturulan disk etkinliğinin miktarı hakkında bize bilgi verir.

Scan count: Sorguda kullanılan tablolara erişilme sayısını gösterir.

Logical reads: Veri önbelleğinden (data cache) okunan sayfa sayısına karşılık gelir. Logical reads'in azalması daha az server kaynağının kullanılması anlamına gelir ve performans artışı sağlar. Bir sorgunun performansını değerlendirirken bakacağımız en önemli kısım burasıdır.

Physical reads: Diskten okunan sayfa sayısını gösterir. Physical reads sayısı doğrudan server'ın RAM miktarı ile

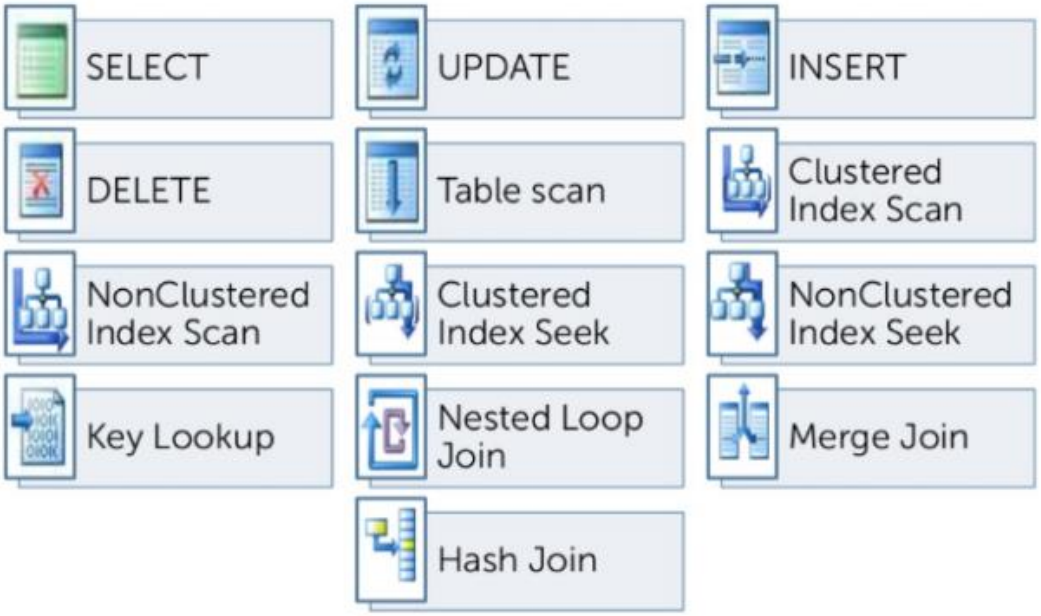
bağlantılı olduğundan SQL sorgu optimizyonu yaparak bunu azaltamayız.

Read-ahead reads: Sorgu için önbelleğe yerleştirilen sayfa sayısıdır.

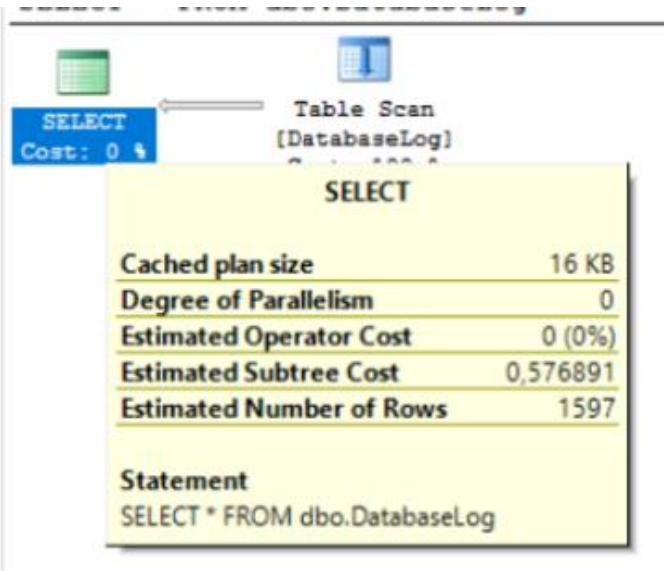
Query Store

Execution planların geçmişini tutan bir yapıdır.

Execution Plan Operatörleri



Select



Cached plan size planımızın cache'te ne kadar yer kapladığını belirtir.

Estimated Operator Cost select işleminin tüm sorgu çalıştığında geçen zamana oranıdır. Burada sıfır olması tüm sorguya oranla önemsenmeyecek kadar zaman aldığı anlamını taşımaktadır.

Estimated Subtree Cost bu adımla birlikte önceki adımlarda toplam harcanan zamanı göstermektedir.

Extimated Number of Rows değeri ise Select işleminde kaç kayıt üzerinde işlem yaptığını gösterir.

Table Scan

Bu tablolar üzerinde bir veri arandığında SQL Server veriyi bulmak için tüm tabloyu satır satır gezer. Yani bu operatör sorgu performansını düşürür.

Phsical Operation sorgu çalıştırıldığında yapılacak işlemi gösterir.

Logical Operation SQL Server tarafından tahmin edilen işlemi gösterir.

Actual Execution Mode ve **Estimated Execution Mode** veriye satır satır ulaşılacağını belirtir.

Storage işlemin gerçekleşeceği storage'ı belirler.

Number Of Rows Read ilgili işlemden etkilenen kayıt sayısını gösterir.

Actual Number of Rows ise ilgili işlemin gerçekleşmesi için okunan satır sayısını gösterir. Bu ikisi arasındaki fark ise 500 satırlı tabloda sorgumuzun koşuluna göre arama yaptığımız zaman kayıtları bulmak için bu 500 satırı da (Number of rows read) gezecektir. Ama sorgu sonucundan 50 kayıt(Actual Number of Rows) dönüyor.

Estimated Operator Cost ise bu işlem için harcanan tüm süreyi ve bu işlemin tüm işlemler içindeki yüzdesini gösterir. **Estimated I/O ve Estimated CPU Cost** değeri ise bu işlem için gerekli I/O ve CPU maliyetini

belirtir. **Estimated Subtree Cost** execution planı sağdan sola doğru okumaya başladığımızda ilgili adıma kadar geçen süreyi gösterir. **Estimated Number of Rows** işlem sonucunda etkilenen kayıt sayısını gösterir. **Estimated Row Size** ise her bir satırın kapladığı alanı belirtir. **Ordered** ise üzerinde işlem yapılan tablonun sıranmış olup olmadığını belirtir. Özellikte büyük boyutlu tablolarla çalıştığımızda bu değer **true** olursa ve tablonun sıralı olması önemli değil ise sorgudan sıralama ifadesini kaldırılmalıdır. **NodeId** değeri ise işlemin gerçekleşme sırasını gösterir. Sıralama soldan sağa yapılır ama execution plan sağdan sola doğru okunur.

Clustered Index Scan

Tablomuzda bir Clustered index tanımlı fakat sorgu koşulu index'i içermediği durumda SQL Server veriye table scan'de olduğu gibi satır satır arama yapar.

Burada dikkat edilmesi gereken nokta operatör adında clustered index olmasına rağmen veriye erişmek için bir index kullanmamasıdır.

Clustered Index Seek

Bu operatör clustered index scan'den farklı olarak veriye bir clustered index üzerinden eriştiğini gösterir. Yani tüm tabloyu gezmek yerine veriye daha az satır gezerek ulaşır.

Index scan işlemine göre daha performanslı çalışır.

Nonclustered Index Seek

Bu operatör ise veriye tüm tabloyu okumak yerine nonclustered index üzerinden eriştiğini gösterir. Sorgu koşulumuzun tanımlanan bir nonclustered index ile eşleştiğinde bu operatör kullanılır.

Key Lookup

Sorguya uyan index kullanıldığında ve bu index çekmek istediğimiz kolonları içermiyorsa key lookup işlemi gerçekleşir.

RID Lookup

Bu operatör primary key olan ama clustered index tanımlanmamış tablolardaki (heap table) verilere ulaşmak istediğimizde ortaya çıkar. Tabloda tanımlı nonclustered index'i kullansa bile data page'lere ulaştığında clustered index tanımlı olmadığından SQL Server primary key üzerinden bir sanal tablo oluşturuyor ve key lookup işlemi yapar.

SQL Server, yazdığımız join ifadelerini execution planda nested loops join, merge join ve hash match operatörlerine çevirir. Şimdi bunları sırayla inceleyelim.

Hash Match

Diğer join türlerine göre hem disk I/O hem de bellek kullanımı daha yüksektir.

Index tanımlanmamış, büyük hash tablelar join işlemine girdiğinde SQL Server bu tabloları birleştirmek için hash join operatörünü kullanır.

İki tablodan küçük olanında hash'leme işlemi yaparak bellekte hash table oluşturur.

Sonrasında büyük tablo taranır ve büyük tablodaki hash değerleri ile bellekte oluşturulan hash table'daki değerler karşılaştırılır. Uyan kayıtlar hash table'da kalır diğerleri silinir ve sonuç döndürülür.

Bu operatörü gördüğümüzde eksik bir index olduğunu veya koşul ifadesinin yanlış olduğu anlamı çıkarabiliriz.

Nested Loops Join

Nested Loops, Hash Match'e göre daha performanslı diyebiliriz ama cpu kullanımı yüksektir.

Tablolardan birine inner diğerine ise outer olarak belirler ve

Outer olan tablodaki her satırı tek tek alıp,

İnner tablosundaki her satır ile karşılaştırır. Eşleşen kayıtları birleştirir ve sonucu döndürür.

Her zaman hash match'e göre iyidir diyemeyiz. Özellikle verinin çok büyük olduğu tablolarda durum değişebilir.

Merge Join

Merge join birleştirilen iki tablonun birleştirme için kullanılan kolonlarının sıralı olması durumunda yani bu kolonlarda index varsa SQL Server tarafından tercih edilen bir operatördür.

SQL Server join işlemi ile karşılaştığında öncelikle birleştirilecek kolonlarda index tanımlı olup olmadığına bakar. Tanımlıysa yani sıralıysa join işlemlerinde en performanslı çalışan merge join operatörünü tercih eder. Tanımlı değilse

SQL Server ya veriyi sıralar ya da diğer join operatörlerinden birini seçer. Bu tercihi de sıralama ve diğer operatörlerin maliyetini değerlendirip çıkan sonuca göre bir seçim yapar.

Hash Match (Aggregate)

Sorgularımız gruplama işlemi var ise SQL Server'ın tercih ettiği operatördür.

Nested loop vs Merge join vs Hash Match

SQL Server bu operatörler arasından sorgumuza göre en iyisini seçer ve bunu seçerken tanımlanmış indexlere, birleştirilecek tabloların satır sayılarına ve join türüne dikkat eder.

- **Nested loop** genellikle bir tablonun çok küçük diğer tablonun büyük olduğu durumlarda seçilir.
- **Merge Join** büyük tablolarda birleştirilecek alanlarda sıralama yani index tanımlı ise tercih edilir. Performansı en iyi join operatörüdür.
- **Hash Match** ise hash işlemi ve hash table kullandığından daha fazla I/O ve bellek tükettiğinden mümkün oldukça index vs tanımlayıp bu operatörden uzak durmalıyız. Performansı en kötü join operatörüdür.