STORED PROCEDURE

Birden fazla işlemi paketlenmiş bir halde bir tek komutla çalıştırmamız gerektiğinde stored procedüre’ler kullanılır.

İşlemlerden kastedilen tüm transact-sql deyimleridir.

Bir prosedür oluşturulduktan sonra veri tabanı sunucusunda saklanır ve defalarca çalıştırılabilir.

Ağ bazlı uygulamalar yapılabilir. Veritabanı sunucusunda saklandığı daha az ağ kaynaklarını kullanır.

Genel ifade

CREATE : oluşturmak

ALTER : yapıyı değiştirmek

DROP : nesneyi silmek

CREATE PROC

Yada

CREATE PROCEDURE

CREATE PROC prosedürAdı

VarsaParametreler

AS

…..

….

Sql deyimleri

….

…

Go

Örnek:

Hesaplar arası para transferi yapan bir prosedür oluşturunuz. Ancak bakiye değeri sıfırın altına düşerse “bakiye az” mesajı versin.

İhtiyaç duyulan parametreler

Gönderen hesap no

Alıcı hesap no

Gönderilen para miktarı

Prosedürün adı : sp\_para\_transfer

CREATE PROC sp\_para\_transfer

@gonderen varchar(10),

@alici varchar(10),

@para Money

AS

…

SQL DEYİMLERİ

….

Şimdi prosedür içindeki sql deyimlerini yazalım.

-- Bakiye sıfırın altına iner mi?

IF (Gönderen bakiye)>=@para

Parayı gönder

ELSE

Bakiye düşük mesajı ver.

IF (select bakiye from hesaplar where hesapno=@gonderen)>=@para

begin

-- para gönder

Update hesaplar set bakiye=bakiye-@para where hesapno=@gonderen

Update hesaplar set bakiye=bakiye+ @para where hesapno=@alici

End

ELSE

Print ‘Bakiye yetersiz.’

Go

Yukarıdaki kodları birleştirelim.

CREATE PROC sp\_para\_transfer

@gonderen varchar(10),

@alici varchar(10),

@para Money

AS

IF (select bakiye from hesaplar where hesapno=@gonderen)>=@para

begin

-- para gönder

Update hesaplar set bakiye=bakiye-@para where hesapno=@gonderen

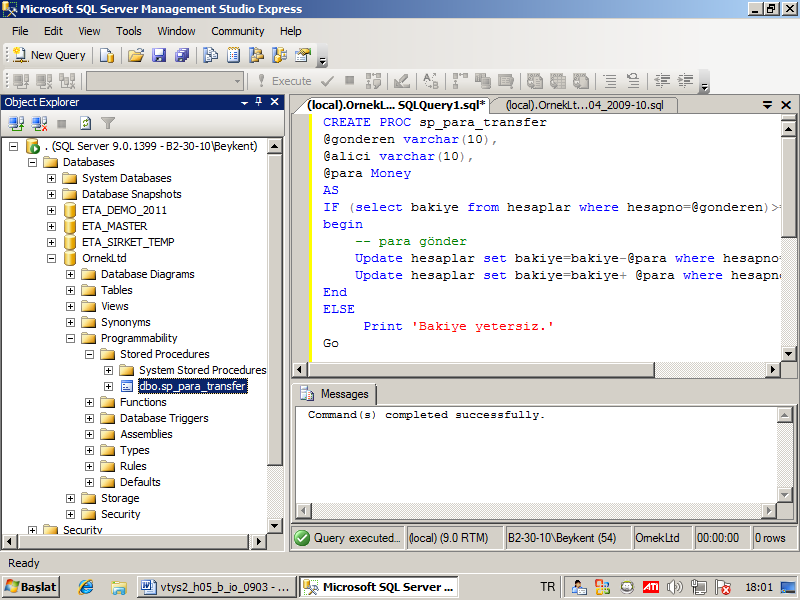
Update hesaplar set bakiye=bakiye+ @para where hesapno=@alici

End

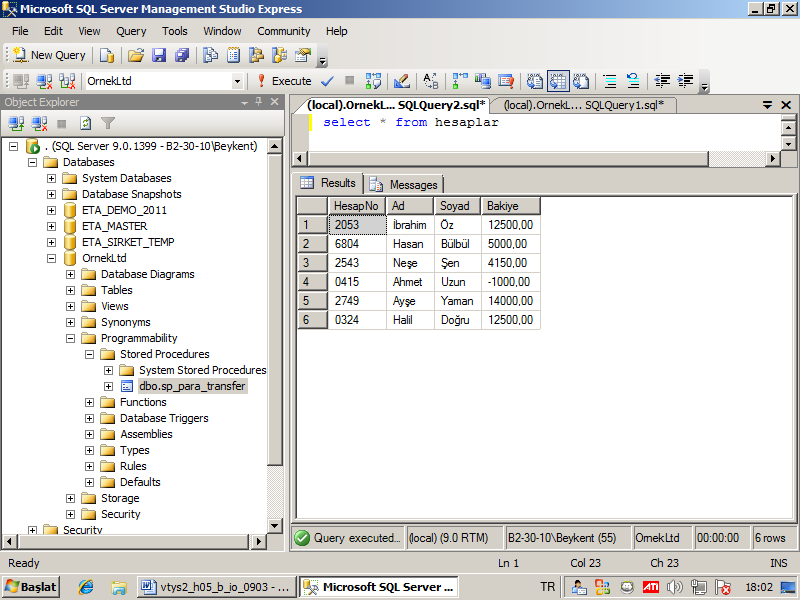
ELSE

Print ‘Bakiye yetersiz.’

Go

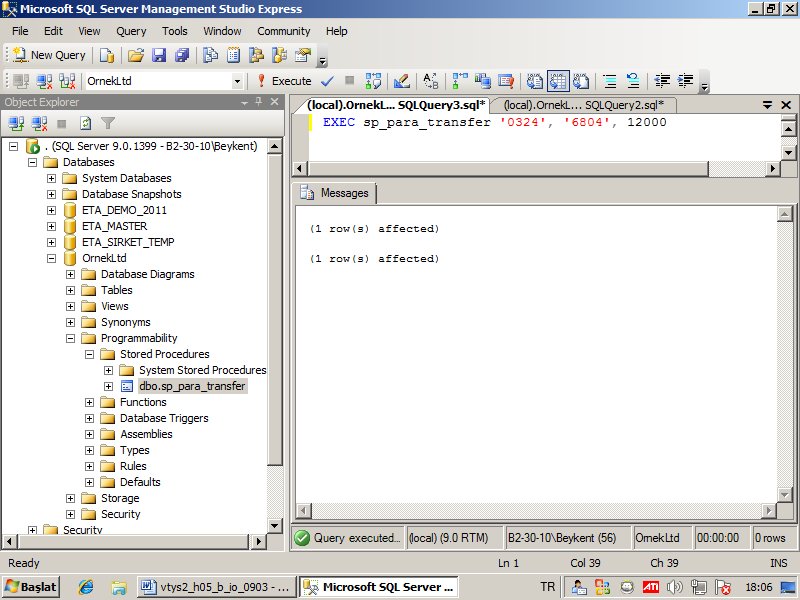


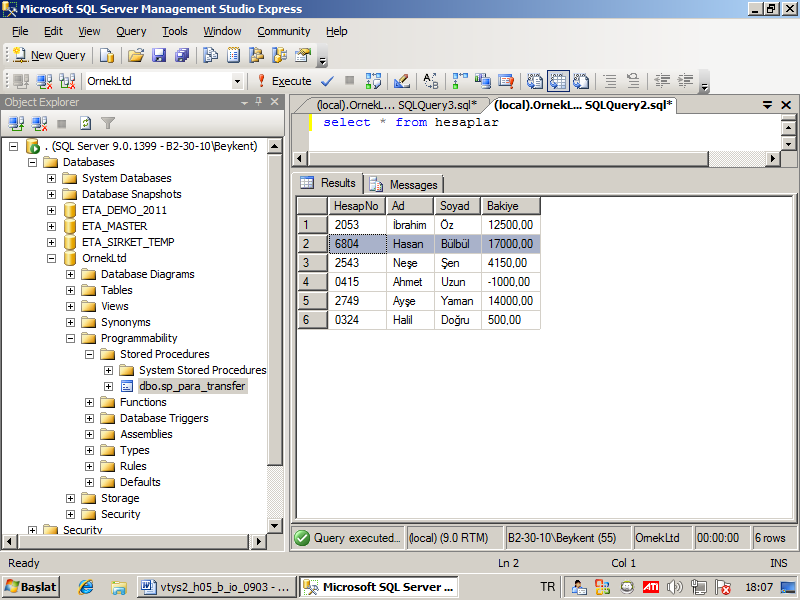
Yukarıdaki prosedürü çalıştıralım.



Yukarıdaki prosedürü çalıştırarak, 0324 nolu hesaptan, 6804 nolu hesaba 12000 değerini gönderiniz.

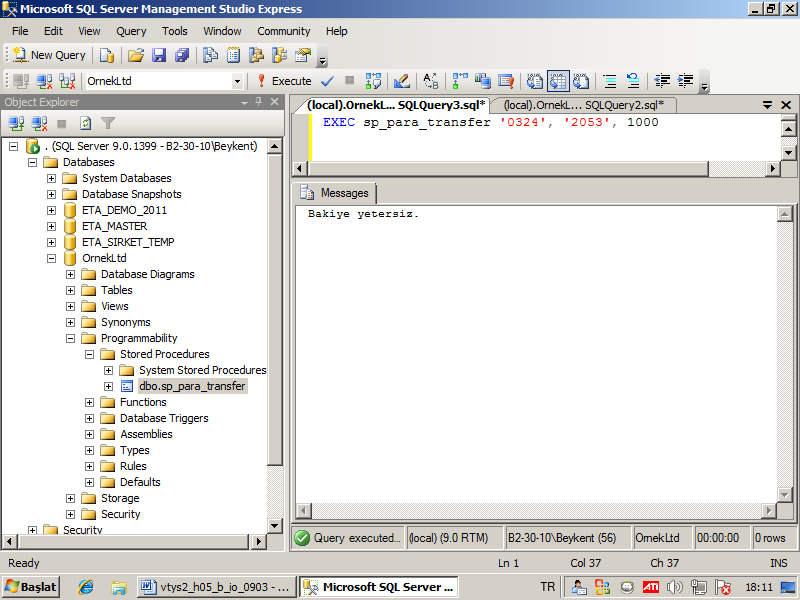
EXEC sp\_para\_transfer ‘0324’, ‘6804’, 12000



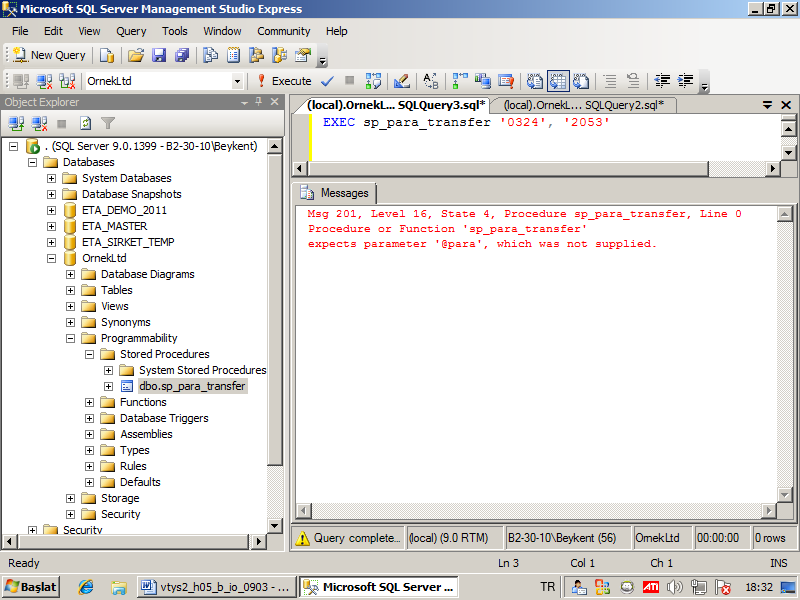


Yukarıdaki prosedürü çalıştırarak 0324 nolu hesaptan, 2053 nolu hesaba 1000 değerini gönderiniz.

EXEC sp\_para\_transfer ‘0324’, ‘2053’, 1000



Yukarıdaki prosedürü eksik parametre ile çalıştıralım.



Parametre eksik olduğunda ve varsayılan da yoksa hata mesajı verir.

Prosedürdeki parametrelere varsayılan değer atanabilir.

@parametre veritipi = değer

Yukarıdaki prosedürdeki parametreler NULL değerini varsayılan olarak atayalım.

ALTER PROC sp\_para\_transfer

@gonderen varchar(10) =NULL,

@alici varchar(10) =NULL,

@para Money =NULL

AS

IF (select bakiye from hesaplar where hesapno=@gonderen)>=@para

begin

-- para gönder

Update hesaplar set bakiye=bakiye-@para where hesapno=@gonderen

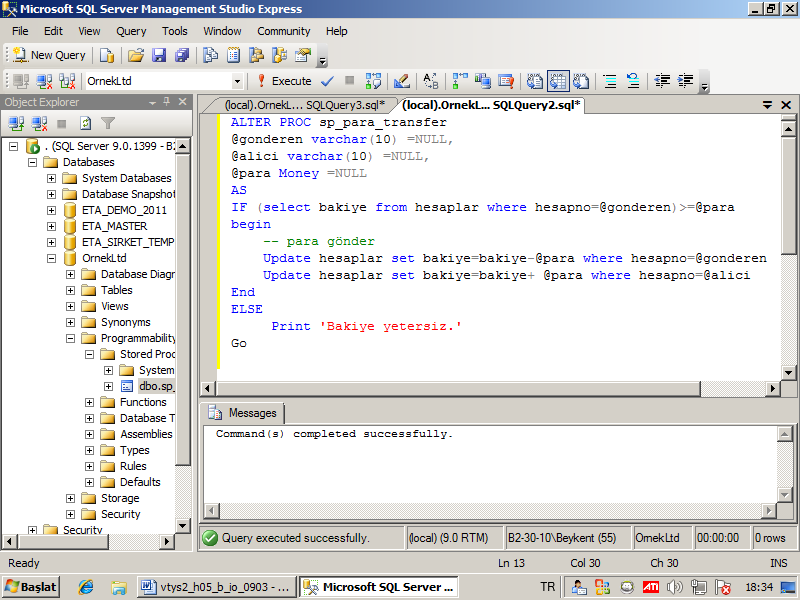
Update hesaplar set bakiye=bakiye+ @para where hesapno=@alici

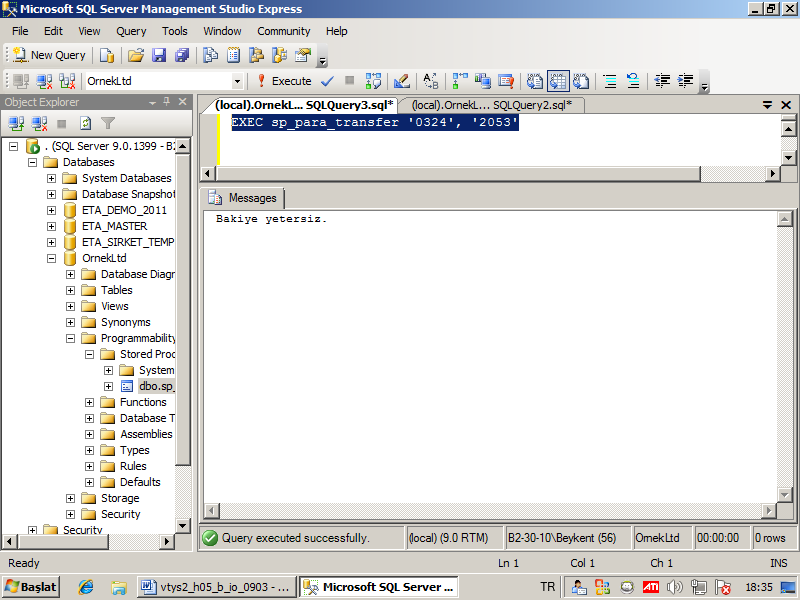
End

ELSE

Print ‘Bakiye yetersiz.’

Go



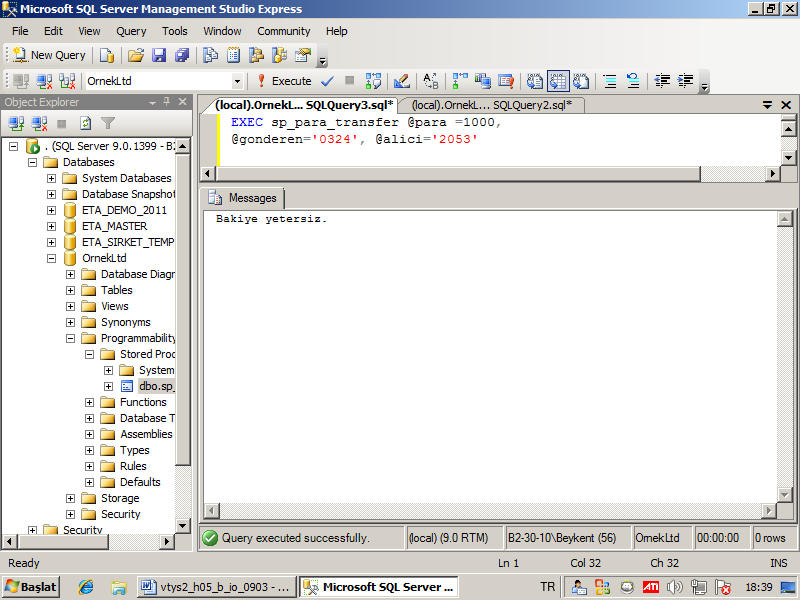


EXEC sp\_para\_transfer @gonderen=‘0324’, @alici=‘2053’,

@para =1000

Parametre adı yazılarak prosedür çalıştırılabilir. Parametre adı yazıldığında yer değişikliği olabilir.

EXEC sp\_para\_transfer @para =1000, @gonderen=‘0324’, @alici=‘2053’



Örnek:

Verilen bir hesap numarasının bakiyesini artırıp azaltan bir prosedür yazınız. Bu prosedürde parametreler: Hesapno, para miktarı ve işlem kodudur. (işlem kodu = 1 ise azalsın, 0 ise artsın)

CREATE PROC sp\_bakiye\_artir\_azalt

@hesap\_no varchar(10),

@para Money

@islem\_kodu tinyint=1

AS

…

Sql komutları

…

Go

Sql komutları yazalım

IF @islem\_kodu = 1

--bakiye azalt

ELSE

--bakiye artır

-- bakiye azalt

Update hesaplar set bakiye=bakiye-@para where hesapno=@hesap\_no

-- bakiye artır

Update hesaplar set bakiye=bakiye+@para where hesapno=@hesap\_no

Yukarıdaki kodları birleştirelim

CREATE PROC sp\_bakiye\_artir\_azalt

@hesap\_no varchar(10),

@para Money,

@islem\_kodu tinyint=1

AS

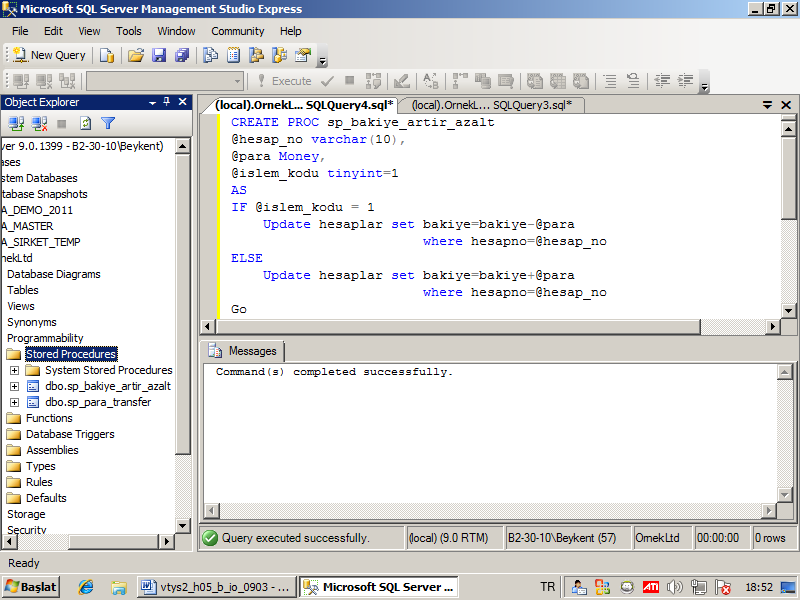
IF @islem\_kodu = 1

Update hesaplar set bakiye=bakiye-@para where hesapno=@hesap\_no

ELSE

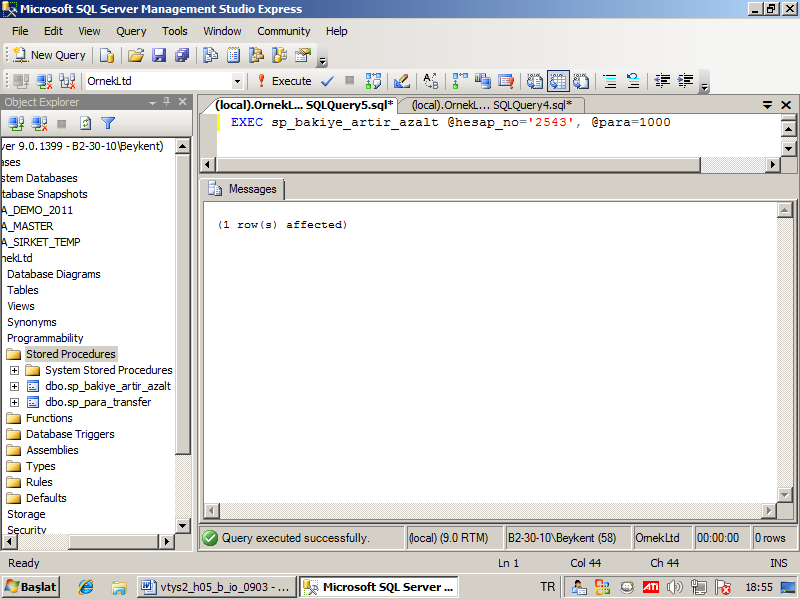
Update hesaplar set bakiye=bakiye+@para where hesapno=@hesap\_no

Go



Yukarıdaki prosedürü kullanarak, 2543 nolu hesabın bakiyesini 1000 azaltınız.

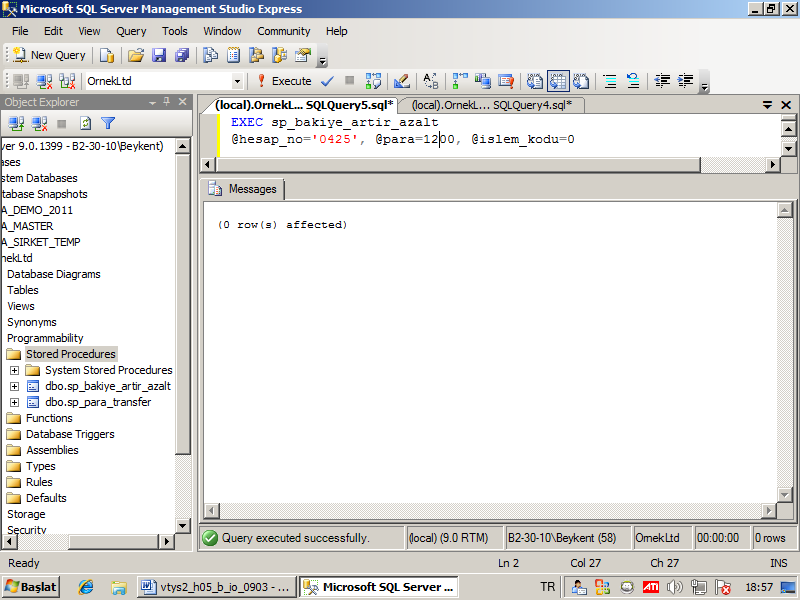
EXEC sp\_bakiye\_artir\_azalt @hesap\_no=’2543’, @para=1000



Yukarıdaki prosedürü kullanarak, 0425 nolu hesabın bakiyesini 1200 artırınız.

EXEC sp\_bakiye\_artir\_azalt

@hesap\_no=’0425’, @para=1200, @islem\_kodu=0



2749 nolu hesabı 4000 azaltalım.

EXEC sp\_bakiye\_artir\_azalt

@hesap\_no=’2749’, @para=4000, @islem\_kodu=1



EXISTS ve NOT EXISTS fonksiyonları

Bu fonksiyon IF içinde kullanılır. Sonuç true/false’tur.

EXISTS (sorgu) : Eğer sorgudan bir yada daha fazla kayıt dönerse değeri true, hiç kayıt dönmez ise değeri false’dir.

NOT EXISTS(sorgu) : exists fonksiyonunun olumsuzudur.

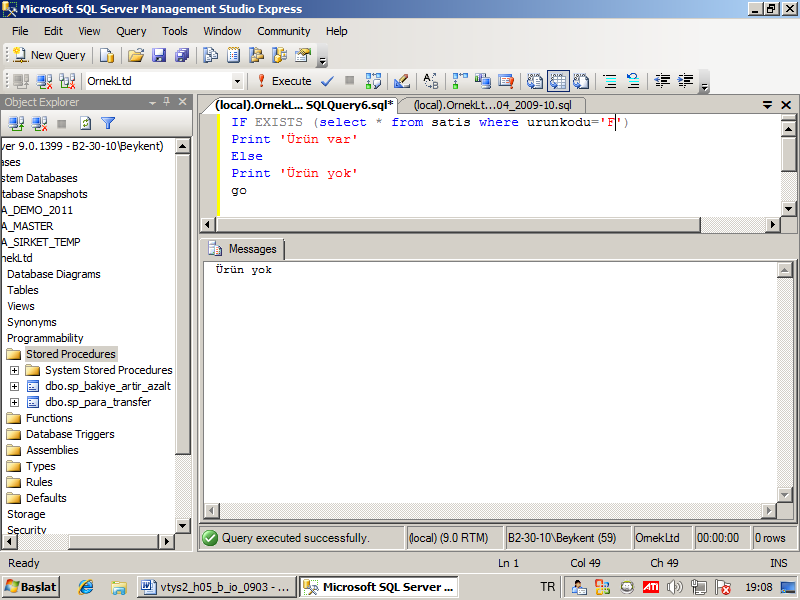
IF EXISTS (select \* from satis where urunkodu=’F’)

Print ‘Ürün var’

Else

Print ‘Ürün yok’

go



TRIGGER

Bir çeşit prosedürdür. Ancak bu prosedür bir olaya bağlı olarak kendiliğinden çalışır. Çalışması için tabloya veri eklemek, veri değiştirmek veya veri silmek yeterlidir.

Trigger’lar tablo üzerinde oluşturulur.

Herhangi bir tablo üzerinde yapılan insert, update veya delete işlemi sonrasında otomatik olarak, aynı tablo veya başka bir tabloda veri eklenebilir, güncellenebilir veya silinebilir.

Triggerlarla girilen veriler kontrol edilebilir. Uygun veri değilse reddedilebilir.

Trigger çeşitleri:

* After trigger : insert, update veya delete işlemi gerçekleştikten sonra çalışan trigger’dır. Çoğunlukla bu tip trigger kullanılır.
* instead of trigger : insert, update veya delete işlemi gerçekleşmeden önce çalışan trigger’dır.

Örnek:

Satış tablosuna yeni veri girildiğinde çalışacak “veri eklendi” mesajı veren bir trigger yazınız ve satış tablosuna veri girerek bu trigger’ın çalışmasını sağlayınız.

CREATE TRIGGER trg\_satis\_gir

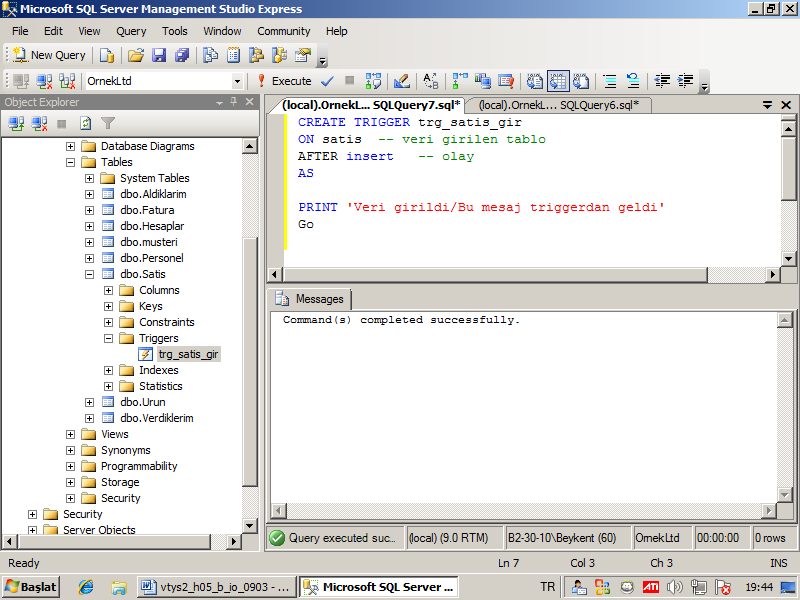
ON satis -- veri girilen tablo

AFTER insert -- olay

AS

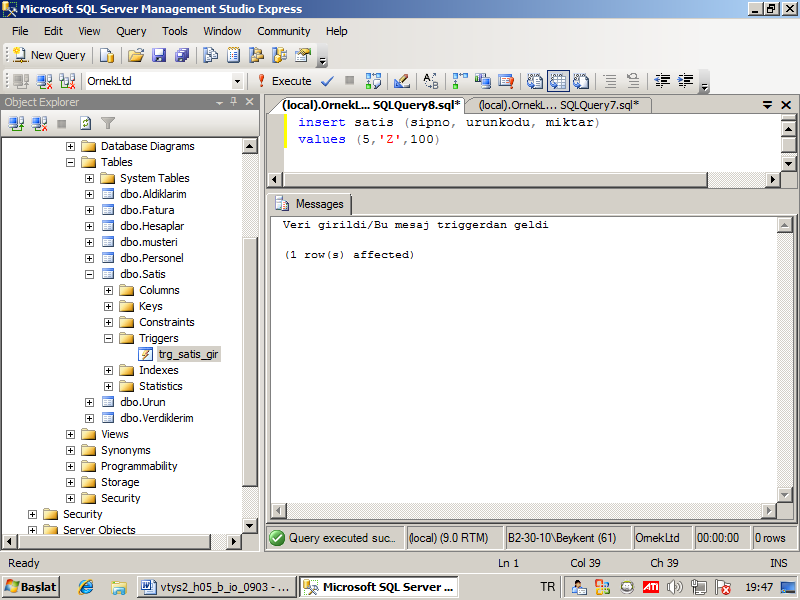
PRINT ‘Veri girildi/Bu mesaj triggerdan geldi’

Go



Satış tablosuna veri girelim

insert satis (sipno, urunkodu, miktar) values (5,’Z’,100)



SON EKLENEN, DEĞİŞEN veya SİLİNEN bilginin içeriğine erişmek

Trigger’da bu verilere erişmek için iki sanal tablo kullanılır.

Bu iki tablonun adı (bu tabloya sadece trigger erişebilir.)

* inserted tablosu
* deleted tablosu

inserted tablosu içinde

* yeni eklenen kayıtlar bulunur (insert deyimi ile)
* update deyimi çalıştığında değiştikten sonraki veri bulunur.

Deleted tablosu içinde

* silinen kayıtlar bulunur
* update deyimi çalıştığında değişmeden önceki veri bulunur.

CREATE TRIGGER trg\_satis

ON satis

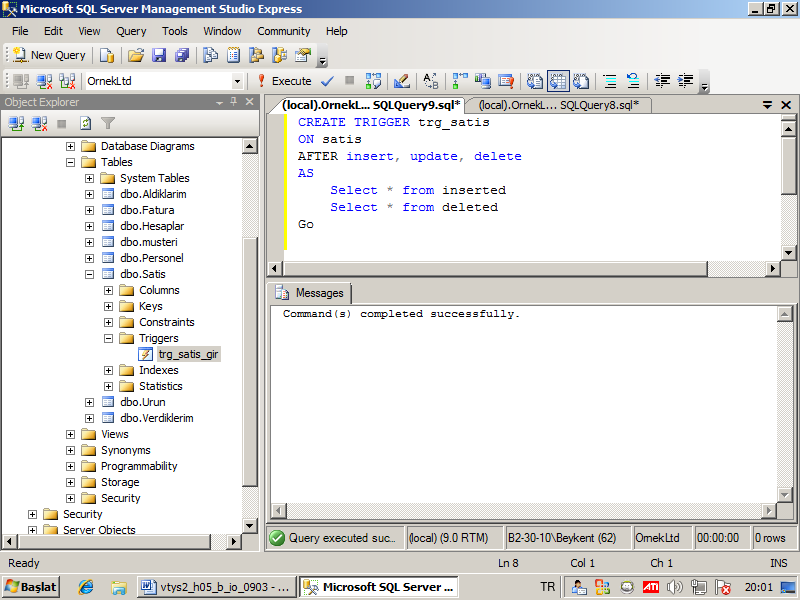
AFTER insert, update, delete

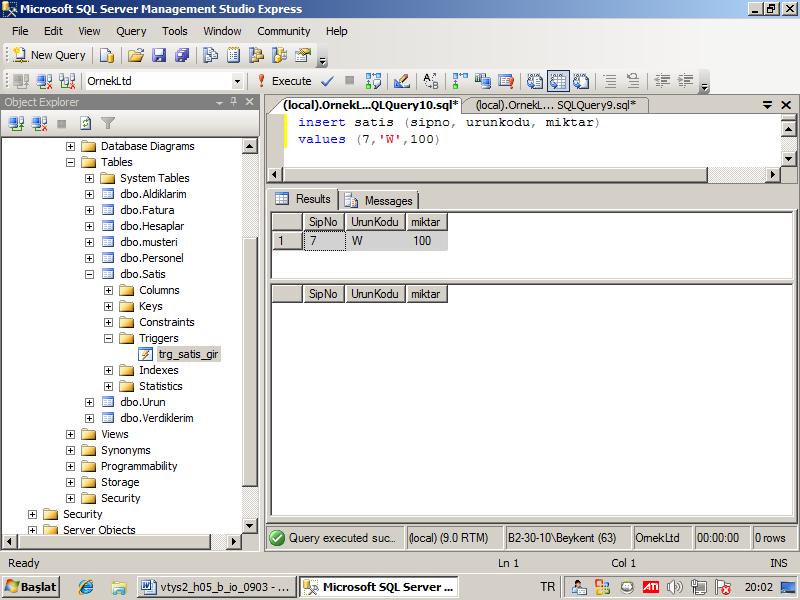
AS

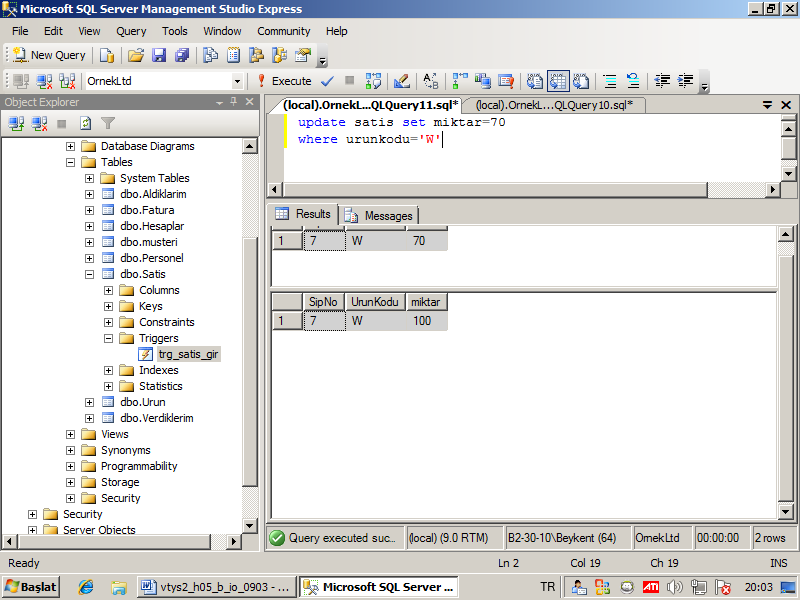
Select \* from inserted

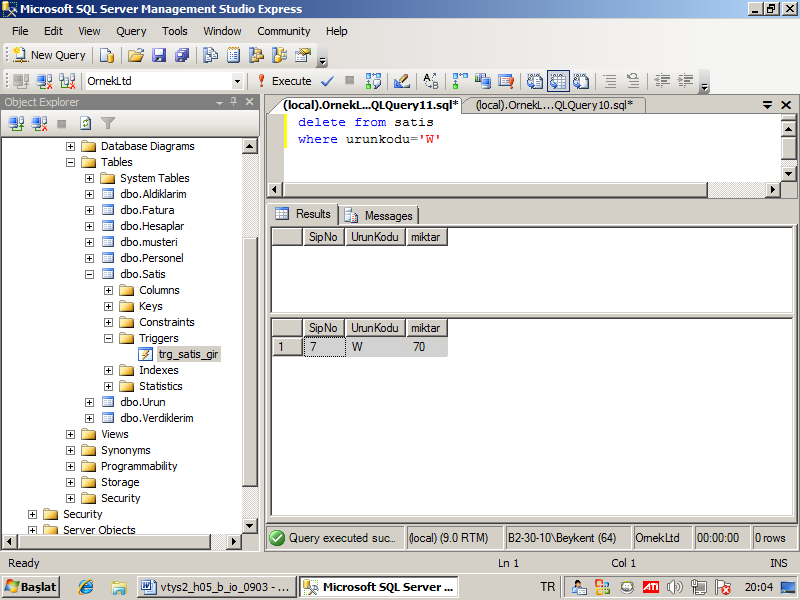
Select \* from deleted

Go









İnserted ve deleted tablosunun kolon isimleri nereden gelir?

Cevabı: bu tabloların yapısı ile üzerinde bulunduğu tablonun yapısı aynıdır.

Biz satış üzerinde trigger’ı oluşturmuştuk dolayısıyla inserted ve deleted tablolarının yapısı, satış tablosu ile aynıdır.