Yereldeki SQL Server’a “.”, “localhost”, “(local)”, “127.0.0.1” ve network ip’sini Server Name olarak kullanarak bağlanabiliriz.

SQL Server’a 127.0.0.1 ile bağlanırken Named Pipes hatası alınıyorsa güvenlik duvarı kapatılmalı, antivirüs kapatılmalı veya SQL Server Configuration Manager üstünden SQL Server Network Configuration🡪Protocols for MSSQLSERVER: Burada görülen Shared Memory aynı bilgisayar üzerinden . localhost local gibi isimlerle yapılan bağlantıyı aktifleştirir. Named Pipes unix lunix için olan bağlantı protokolünü etkinleştirir. TCP/IP ise ip ile bağlantıyı aktifleştirir. Hata alınırsa bunlar aktifleştirilmeli.

Network kullanarak windows authentication yapabilmek için domain yapısı kurulmalı aksi takdirde sql bu bağlantıyı güvenilmez olarak görecektir.

Sadece belirli kod satırlarının çalışmasını istiyorsak onları seçip, execute edebiliriz.

SP\_SPACEUSED ’TabloAdı’ ile tablonun boyutu vs. hakkında bilgi alabiliriz.

**MDF**: Birincil veri dosyasıdır, kullanıcı veri ve nesneleri birincil veri dosyalarında saklanır.

**LDF**: İşlem günlüğü dosyalarının uzantısıdır. Günlük bilgileri tutar, bu bilgiler daha sonra veri tabanını kurtarmak için işe yarayabilir. Her bir veri tabanında en az bir tane işlem günlüğü dosyası bulunmalıdır.

**SQL KOMUTLARINA GİRİŞ**

**SELECT:** Veritabanındaki tablolardan veri çeker.

**UPDATE:** Bir tablodaki kaydın bir ya da daha fazla alanını günceller, değiştirir.

**DELETE:** Bir tablodan kayıt siler.

**INSERT:** Tabloya yeni kayıt ekler.

**TRUNCATE:** Tablonun içini boşaltır.

**CREATE:** Bir veritabanı nesnesi oluşturur.

**ALTER:** Bir veritabanı nesnesinin özelliğini değiştirir.

**DROP:** Bir veritabanı nesnesini siler.

**CREATE DATABASE:** Yeni veritabanı oluşturur.

**ALTER DATABASE**: Bir veritabanı nesnesinin özelliklerini değiştirir.

**CREATE TABLE:** Yeni bir tablo oluşturur.

**ALTER TABLE:** Bir tablonun özelliklerini değiştirir.

**DROP TABLE:** Bir tabloyu tamamen siler.

**CREATE INDEX:** Index oluşturur.

**DROP INDEX:** Index’i siler.

**SELECT**

-- Kullanacağımız database'i seçme.

USE ETRADE

-- \* işareti tüm kolonların isimlerini yazmamak için var.

SELECT \* FROM CUSTOMERS

-- Tablo ismi case sensitive iken columnlar değildir.

-- Boşluklu sütunları [] ile çağırırız ki problem olmasın.

Select ID, CustomerName, [FATHERS NAME] from CUSTOMERS

--------------------------------------------------------------------------------------------------------------------------

**INSERT**

-- ID column'unu otomatik arttırdığımız için ona değer yollayamayız.

-- Değerleri tek tırnakla yollamamız gerekiyor.

INSERT INTO CUSTOMERS(CUSTOMERNAME, CITY, BIRTHDATE, DISTRICT, GENDER) VALUES('ŞAHİZAR KESKİN', 'İSTANBUL', '1947-08-26', 'ÜMRANİYE', 'K')

--------------------------------------------------------------------------------------------------------------------------

**UPDATE**

-- İki tarih arası farkı yıl olarak alma

select DATEDIFF(YEAR,'1999-07-27','2019-08-17')

-- GETDATE şu anki tarihi alır.

select DATEDIFF(YEAR,'1999-07-27',GETDATE())

-- Herkesin yaşını hesapladık.

update CUSTOMERS set NATION='TR', AGE=DATEDIFF(YEAR, BIRTHDATE, GETDATE())

select \* from CUSTOMERS

--------------------------------------------------------------------------------------------------------------------------

**DELETE**

-- Tüm verileri siler, genelde filtreleme yaparak kullanılır.

DELETE FROM CUSTOMERS

--------------------------------------------------------------------------------------------------------------------------

**TRUNCATE**

-- Delete'den farkı tabloyu fabrika ayarlarına döndürmesidir.

-- Örneğin bir tabloyu delete ile boşalttığımızda id'ler kaldığı yerden devam eder.

-- Performans olarak truncate çok daha iyidir.

TRUNCATE TABLE CUSTOMERS

--------------------------------------------------------------------------------------------------------------------------

**WHERE**

-- <> eşit değildir. = - > - < - >= - =< - BETWEEN - LIKE - IN - NOT LIKE - NOT IN gibi parametreleri vardır.

-- Case sensitive değildir

select \* from CUSTOMERS where CUSTOMERNAME='Furkan keskin'

select \* from CUSTOMERS where CITY='İSTANBUL' AND AGE BETWEEN 50 AND 60

-- Ali ile başlayan isimleri verir

select \* from CUSTOMERS where CUSTOMERNAME LIKE 'Ali%'

-- Al ile biten isimleri verir

select \* from CUSTOMERS where CUSTOMERNAME LIKE '%Al'

-- İçerisinde AN geçen isimleri verir

select \* from CUSTOMERS where CUSTOMERNAME LIKE '%AN%'

-- set edilirken case sensitive şekilde set edilir.

UPDATE CUSTOMERS set GENDER= 'ERKEk' where GENDER='E'

--------------------------------------------------------------------------------------------------------------------------

**AND / OR**

-- OR ile birçok sorgudan birine uyan verileri çekebiliriz.

-- AND ile sorgu kriterlerimizi arttırabiliriz.

select \* from CUSTOMERS where CITY='İstanbul'

AND GENDER='erkek' AND AGE>30

OR CITY='Ankara' AND GENDER='erkek' AND AGE>30

--------------------------------------------------------------------------------------------------------------------------

**DISTINCT**

-- Tabloda var olan şehirleri görmek istediğimizde hepsini tek tek görebiliriz.

-- Aynı verileri tekrarlamadan veriyi çeker.

select DISTINCT CITY from CUSTOMERS

-- Tabloda istanbula ait tüm ilçeleri tekil olarak çeker

select DISTINCT DISTRICT from CUSTOMERS where CITY='istanbul'

-- Tabloda istanbula ait tüm ilçeleri tekil olarak çeker ve şehri de gösterir.

select DISTINCT CITY, DISTRICT from CUSTOMERS where CITY='istanbul'

--------------------------------------------------------------------------------------------------------------------------

**ORDERBY**

-- ASC ARTAN DESC AZALAN ŞEKİLDE GETİRİR.

select \* from CUSTOMERS ORDER BY CUSTOMERNAME ASC

-- Önce şehre bakar, şehirler aynıysa isme göre sıralar

select \* from CUSTOMERS order by CITY, CUSTOMERNAME ASC

--------------------------------------------------------------------------------------------------------------------------

**TOP**

-- Şehri istanbul olanlardan id'sine göre ilk 10'u verir.

SELECT TOP 10 \* from CUSTOMERS Where CITY='İstanbul'

**COUNT**

-- Satır sayısını sayar.

select COUNT(\*) from SALES

-------------------------------------------------------------------------------------------

**AGGREGATE FUNCTIONS**

SELECT MIN(AMOUNT),MAX(AMOUNT), COUNT(AMOUNT), SUM(AMOUNT), AVG(AMOUNT) FROM SALES WHERE CITY='istanbul'

**GROUPBY**

-- Markaların yaptığı satış miktarlarının toplamı

SELECT BRAND, SUM(TOTALPRICE) FROM SALES GROUP BY BRAND

-- Şehre göre en az satışı, en çok satışı, kaç ürün satıldığını, toplam kazancı ve ortalama kazancı verir.

SELECT CITY,

MIN(AMOUNT) AS MINPRICE,

MAX(AMOUNT) AS MAXPRICE,

COUNT(AMOUNT) ROWCOUNT\_,

SUM(AMOUNT) AS TOTAL,

AVG(AMOUNT) AS AVERAGE

FROM SALES

GROUP BY CITY

ORDER BY SUM(TOTALPRICE)

-- Convert veri türünü dönüştürür.

-- DATE tarih, Time saat, DATETIME tarih ve saat tutar.

SELECT CONVERT(DATE,DATE\_) AS DATE2,\*

FROM SALES WHERE CITY='ANKARA'

ORDER BY DATE2

-- Tabloya design'dan date2 sütununu ekledik.

-- Ona da tarih ve saat tutan DATE\_ sütunundan tarih kısmını ekledik.

UPDATE SALES SET DATE2=CONVERT(DATE,DATE\_)

-- Ankara'nın günlük toplam satışını aldık.

SELECT CITY,DATE2,SUM(TOTALPRICE) AS TOTALPRICE

FROM SALES WHERE CITY='ANKARA'

GROUP BY CITY,DATE2

ORDER BY CITY,DATE2

-- 2019-07-27'de şehirlere göre toplam kazancı hesapladık.

select CITY,DATE2,SUM(TOTALPRICE) AS TOTALPRICE

FROM SALES WHERE DATE2='2019-07-27'

GROUP BY CITY,DATE2

-- Tarihin ay numarasını verir.

SELECT DATEPART(MONTH,'2019-07-27')

-- Ay rakamı 3 olan verilerin MONTHNAME\_ sütununa mart yazdık.

UPDATE SALES SET MONTHNAME\_='Mart' WHERE DATEPART(MONTH,DATE2)=3

-- Kategoriler her ürün için genelden özele doğru 4 tane

-- Biz de bunlara göre ürünlerin toplam satışlarını aldık.

select CATEGORY1,CATEGORY2,CATEGORY3,CATEGORY4,

SUM(TOTALPRICE) AS TOTAL

FROM SALES

GROUP BY CATEGORY1,CATEGORY2,CATEGORY3,CATEGORY4

-- Bir aggregate where ile sorgulanamaz. Çünkü tabloda kayıtlı değildir.

-- Aggregateler bizlerin geçici olarak oluşturduğu kolonlardır.

-- Ama HAVING ile sorgulanabilir.

select CITY,SUM(TOTALPRICE)

from SALES

GROUP BY CITY

having SUM(TOTALPRICE)>40000

-- Şehre göre müşteri sayılarını alma

select CITY, COUNT(DISTINCT CUSTOMERNAME) AS CUSTOMERCOUNT

from SALES

GROUP BY CITY

ORDER BY CUSTOMERCOUNT DESC

**VERİ TİPLERİ**

**Bigint:** 8 byte yer kaplar. -2^63 ve 2^63 arasında değer alır.

**İnt:** 4 byte yer kaplar. -2^31 ve 2^31 arasında değer alır.

**Smallint:** 2 byte yer kaplar. -2^15 ve 2^15 arasında değer alır.

**Tinyint:** 1 byte yer kaplar. 0 ve 255 arasında değer alır.

**Bit:** 0 ve 1 arasında değer alır.

**Decimal:** Kapladığı yer 5 byte ile 17 byte arasında değişir. -10^38+1 ve 10^38-1 arasında değer alır.

**Money:** 8 byte yer kaplar. -992,337,203,685,477.5808 ve 992,337,203,685,477.5807 arasında değer alır.

**Smallmoney:** 4 byte yer kaplar. -214,748.3648 ve 214,748.3647 arasında değer alır.

**Float:** 7 basamakta 4 byte, 15 basamakta 8 byte yer kaplar. Ondalık sayılar için kullanılır.

**Real:** 4 byte yer kaplar. Virgülden sonra 4 karakter tutar. Çok kullanılmaz.

**Char:** Her bir karakter 1 byte yer kaplar. Sınırını biz belirleriz. Girilen karakterleri boşlukla sınıra tamamlar. Tc, tel no gibi sabit uzunluktaki alanlar için char kullanılmalı. Karakter sayısı sabit olduğu için performans sağlar.

**Nchar:** Girilen karakterleri boşlukla tamamlamaz. N harfi nedeniyle unicode’u destekler. Ayrıca her karakter N harfi nedeniyle 1 değil 2 byte yer kaplar. Bu da uluslararası karakterler demektir.

**Ntext:** Girilen karakterleri boşlukla tamamlamaz. N harfi nedeniyle unicode’u destekler. Ayrıca her karakter N harfi nedeniyle 1 değil 2 byte yer kaplar. Artık kullanılmaz. Eski veritabanlarına uyum için tutulur.

**Text:** Girilen karakterleri boşlukla tamamlamaz. Artık kullanılmaz. Eski veritabanlarına uyum için tutulur.

**Nvarchar:** Girilen karakterleri boşlukla tamamlamaz. N harfi nedeniyle unicode’u destekler. Ayrıca her karakter N harfi nedeniyle 1 değil 2 byte yer kaplar.

**Nvarchar(MAX):** Girilen karakterleri boşlukla tamamlamaz. N harfi nedeniyle unicode’u destekler. Ayrıca her karakter N harfi nedeniyle 1 değil 2 byte yer kaplar. MAX sayesinde 2 gb’a kadar metin tutabilir.

**Varchar:** Girilen karakterleri boşlukla tamamlamaz. Char’ın boşluk bırakmayanı.

**Varchar(MAX):** Girilen karakterleri boşlukla tamamlamaz. Char’ın boşluk bırakmayanı. MAX sayesinde 2 gb’a kadar metin tutabilir.

**Date:** 4 byte yer kaplar. Sadece tarih tutar. 0 ile 9999.12.31’e kadar veri tutar.

**Smalldate:** 3 byte yer kaplar. Sadece tarih tutar. 1900 ile 2079 yılına kadarki verileri tutar.

**Datetime:** 8 byte yer kaplar. Tarih ve saat tutar. 1753 ile 9999 arasına kadarki tarih ve saati tutar. Saati en fazla milisaniyeye kadar tutar.

**Datetime2:** 6 ile 8 byte arası yer kaplar. Tarih ve saat tutar. Saati milisaniyeden sonra 4 basamak daha tutar. 0 ile 9999 arası veri tutar.

**Datetimeoffset:** 1 ile 10 byte arasında yer kaplar. Tarih, saat ve zaman dilimini tutar. 0 ile 9999 arası veri tutar. Çok kullanılmaz.

**Time:** 5 byte yer tutar. Milisaniyeden sonra 4 basamak daha tutar.

**İmage:** Resim dosyalarını tutmak amacıyla kullanılan bir veri tipiydi. Artık pek kullanılmıyor. 2 gb dosya tutmaya izin verir.

**Binary:** 8000 byte’a kadar binary değeri tutar.

**Varbinary:** 8000 byte’a kadar binary değeri tutar. Sonuna boşluk karakter atmaz.

**Varbinary(MAX):** İmage’in yeni halidir.

**Sql\_variant:** Bazı veri tiplerini saklamak için kullanılır.

**Xml:** Xml veriler için kullanılır.

**Table:** Sonradan kullanım amacıyla bir sonuç kümesini saklamak amacıyla kullanılır.

**Uniqueidentifier:** Dünyada eşi benzeri olmayan bir id verir.

**Hierarchyid:** Hiyerarşik yapılarda hiyerarşi yapısını tutar.

**Geography:** Dünyadaki eğimleri de hesaba katarak dünyadaki koordinatları tutar.

**Geometry:** Dünyadaki eğimleri hesaba katmadan koordinatları tutar. 2 düzlem üzerinden hesaplanır.

**İLİŞKİSEL VERİ TABANI KAVRAMI(RDMS)**

**Tablolar arası ilişki kurmak için örneğin;**

Cities Table > Sağ click > Relationshipes > Add

Oluşturulan relationship’i düzenleyerek relationship ekleyeceğiz.

Tables and Columns’un en sağındaki üç noktaya bas açılan pencerede;

Foreign key table bağlanacak key’in tablosu, primary key table ise bağlanılacak key’in tablosu

Foreign key kısmından Countries tablosunu seçiyoruz

Aşağıda her iki tablodaki bağlamak istediğimiz keyleri seçip ok diyoruz.

**INSERT and UPDATE Specification**

No Action iken bir city countryid’ye bağlıysa o country’i silmeye izin vermez.

Cascade iken country’i silmeye izin verir ve o country’ye bağlı şehirleri de siler.

Set Null iken country’i siler, şehri silmez ama bağlı olduğu country id’i null yapar.

Set Default iken country’i siler, şehri silmez ama bağlı olduğu country id’i null yapar. Default’un olayı Concrete tanımlarken default değer atanırsa onu atar.

**JOIN**

-- USERS tablosundan her şeyi , ADDRESS tablosundan ADDRESSTEXT'i getir.

SELECT USERS.\*, ADDRESS.ADDRESSTEXT FROM

USERS,ADDRESS

WHERE USERS.ID=ADDRESS.USERID

AND USERS.ID=1

-- USERS tablosundan NAMESURNAME'i, EMAIL'i, GENDER'ı, BIRTHDATE'i ve ADDRESS tablosundan ADDRESSTEXT'i getir.

-- USERS U ve ADDRESS A ile bu tablolara bu harfleri atadık. Buna alias denir. Kodu temiz hale getirmek için kullanılır.

-- Alias'ı kolonlara da uygulayabiliriz. Alias uygularken as kullanıp, kullanmamak fark yaratmaz.

SELECT U.NAMESURNAME as ADSOYAD, U.EMAIL MAILADRESI,U.GENDER CINSIYET,U.BIRTHDATE DOGUMTARIHI , A.ADDRESSTEXT as ACIKADRES

FROM USERS as U,ADDRESS A

WHERE U.ID=A.USERID

-- Birden fazla tabloyu birbiriyle bağlayabiliriz.

SELECT U.NAMESURNAME as ADSOYAD, U.EMAIL MAILADRESI,U.GENDER CINSIYET,U.BIRTHDATE DOGUMTARIHI , A.ADDRESSTEXT as ACIKADRES,

C.COUNTRY AS ULKE, CT.CITY AS SEHIR,T.TOWN ILCE, D.DISTRICT MAHALLE, A.POSTALCODE AS POSTAKODU

FROM USERS as U,ADDRESS A, COUNTRIES C, CITIES CT, TOWNS T, DISTRICTS D

WHERE U.ID=A.USERID AND C.ID=A.COUNTRYID AND CT.ID=A.CITYID

AND T.ID=A.TOWNID AND D.ID=A.DISTRICTID AND CT.CITY LIKE 'İZMİR'

ORDER BY U.NAMESURNAME

-- İkisi aynı sonucu verir. Tekrar edenleri saymaz.

SELECT DISTINCT U.NAMESURNAME

FROM USERS as U,ADDRESS A, COUNTRIES C, CITIES CT, TOWNS T, DISTRICTS D

WHERE U.ID=A.USERID AND C.ID=A.COUNTRYID AND CT.ID=A.CITYID

AND T.ID=A.TOWNID AND D.ID=A.DISTRICTID

SELECT U.NAMESURNAME

FROM USERS as U,ADDRESS A, COUNTRIES C, CITIES CT, TOWNS T, DISTRICTS D

WHERE U.ID=A.USERID AND C.ID=A.COUNTRYID AND CT.ID=A.CITYID

AND T.ID=A.TOWNID AND D.ID=A.DISTRICTID

GROUP BY U.NAMESURNAME

-- İsimleri ve adres sayılarını verir. Tekrar edenleri saymaz.

SELECT U.NAMESURNAME, COUNT(A.ID) AS ADRESSAYISI, COUNT(DISTINCT C.ID) ULKESAYISI, COUNT(DISTINCT CT.ID) SEHIRSAYISI

FROM USERS as U,ADDRESS A, COUNTRIES C, CITIES CT, TOWNS T, DISTRICTS D

WHERE U.ID=A.USERID AND C.ID=A.COUNTRYID AND CT.ID=A.CITYID

AND T.ID=A.TOWNID AND D.ID=A.DISTRICTID

GROUP BY U.NAMESURNAME

HAVING COUNT(CT.ID) >2

**-- Bu teknikler eski teknikler ve bazı yetersizlikleri olan teknikler olduğu için bu teknik yerine join kullanılır.**

SELECT U.NAMESURNAME, U.EMAIL, A.ADDRESSTEXT, C.COUNTRY, CT.CITY, T.TOWN, D.DISTRICT FROM

USERS U

JOIN ADDRESS A ON A.USERID= U.ID

JOIN COUNTRIES C ON C.ID=A.COUNTRYID

JOIN CITIES CT ON CT.ID=A.CITYID

JOIN TOWNS T ON T.ID=A.TOWNID

JOIN DISTRICTS D ON D.ID= A.DISTRICTID

**Inner Join:** Her iki tarafın kesişimi olan kayıtları verir. Inner Joini USERS.ID ve ADDRESS.USERID üzerinde yaptıysak, bunların her iki taraftada bulunan kayıtlarını alırız. INNER JOIN ve JOIN aynı şeydir.

SELECT U.\*, A.ADDRESSTEXT FROM

USERS U

INNER JOIN ADDRESS A

ON U.ID= A.USERID

**Left Join:** Sol taraftaki kümenin tamamını getirir ve sağ tarafta bunlarla kesişen kayıt varsa onu da getirir. Karşılığı olmayan değerler NULL olarak gösterilir.

SELECT U.\*, A.ADDRESSTEXT FROM

USERS U

LEFT JOIN ADDRESS A

ON U.ID= A.USERID

**Right Join:** Sağ taraftaki kümenin tamamını getirir ve sol tarafta bunlarla kesişen kayıt varsa onu da getirir.Karşılığı olmayan değerler NULL olarak gösterilir.

SELECT U.\*, A.ADDRESSTEXT FROM

USERS U

RIGHT JOIN ADDRESS A

ON U.ID= A.USERID

**Full Join:** Her iki tabloda birden ne var ne yoksa getirir. İlişkili olup, olmamaları önemli değildir. Karşılığı olmayan değerler NULL olarak gösterilir.

SELECT U.\*, A.ADDRESSTEXT FROM

USERS U

FULL JOIN ADDRESS A

ON U.ID= A.USERID

**SUBQUERY**

-- ÜRÜNLERİN FİYAT ANALİZİ

-- JOIN'DEN PERFORMANS OLARAK FARKINI GÖRMEK İÇİN

-- JOIN KULLANILABİLİYORSA JOİN KULLANILMALI

SET STATISTICS IO ON

SELECT ITM.ITEMCODE AS URUNKODU,

ITM.ITEMNAME AS URUNADI,

(SELECT MIN(UNITPRICE) FROM ORDERDETAILS WHERE ITEMID=ITM.ID) AS ENDUSUKFIYAT,

(SELECT MAX(UNITPRICE) FROM ORDERDETAILS WHERE ITEMID=ITM.ID) AS ENYUKSEKFIYAT,

(SELECT AVG(UNITPRICE) FROM ORDERDETAILS WHERE ITEMID=ITM.ID) AS ORTALAMAFIYAT,

(SELECT SUM(AMOUNT) FROM ORDERDETAILS WHERE ITEMID=ITM.ID) AS TOPLAMADET

FROM ITEMS ITM

ORDER BY URUNADI

**STRING İŞLEMLERİ**

-- ASCII VE CHAR FONKSIYONLARI

SELECT ASCII ('A')

SELECT CHAR ('65')

-- SUBSTRING

SELECT SUBSTRING('FURKAN KESKIN',1,6)

SELECT \* FROM USERS WHERE SUBSTRING(USERNAME\_,1,1)='F'

SELECT LEN('FURKAN')

-- CHARINDEX

-- BULUNCA DURUR

SELECT CHARINDEX('A', 'FURKAN KESKİN BURADA',1)

-- CONCAT

-- STRİNGLERİ BİRLEŞTİRİR.

SELECT 'FURKAN'+' '+'KESKİN'

SELECT CONCAT('FURKAN',' ','KESKIN')

SELECT CONCAT\_WS('-','FURKAN','KESKIN','BURADA','DURUYOR')

-- FORMAT

SELECT FORMAT(GETDATE(),'d','en-us')

SELECT FORMAT(GETDATE(),'D','en-us')

-- LEFT, RIGHT, LEN

SELECT LEFT('FURKAN KESKIN', 4)

-- FURK

SELECT RIGHT('FURKAN KESKIN',4)

-- SKIN

SELECT LEN('FURKAN KESKIN')

-- TRIM, RTRIM, LTRIM

-- BOŞLUKLARI TEMİZLER

-- ORTADAKİ BOŞLUĞA DOKUNMAZ

SELECT TRIM(' FURKAN KESKIN ')

--LOWER, UPPER, REVERSE, REPLİCATE

SELECT LOWER('FURKAN')

SELECT UPPER('Furkan')

SELECT UPPER(REVERSE('furkan'))

-- 2 kere furkan yazar

SELECT REPLICATE('Furkan',2)

-- REPLACE

-- Neyi değiştireceksin, neresini değiştireceksin, ne ile?

SELECT REPLACE('FURKAN KESKIN','FURKAN','URKAN')

-- STRING\_SPLIT

-- TABLO DÖNDÜRDÜĞÜ İÇİN SELECT \* FROM KULLANDIK.

-- FURKAN VE KESKİNİ 2 FARKLI SATIRDA GÖSTERİR.

DECLARE @KELIME AS VARCHAR(100) = 'FURKAN-KESKİN'

SELECT \* FROM string\_split(@KELIME,'-')

**SİSTEM DATABASELERİ**

**Master DB:** Sistem konfigürasyonu, kullanıcılar, veritabanları, sistem dosyaları, collation bilgisi gibi SQL Server sisteminin temel konfigürasyonlarını tutar.

**Model DB:** Şablon veritabanıdır. Her oluşturulan yeni db model db’nin bir kopyasıdır. Özelleştirilebilir.

**MSDB:** SQL Server Agent servisinin kullandığı veritabanıdır. Periyodik ve otomatik olarak çalışan her türlü işlem burada bulunur. Sistem bakımı, raporlama vs.

**TempDB:** Geçici tabloların oluşturulduğu işlemler burada gerçekleştirilir. Aggregation işlemleri gibi. Temporary table yaratırken CREATE TABLE #TABLOADI bla bla şeklinde oluşturulur. Bu tablo sadece o anki session için kullanılır. Eğer tüm sessionlar için kullanılması isteniyorsa ##TABLOADI şeklinde oluşturulmalı. Bu tablolar tempDB içerisinde tutulur.

**TSQL KODLARI**

Bir programlama dilinde yapılabilecek neredeyse her şeyi yapabilen bir dil.

**DDL Kodları**

Create Table, Alter Table, Drop Table, Drop Column, Alter Column gibi standart sql kodlarıdır.

**Değişken Tanımlama**

Declare @DeğişkenAdı AS VARCHAR(100)=’Furkan’ şeklinde tanımlanır.

Select @DeğişkenAdı komutu değişken içerisindeki değeri verir. Yoksa NULL verir.

SET @DeğişkenAdı = ‘Furkan’ şeklinde de değer atanabilir.

Değişkenler arasında matematiksel işlemler yapılabilir.

Tablodan değer atanabilir.

**Date Time Function**

-- DATEADD

Declare @Tarih as datetime

set @Tarih = '2019-01-01 16:21:37'

Declare @Tarih2 as datetime

-- Tarih'in 6 ay sonrasına gittik.

set @Tarih2 = DateADD(Month,6,@Tarih)

select @Tarih, @Tarih2

-- DATEDIFF

-- Tarih farkını alır.

Declare @Tarih as datetime

set @Tarih = '2019-01-01 16:21:37'

Declare @Tarih2 as datetime

set @Tarih2 = '2020-07-27 16:21:37'

Select DATEDIFF(YEAR, @Tarih,@Tarih2)

Select DATEDIFF(DAY, @Tarih,@Tarih2)

Select DATEDIFF(HOUR, @Tarih,@Tarih2)

Select DATEDIFF(WEEK, @Tarih,@Tarih2)

-- DATEFROMPARTS

declare @tarih as datetime

set @tarih =DATEFROMPARTS(2020,8,21)

select @tarih

-- DATEPART

-- Bir tarih değişkeninin istenen kısmını verir.

Declare @tarih as datetime

set @tarih ='2019-07-27 17:23:45'

select DATEPART(month,@tarih)

select DATEPART(DAY,@tarih)

-- Haftanın gününün başlangıcını pazar olarak alır.

select DATEPART(DW,@tarih)

-- GETDATE

-- Tarihi alır

select GETDATE()

DECLARE @tarih as date

set @tarih = '1999-07-27'

select DATEDIFF(DAY,@tarih,GETDATE())

**Döngüler**

For döngüsü yoktur. Bu işleri while döngüsüne yaptırırız.

Declare @I as integer = 0

WHILE @I<10

BEGIN

SET @I+=1

WAITFOR DELAY '00:00:01'

SELECT @I

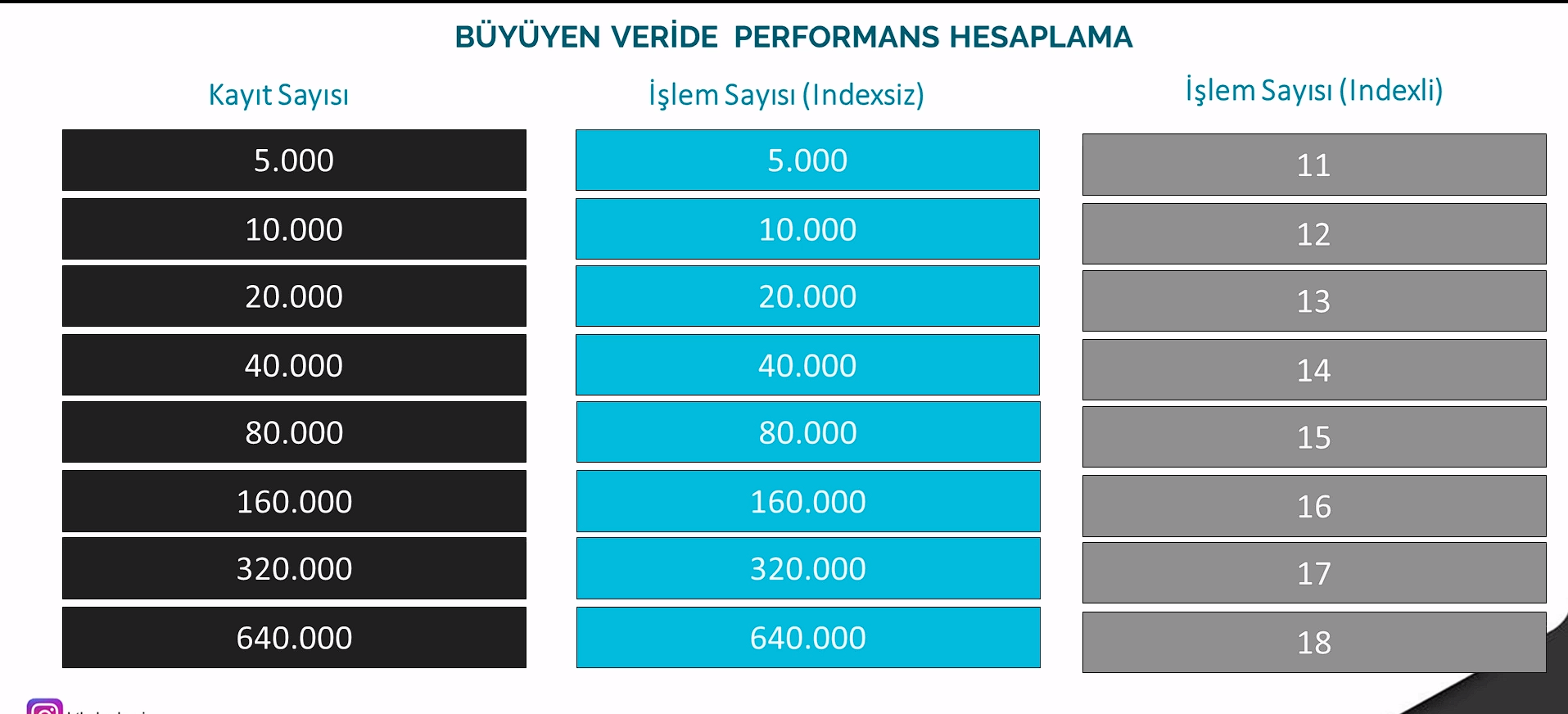
-- Mesaj olarak değerleri yazar.

PRINT @I

END

RAND ve ROUND fonksiyonları da bulunur.

**INDEX**



-- Veri tabanında aranan veriyi bulmak için kayıt sayısını 1’i bulana kadar 2’ye bölüyoruz. Bu bize işlem sayısını veriyor.

-- Sql’in bize verileri hangi arama türüyle getirdiğini ctrl+l ile sorabiliriz.

**Table Scan:**Tabloda primary key yoksa uygular. Rastgele verileri sayfalara yazarak rastgele sayfa seçer. Bu sayfadan teker teker bir veriyi arar. En kötü, en verimsiz arama şeklidir. Bir sayfa 8 kb’dir.

**Clustered Index Scan:** Primary key’e göre sıralar. Primary key id olduğunu düşünürsek isim aradığında yine verilere tek tek bakması gerekir.

**Index Seek ve Key Lookup:** Önce aranan veriyi buluyor. Eğer sadece bu verinin bulunduğu alan istenirse sadece index seek kullanılır ama başka kolonlar da çekilecekse bu sefer index seek yapıp, id’yi alır ve o id’ye ait istenen kolonları getirir. Işlem sayısını 2’ye katlar.

Bu iş yükünden kurtulmak için indeximizin propertiesinden included column olarak diğer tüm kolonları eklemeliyiz. Bunu yaparsak database’in kapladığı alan artar.

SET STATISTICS IO ON

SELECT \* FROM USERS

WHERE NAMESURNAME = 'Mina Kalo'

CREATE NONCLUSTERED INDEX INDEXADI ON USERS

(

NAMESURNAME

)

Satırı index eklenmeden önce 153 page okurken, index eklenince sadece 4 page okuyarak sonucu buldu.

set STATISTICS io on

SELECT \* FROM CUSTOMERS

WHERE TELNR='03423344747'

AND NAMESURNAME='ÖMER ÇOLAKOĞLU'

Birden fazla index eklenmişse SQL en hızlı olacağını düşündüğü indexi kullanır.

İndexlerde kayıt ararken bir kaydı bulsa bile listenin sonuna kadar aramaya devam eder. Bu yüzden tc no, tel gibi sütunlara index eklerken bunları Unique olarak işaretlersek aradığımız kayda ulaştığı an aramayı durdurur. Ayrıca bu sütunlarda var olan kaydın bir kopyasını eklememizi engeller.

**Fragmantation:** İndexler oluşturulduğu zaman örneğin ad kısmını sıralar ama daha sonra eklenen veriler sırasız olarak tutulacaktır. Bu bozukluğa fragmantation denir. Index properties’inden % kaç olduğu görüntülenebilir. Bu değer %40-50 arasına geldiğinde rebuild yapılması gerekir.

**Rebuild:** Fragmantation’u düzeltmek için yapılan işleme rebuild denir. Sistem bakımı sırasında indexleme işlemi tekrar yapılır. Genelde sistem kapalıyken yapılır.

Sürekli çalışan sistemlerde bu işlemi yapmak zor olacağından Sql Pagelerinde boşluklar bırakılır. Örneğin bir page 8 veri tutacaksa içerisine 5 veri girilir ve daha sonra eklenen veriler uygun oldukları page’e yazılır. Buna fill factor denir. Tek bir page içerisinde kaydırma yapmanın maliyeti az olacağı için kaydırma da yapılabilir.



**Reorganize:** Rebuild her şeyi baştan yapar, reorganize ise masanın üstündeki kağıtları tek tek düzeltmeye benzer. Hata payı daha yüksektir ve daha uzun sürer. %40 altında Reorganize kullanılabilir. Üzerindeyse Rebuild yapılmalı. İş garanti olsun bence rebuild yap.

Index klasörüne sağ tık atılarak rebuild veya reorganize yapılabilir. Klasör içerisindeki indexlere de sağ tıklayıp da bu işlem yapılabilir.

**Fill Factor:** İndex properties>options>fill factor’e yüzde kaç doluluk istiyorsak yazarak ayarlayabiliriz.

CREATE NONCLUSTERED INDEX INDEXADI ON USERS

(

NAMESURNAME

)With (Fillfactor=70)

Şeklinde de eklenebilir. Sunucu bazında bir şey eklenmek isteniyorsa Sunucu adı>properties>database settings’den eklenebilir.

-- WITH INDEX ile seçeçeği yolu söyleriz.

-- Tavsiye edilmez.

SELECT \* FROM CUSTOMERS WITH(INDEX=IX1)

WHERE NAMESURNAME='FURKAN KESKİN'

AND BIRTHDATE='1966-12-23'

-- SQL server aramalarda hangi indexi kullanacağına

-- istatistiklere bakarak karar verir. bu istatistikleri

-- güncellemek için 3 yol vardır.

-- Sunucuda bulunan her tabloyu günceller.

SP\_UPDATESTATS

-- Belirli tabloyu güncelleme

UPDATE STATISTICS CUSTOMERS

VIEW

SELECT

U.USERNAME\_ KULLANICIADI, U.NAMESURNAME ADSOYAD, C.COUNTRY ÜLKE, CT.CITY ŞEHİR, T.TOWN İLÇE,

O.ID SİPARİŞİD, O.DATE\_ SİPARİŞTARİHİ, ITM.ITEMCODE ÜRÜNKODU, ITM.ITEMNAME ÜRÜNADI, ITM.BRAND MARKA,

OD.AMOUNT MİKTAR,OD.UNITPRICE BİRİMFİYAT, OD.LINETOTAL SATIRTOPLAMI, A.ADDRESSTEXT ADRES

FROM

ORDERDETAILS OD

INNER JOIN ORDERS O ON OD.ORDERID=O.ID

INNER JOIN ITEMS ITM ON OD.ITEMID=ITM.ID

INNER JOIN USERS U ON O.USERID=U.ID

INNER JOIN ADDRESS A ON O.ADDRESSID=A.ID

INNER JOIN COUNTRIES C ON A.COUNTRYID=C.ID

INNER JOIN CITIES CT ON A.CITYID=CT.ID

INNER JOIN TOWNS T ON A.TOWNID=T.ID

INNER JOIN PAYMENTS P ON O.ID=P.ORDERID

-- Username'i kullanıcı adı olarak adlandırmamıza

-- rağmen bu sorgu hata verir. Çünkü sorguları

-- tablo üstünde yapar. Bu sorunu halletmek için

-- View kullanacağız. View bu alanın bir tablo gibi

-- çalışmasını sağlar. Birazdan onu yapacağım.

WHERE KULLANICIADI='M\_AKCAN'

-- View oluşturma

CREATE VIEW İLKVİEW

AS

SELECT

Blabalbalbalbala

-- Bu sefer de USERNAME\_ ile arama yapamayacağız.

-- Joinleri yazıp, view oluşturunca artık tekrar yazmamıza gerek kalmayacak.

SELECT \* FROM İLKVİEW

WHERE KULLANICIADI='N\_PEKER'

-- View'de kolon isimlerinin nasıl adlandırılacağını update edebileceğimiz için herhangi bir değişiklik durumunda bunu yapmak daha kolay olacaktır.

-- Örneğin excelde ayları sayı ile değil isim ile göstermek istersek view'i update etmemiz yeterli gibi.

-- View üzerinden de filtreli sorgulama yapılabilir.

USER DEFINED FUNCTION

-- Fonksiyon oluşturma

-- Programmability > Functions içerisinde tutulur.

CREATE FUNCTION DBO.TOPLA(@SAYI1 AS INTEGER, @SAYI2 AS INT)

RETURNS INT

AS

BEGIN

DECLARE @SONUC AS INT

SET @SONUC=@SAYI1+@SAYI2

RETURN @SONUC

END

-- Sonucu 50 olarak verir.

SELECT DBO.TOPLA(20,30)

-- Doğum tarihini alıp, yaşı verir.

CREATE FUNCTION DBO.CALCULATEAGE(@BIRTHDATE AS DATE)

RETURNS INT

AS

BEGIN

DECLARE @RESULT AS INT

SET @RESULT=DATEDIFF(YEAR,@BIRTHDATE,GETDATE())

RETURN @RESULT

END

SELECT DBO.CALCULATEAGE('19990727')

-- Fonksiyon ile ilk 1000 kullanıcının yaşını hesapladık.

SELECT TOP 1000 \* ,

DBO.CALCULATEAGE(BIRTHDATE) AS AGE

FROM CUSTOMERS

CREATE FUNCTION DBO.GET\_ITEM\_INFO(@ITEMID INT)

RETURNS TABLE

AS

RETURN

(

SELECT

MIN(UNITPRICE) AS MINPRICE,

MAX(UNITPRICE) AS MAXPRICE,

AVG(UNITPRICE) AS AVGPRICE,

SUM(LINETOTAL) AS TOTALSALE,

SUM(AMOUNT) AS TOTALAMOUNT

FROM ORDERDETAILS WHERE ITEMID=@ITEMID

)

SELECT \* FROM DBO.GET\_ITEM\_INFO(15)

SELECT ITEMS.ID,ITEMS.ITEMCODE,ITEMS.ITEMNAME,

ITEMINFO.MINPRICE AS ENDUSUKFIYAT,

ITEMINFO.MAXPRICE AS ENYUKSEKFIYAT,

ITEMINFO.AVGPRICE AS ORTALAMAFIYAT,

ITEMINFO.TOTALSALE AS TOPLAMSATIS,

ITEMINFO.TOTALAMOUNT AS TOPLAMSATISMIKTARI

FROM ITEMS

CROSS APPLY DBO.GET\_ITEM\_INFO(ITEMS.ID) AS ITEMINFO

**STORED PROCEDURE**

T-SQL cümlelerinin SQL Server’ın hafızasına kaydedilerek derlendiği ve derlenmiş hallerinin çalıştırıldığı yapılardır. Okul ve iş yerlerindeki turnikeler bu mantıkla çalışır.

Database adı>Programmability>Stored Procedures’te tutulurlar.



--ÖRNEK STORED PRODUCE

CREATE PROCEDURE SP\_WORKER\_INOUT

@WORKERBARCODE AS VARCHAR(50),

@DATE AS DATETIME,

@IOTYPE AS VARCHAR(1),

@GATEID AS INTEGER

AS

BEGIN

DECLARE @WORKERNAME AS VARCHAR(100)

DECLARE @WORKERID AS INTEGER

SELECT @WORKERNAME=WORKERNAME, @WORKERID=ID FROM WORKERS

WHERE WORKERBARCODE=@WORKERBARCODE

IF @WORKERID IS NULL

BEGIN

RAISERROR('OKUTULAN KART GEÇERSİZDİR',16,1)

RETURN

END

DECLARE @LASTIO AS VARCHAR(1)

SELECT TOP 1 @LASTIO=IOTYPE FROM WORKERTRANSACTIONS

WHERE @WORKERID=WORKERID AND DATE\_>=CONVERT(DATE,GETDATE())

ORDER BY DATE\_ DESC

IF @LASTIO=@IOTYPE

BEGIN

IF @IOTYPE='G'

BEGIN

RAISERROR('ZATEN İÇERİDESİNİZ',16,1)

RETURN

END

IF @IOTYPE='Ç'

BEGIN

RAISERROR('ZATEN DIŞARIDASINIZ',16,1)

RETURN

END

END

INSERT INTO WORKERTRANSACTIONS(WORKERID,DATE\_,IOTYPE,GATEID)

VALUES (@WORKERID, @DATE,@IOTYPE,@GATEID)

SELECT @WORKERNAME AS WORKERNAME, @DATE AS DATE\_, @IOTYPE AS IOTYPE

END

-- ÇALIŞTIRMAK İÇİN

EXEC SP\_WORKER\_INOUT '0846B9B5-CEB1-488F-97A4-D08AAAD32C02','2020-07-27 09:15:22','G',1

-- VEYA

EXEC SP\_WORKER\_INOUT @WORKERBARCODE='0846B9B5-CEB1-488F-97A4-D08AAAD32C02',@DATE='2020-07-27 09:15:25',@IOTYPE='G',@GATEID=1

-- ŞEKLİNDE KULLANABİLİRİZ.

-- DEĞİŞKEN ADINA ATAMA YAPMAZSAK, SIRALI YAZMAK ZORUNDAYIZ.

-- '%' ile boş parametre gönderebiliriz.

-- Stored Produces'tan gelen verileri tabloya atabiliriz.

INSERT INTO TABLOADI EXEC SP\_ADI PARAMETRELER

-- SP istatistiklerini görme

SELECT

DB\_NAME(database\_id) as DBNAME,

OBJECT\_NAME(object\_id) AS PROCNAME,

\*

FROM sys.dm\_exec\_procedure\_stats

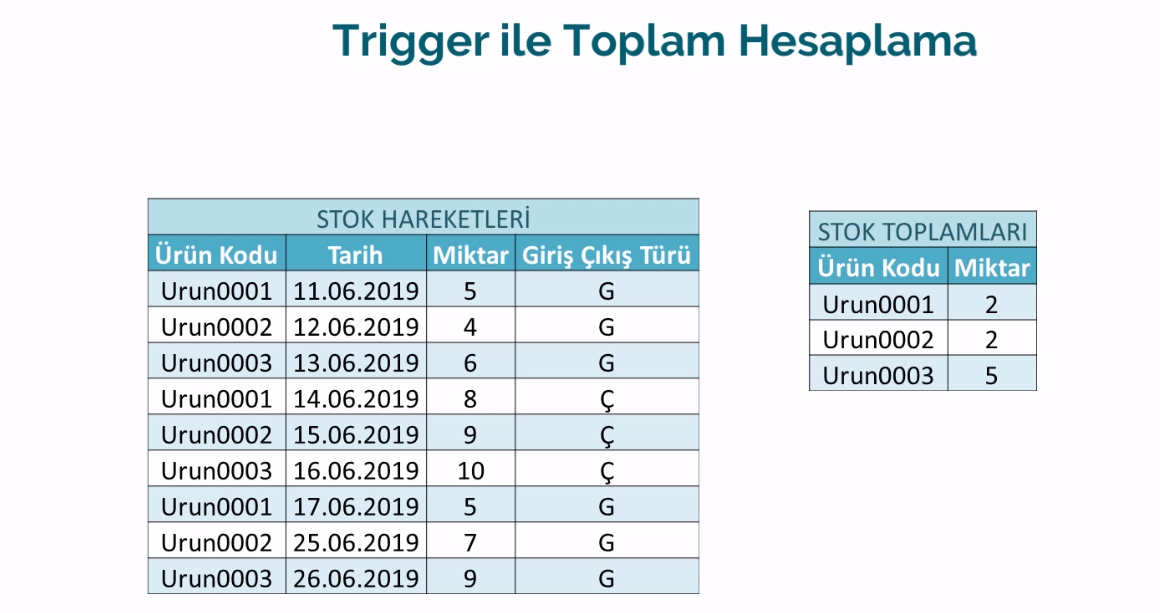
Where DB\_NAME(database\_id)='HR'

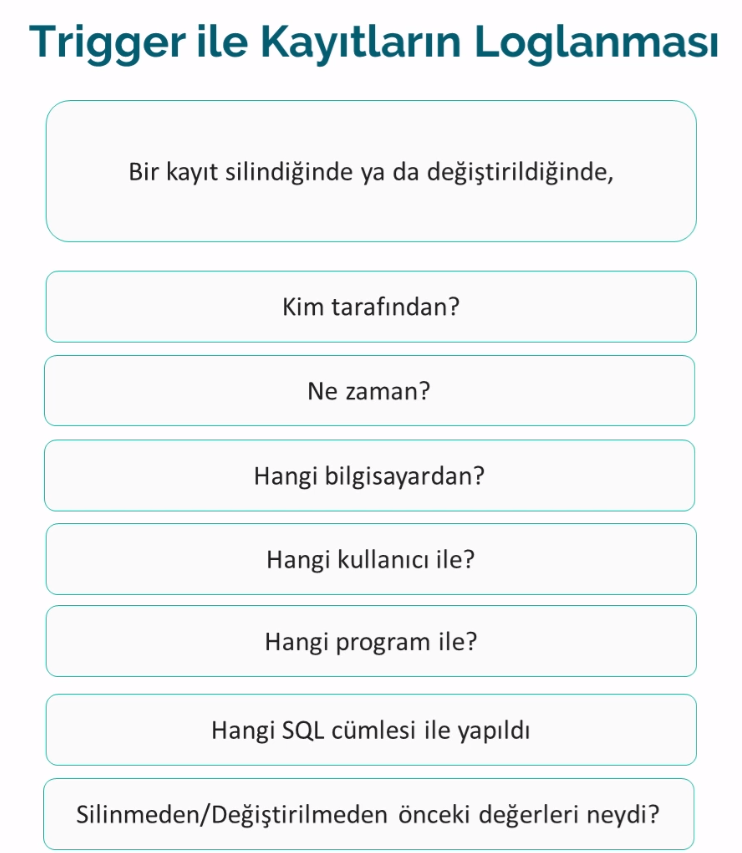
**Stored Procedure Kullanmanın Faydaları**

* Execution plan çıkarıldığı ve derlendiği için ilk 4 aşamayı atlar ve daha hızlı çalışır.
* Client Server mimarisinin aksine toplu işlemler kendi içerisinde çalıştığı için sorgular network hızında değil ram hızında çalışır.
* Yazılım güncellemeden değişiklikler yapılabilir.
* SQL Injection saldırılarına karşı kesin çözümdür.
* Güvenlidir. Kritik raporlar için stored procedure bazında yetki verilebilir.
* Herhangi bir programlama dilinde yazılabilecek hemen hemen her türlü komut burada yazılıp, çalıştırılabilir.
* Stored Procedurelar birbiri içerisinde çağırılabilir.
* Performansı ölçülebilir. Kaç kez çalıştırılmış, en son ne zaman çağırılmış gibi bilgiler görüntülenebilir.

**TRIGGER**

* Veritabanı tablosunda bir işlem gerçekleştiğinde başka bir işlemin otomatik olarak gerçekleşmesidir.
* Burada işlem olarak kast edilen şey data manipülasyonudur.(insert, update, delete)
* Yazılan triggerlar insert, update, delete işlemlerinden sonra otomatik olarak çalışan yapılardır.
* Triggerların içinde sanal olarak inserted ve deleted tabloları oluşur.
* Inserted tablosu yeni eklenen veya update edilen kaydın yeni değerlerinin tutulduğu tablodur.
* Deleted tablosu ise silinen veya değiştirilen kaydın eski değerlerini tutar.
* Triggerlar genelde otomatik toplam hesaplama, son değeri alma ya da loglama amacıyla kullanılır.
* Database>Tables>Tablo>Triggers’da tutulur.





-- Itemtransaction tablosunu doldurmak için script yazdık.

DECLARE @I AS INT=0

WHILE @I<1006880

BEGIN

DECLARE @ITEMID AS INT

DECLARE @DATE AS DATETIME

DECLARE @AMOUNT AS INT

DECLARE @IOTYPE AS SMALLINT

SET @ITEMID=ROUND(RAND()\*527,0)

IF @ITEMID=0

SET @ITEMID=1

SET @DATE= DATEADD(DAY,-ROUND(RAND()\*365,0),GETDATE())

SET @AMOUNT=ROUND(RAND()\*9,0)+1

SET @IOTYPE=ROUND(RAND()\*1,0)+1

INSERT INTO ITEMTRANSACTIONS

(ITEMID,DATE\_,AMOUNT,IOTYPE)

VALUES(@ITEMID, @DATE, @AMOUNT, @IOTYPE)

SET @I=@I+1

END

**Toplam Tablosu Insert Trigger**

-- Stock tablosu tüm itemidler içerisinde bulunan ve stock değerleri 0 olan bir tablo olduğu için insert olayına karışmadan direkt update yaptık

CREATE TRIGGER TRG\_TRANSACTION\_INSERT

ON ITEMTRANSACTIONS

AFTER INSERT

AS

BEGIN

DECLARE @ITEMID AS INT

DECLARE @AMOUNT AS INT

DECLARE @IOTYPE AS SMALLINT

SELECT @ITEMID=ITEMID, @AMOUNT=AMOUNT, @IOTYPE= IOTYPE FROM INSERTED

IF @IOTYPE=1

UPDATE STOCK SET STOCK = STOCK + @AMOUNT WHERE ITEMID=@ITEMID

IF @IOTYPE=2

UPDATE STOCK SET STOCK = STOCK - @AMOUNT WHERE ITEMID=@ITEMID

END

**Toplam Tablosu Delete Trigger**

CREATE TRIGGER TRG\_TRANSACTION\_DELETE

ON ITEMTRANSACTIONS

AFTER DELETE

AS

BEGIN

DECLARE @ITEMID AS INT

DECLARE @AMOUNT AS INT

DECLARE @IOTYPE AS SMALLINT

SELECT @ITEMID=ITEMID, @AMOUNT=AMOUNT, @IOTYPE= IOTYPE FROM DELETED

IF @IOTYPE=1

UPDATE STOCK SET STOCK = STOCK - @AMOUNT WHERE ITEMID=@ITEMID

IF @IOTYPE=2

UPDATE STOCK SET STOCK = STOCK + @AMOUNT WHERE ITEMID=@ITEMID

END

**Toplam Tablosu Update Trigger**

CREATE TRIGGER TRG\_TRANSACTION\_UPDATE

ON ITEMTRANSACTIONS

AFTER UPDATE

AS

BEGIN

DECLARE @ITEMID AS INT

DECLARE @IOTYPE AS SMALLINT

DECLARE @OLDAMOUNT AS INT

DECLARE @NEWAMOUNT AS INT

DECLARE @AMOUNT AS INT

SELECT @ITEMID=ITEMID, @OLDAMOUNT=AMOUNT, @IOTYPE= IOTYPE FROM DELETED

SELECT @ITEMID=ITEMID, @NEWAMOUNT=AMOUNT, @IOTYPE= IOTYPE FROM INSERTED

SELECT @AMOUNT = @OLDAMOUNT - @NEWAMOUNT

IF @IOTYPE=1

UPDATE STOCK SET STOCK = STOCK - @AMOUNT WHERE ITEMID=@ITEMID

IF @IOTYPE=2

UPDATE STOCK SET STOCK = STOCK + @AMOUNT WHERE ITEMID=@ITEMID

END

**Triggerla Son Kaydı Loglama**

-- Tablodan tabloya veri atma

INSERT INTO WORKERS

(WORKERCODE, WORKERNAME, GENDER, BIRTHDATE, TCNO, WORKERBARCODE)

SELECT TOP 1000

TCNO, NAMESURNAME, GENDER, BIRTHDATE, TCNO,NEWID()

FROM CRM2.DBO.CUSTOMERS

SELECT \* FROM WORKERS

**------------------------------------------------------------------**

CREATE TRIGGER TRG\_TRANSACTION\_INSERT

ON WORKERTRANSACTIONS

AFTER INSERT

AS

BEGIN

DECLARE @WORKERID AS INT

DECLARE @DATE AS DATETIME

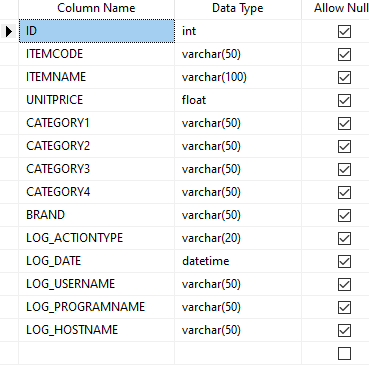
DECLARE @IOTYPE AS VARCHAR(1)

SELECT @WORKERID= WORKERID, @DATE= DATE\_, @IOTYPE=IOTYPE FROM INSERTED

UPDATE WORKER\_LAST\_TRANSACTION SET LASTIOTYPE=@IOTYPE, LASTDATE=@DATE WHERE WORKERID=@WORKERID

END

**Triggerla Loglama**



CREATE TRIGGER TRG\_ITEMS\_UPDATE

ON ITEMS

AFTER UPDATE

AS

BEGIN

INSERT INTO ITEMS\_LOG ([ID], [ITEMCODE], [ITEMNAME], [UNITPRICE], [CATEGORY1], [CATEGORY2], [CATEGORY3], [CATEGORY4], [BRAND], [LOG\_ACTIONTYPE], [LOG\_DATE], [LOG\_USERNAME], [LOG\_PROGRAMNAME], [LOG\_HOSTNAME])

SELECT [ID], [ITEMCODE], [ITEMNAME], [UNITPRICE], [CATEGORY1], [CATEGORY2], [CATEGORY3], [CATEGORY4], [BRAND], 'UPDATE', GETDATE(), SUSER\_NAME(), PROGRAM\_NAME(),HOST\_NAME()

FROM DELETED

END

**Instead of Trigger**

Instead of Delete veya Update dediğimizde silme veya update işlemini yapmaz. Bizim istediğimiz işleri yapar.

**TRANSACTION**

SELECT C.CUSTOMERNAME, A.ACCOUNTNO, A.ACCOUNTNAME, A.CURRENCY, B.BALANCE FROM ACCOUNTBALANCE B

INNER JOIN ACCOUNTS A ON A.ID=B.ACCOUNTID

INNER JOIN CUSTOMERS C ON C.ID=A.CUSTOMERID

BEGIN TRAN

INSERT INTO MONEYTRANSACTIONS

(ACCOUNTID, TRANTYPE, AMOUNT, DATE\_, CURRENCY, EFTCODE1, EFTCODE2)

VALUES

(1,2,1000,'2020-04-14 14:21:36', 'TL', '12345678','')

IF @@ERROR>0

BEGIN

ROLLBACK TRAN

RETURN

END

UPDATE ACCOUNTBALANCE SET BALANCE= BALANCE-1000 WHERE ACCOUNTID=1

IF @@ERROR>0

BEGIN

ROLLBACK TRAN

RETURN

END

INSERT INTO MONEYTRANSACTIONS

(ACCOUNTID, TRANTYPE, AMOUNT, DATE\_, CURRENCY, EFTCODE1, EFTCODE2)

VALUES

(2,1,1000,'2020-04-14 14:21:36', 'TL', '12345678','0987654321')

IF @@ERROR>0

BEGIN

ROLLBACK TRAN

RETURN

END

UPDATE ACCOUNTBALANCE SET BALANCE= BALANCE+1000 WHERE ACCOUNTID=2

IF @@ERROR>0

BEGIN

ROLLBACK TRAN

RETURN

END

COMMIT TRAN

-- Triggerlar transactionların bir parçasıdır. Geri alınan işlemler onları da etkiler.

--Transactionlar oluşurken değişen verileri kilitler. Bu verileri çekmek için With(NOLOCK) kullanmalıyız.

-- With(NOLOCK) ile yapılan sorgularda transaction devam etse bile veri görüntülenir.

-- SQL Server transaction'ın tamamlanma ihtimalini daha yüksek gördüğünden tamamlanınca göstereceği değeri verir.

SELECT \* FROM ITEMS

WITH (NOLOCK)

WHERE ID = 3

-- Sistemde açık transaction var mı yok mu gösterir.

DBCC OPENTRAN

**BACKUP / RESTORE İŞLEMLERİ**

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, kuş içeren bir resim

Açıklama otomatik olarak oluşturuldu

* DatabaseAdı>Sağtık>Task>Backup kısmından backup alınabilir.
* Backup options’tan sıkıştırılmış olarak backup alabiliriz.
* Backupları sıkıştırılmış olarak almayı SQL Server ayarlarından da ayarlayabiliriz.
* Select \* INTO YENITABLOADI FROM ESKİTABLOADI komutuyla eski tabloyu yeni tabloya kopyalayabiliriz.
* Database>Properties>Options>Recovery Model’dan Full’u seçersek yapılan her hareket transaction log’da tutulur, Bulk-logged seçersek sadece yapılan toplu işlemler tutulur, simple’ı seçersek işlemlerin log’u tutulmaz.
* Transaction log backup’ı alındığında log dosyası temizlenir. Yani alınan dosyalar birbirinin parçasıdır.
* Differantial backup’da ise bu durum farklıdır. Her differantial backup kendinden önceki full backup’ın parçasıdır.
* DatabaseAdı>Sağtık>Task>Restore kısmından transaction log ile geri dönüş yapılabilir.
* SQL SERVER>Databases>Sağtık>Attach ile database eklenebilir.
* DatabaseAdı>Sağtık>Detach ile database kaldırılabilir. Database silinmez.
* DatabaseAdı>Sağtık>Take Offline ile database offline hale getirilebilir.

**SQL SERVER AGENT**

SQL Server’da otomatik ve periyodik olarak çalışmasını istediğimiz işlemler için kullandığımız yapıdır. Örnek olarak; Yedek Alma, Index Bakım Planları, Database Shrink Planları, Veri Ambarı Doldurma, Otomatik Rapor Oluşturma, Mail Gönderme verilebilir.

Tüm konfigürasyonunu MSDB sistem database’inde tutar.

SQLServerAgent>Jobs>Sağtık>NewJobs diyerek yeni iş eklenebilir.

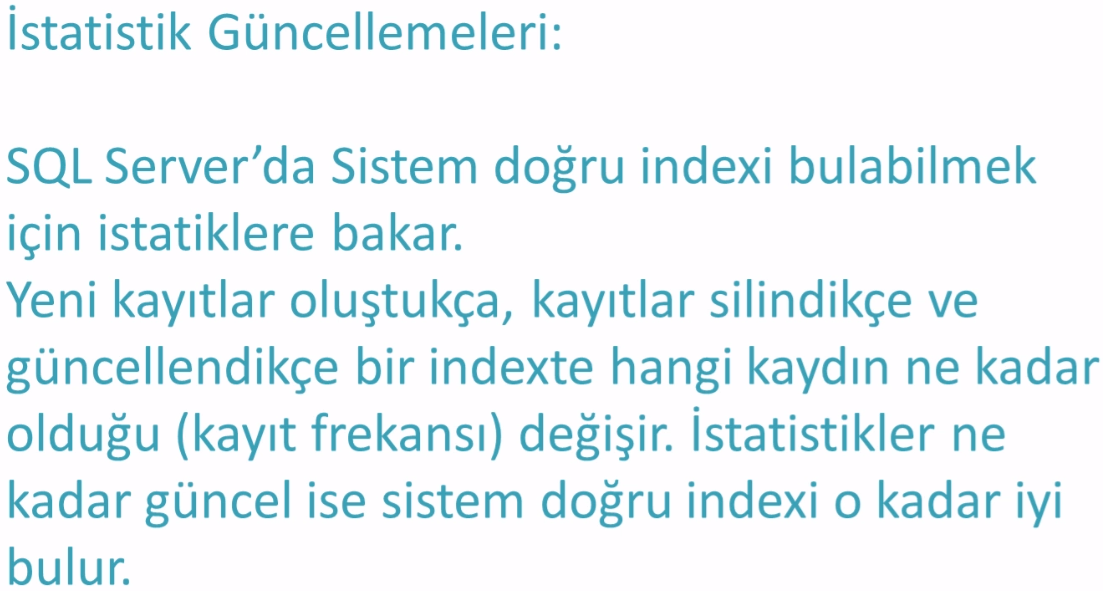
Joblara sağ tık ile enable-disable atanabilir.

SQLServer>Managament>DatabaseMail>Sağtık>ConfigureDatabaseMail ile yeni mail eklenebilir. Mail türüne göre ayarlara bakılmalı.

<https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/sp-send-dbmail-transact-sql?view=sql-server-ver15> sayfasından örneklere bakılabilir.

DATABASE BAKIM PLANLARI

kuş içeren bir resim

Açıklama otomatik olarak oluşturuldukuş içeren bir resim

Açıklama otomatik olarak oluşturuldukuş içeren bir resim

Açıklama otomatik olarak oluşturuldutablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

-- SQLServerAdı>Management>MaintenancePlans altında yer alırlar. Sağ tıklayıp MaintenancePlanWizard veya NewMaintenancePlan diyerek yeni bir bakım planı ekleyebiliriz.

-- Buradan bakım planlarını, eski bakım dosyalarını silme gibi işlemleri yapabiliriz.

-- TSQL kodlarıyla mail atarak veya SQL Server Agent’a operatör ekleyerek notify operator tasklarla işlemin başarı durumu hakkında istenen kişileri bilgilendirebiliriz.

-- DatabaseAdı>Sağtık>Tasks>Shrink’ten veya maintenance ekleyerek database’in boyutunu şişiren gereksiz boşluklar silinebilir. Otomatik olarak yapılması çok doğru değil. Sıkıntı çıkarabilir.

BIG DATA

-- Tools>SQL Server Profiler’dan thraceler oluşturup, filtreleyerek sorguların denetimini yapabiliriz. Örneğin bir sorgunun ne kadar sürdüğüne vs. bakabilir ve buna göre hızlandırma çalışması yapabiliriz.

-- Sistemin bu logları bir dosyaya veya bir tabloya kaydetmesini de sağlayabiliriz.

-- Büyük datalarda indexsiz çalışmak imkansız.

CURSOR

-- Diğer programlardaki for döngüsüne benzer.

DECLARE @NAMESURNAME AS VARCHAR(50)=''

DECLARE @AGE AS INT

DECLARE @EMAIL AS VARCHAR(100)=''

DECLARE @MSG AS VARCHAR(500)

DECLARE CRS CURSOR FOR

SELECT NAMESURNAME,DATEDIFF(YEAR,BIRTHDATE,GETDATE()) AS AGE,EMAIL

FROM USERS

WHERE DAY(BIRTHDATE)=25 AND MONTH(BIRTHDATE)=4

OPEN CRS

FETCH NEXT FROM CRS INTO @NAMESURNAME,@AGE,@EMAIL

WHILE @@FETCH\_STATUS=0

BEGIN

SET @MSG ='Sayýn '+@NAMESURNAME+', Doðum gününüz kutlu olsun. '+CONVERT(VARCHAR,@AGE)+'. Yaşınızı, nice mutlu yıllar dileriz.'

EXEC msdb.dbo.sp\_send\_dbmail @profile\_name='SQLMAIL',

@SUBJECT='MUTLU YILLAR',

@BODY=@MSG,

@recipients=@EMAIL

FETCH NEXT FROM CRS INTO @NAMESURNAME,@AGE,@EMAIL

END

CLOSE CRS

DEALLOCATE CRS