T.C.
ONDOKUZ MAYIS ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Veri Tabanı Laboratuvarı Dersi

Deney Föyü-7

T.C
ONDOKUZMAYIS ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Veri Tabanı Lab. Dersi Deney Föyü-7

TRANSACTION KAVRAMI

Transaction, özet olarak daha küçük parçalara ayrılamayan işlem demektir. Özellike bir grup işlemin arka arkaya gerçekleşiyor olmasına rağmen, seri işlemler halinde ele alınması gerktiğinde kullanılır. Transaction bloğu içerisindeki işlemlerin tamamı gerçekleşinceye kadar hepsi gerçekleşmemiş varsayılır.
Bir banka uygulamasını düşünün. Bir kullanıcı başka bir kullanıcıya havale yaptığında ne olacağına bakalım. Öncelikle havale yapanın hesap bilgilerinden havale yaptığı miktar düşülür. Ardından alıcının hesabına bu miktar eklenir ve havale gerçekleşmiş olur. Ancak her zaman şartlar istendiği gibi olmayabilir. Örneğin, gönderenin hesabından para düşüldüğü anda elektrik kesilebilir ya da program takılabilir. Bu durumda, ne olur? Örneğin, gönderenin hesabından para düşülmüştür ama alıcının hesabına da geçmemiştir yani bir kısım paranın sahibinin kimliği kaybedilmiş olur. Bu da sistemin olası durumlar dışında veri kaybetmeye müsait bir hal alması demektir. Bu durumun bir şekilde önlenmesi gerekir.

Daha küçük parçalara ayrılamayan en küçük işlem yığınına Transaction denir. Geçerli kabul edilmesi bir dize işlemlerin tamamının yolunda gitmesine bağlı durumlarda transaction kullanılır. Transaction bloğu ya hep ya hiç mantığı ile çalışır. Ya tüm işlemler düzgün olarak gerçekleşir ve geçerli kabul edilir veya bir kısım işlemler yolunda gitse bile, blok sona ermeden bir işlem bile yolunda gitmese hiçbir işlem olmamış kabul edilir.

SQL Server 3 farklı transaction desteği sağlar:

1. Harici(Explicit) Transaction: SQL Server'in kullanıcı tarafından bir BEGIN TRAN ifadesi ile transaction'a başlatılması şeklindeki bloktur. Bir aksilik olması halinde SQL Server tarafından veya kullanıcı tarafından COMMIT ifadesi ile gerçekleşmiş olarak veya ROLLBACK ifadesi ile hiç olmamış olarak sonlandırlabilir.

2.Dahili(Implict) Transaction: SQL Server'in belli ifadelerden sonra otomatik olarak transaction açmasını sağlar. Bu modda, bu belli ifadeler kullanıldıktan sonra, kullanıcı tarafından transaction'ın sonlandırılması gerekir. Bu nedenle zahmetli bir mod'dur.

3.Auto Commit : Hiç bir transaction mod'u tayin edilmedi ise, SQL Server bu modda çalışır. Auto Commit modunda iken, her bir batch(yığın, Query Analyzer için iki go arasındaki ifade veya bir defada çalıştırılan bütün SQL ifadeleri) bir transaction bloğu olarak ele alınır. Batch içerisinde bir sorun olursa, SQL Server otomatik olarak bütün batch'i geri alır(ROLLBACK eder).

Ancak biz genel olarak transaction denilince, harici transaction işlemlerini kastederiz. Harici

transaction bloğunun başlatılması ve gelişimini ele alacak olursak:

1.Transaction bloğu başlatılır. Böylece yapılan işlemlerin geçersiz sayılabileceği VTYS'ye deklare edilmiş olur ve SQL Server Auto Commit modundan çıkıp, Explicit moda geçer.

2.Transaction bloğu arasında yapılan her bir işlem bittiği anda başarılı olup olmadığına gerek varsa, programcı tarafından bakılıp, başarılı olmadığı anda geri alım işlemine geçilebilir(ROLLBACK). Ancak bir sorun olması halinde, SQL Server tarafından da verilerin tutarlılığını denetlemek üzere, transaction bloğunun başladığı andan itibaren bir güç kesilmesi gibi durum ortaya çıkarsa, değişiklikler dikkate alınmayacak şekildedir. Bu, transaction logları denilen yöntem ile yapılır. Bu yöntemde, bir transaction başladıktan sonra, verileri tutan sayfalar diskten(HDD) hafızaya(RAM) yüklenir ve ilgili değişiklikler, önce hafızada yapılır. Ardından, değişikliklerin izdüşümü loglar diske yazdırılır, ardından veriler de güncellenir.

3.Tüm işlemler tamamlandığı anda COMMIT ile bilgiler yeni hali ile sabitlenir. Başarısız bir sonuç ise ROLLBACK ile en başa alınır ve bilgiler ilk hali ile sabitlenir.

Bu örnek için aşağıdaki tabloyu kullanacağız: Hesap tablosu:

```
CREATE TABLE(
 HesapNo CHAR(20) NOT NULL PRIMARY KEY,
 Adi VARCHAR(55),
 Soyadi VARCHAR(55),
 Sube INTEGER,
 Bakiye FLOAT
)
```

Genel Yapısı şu şekildedir:

1. Transaction başlatılır:

Örnek:

```
DECLARE @havaleMiktar FLOAT DECLARE @aliciHesap VARCHAR(20), @gonderenHesap VARCHAR(20)
SET @aliciHesap='1111122132113'
SET @gonderenHesap='1111122132112'
SET @havaleMiktar=20000000
-- 20 milyon havale edilecek
BEGIN TRANSACTION
UPDATE tblHesap
```

```
SET bakiye=bakiye - 20000000
WHERE hesapNo=@gonderenHesap
UPDATE tblHesap
SET bakiye=bakiye + 20000000
WHERE hesapNo=@aliciHesap
```

2. İşlem başarılı olursa, COMMIT ile transaction bitirilir. Başarısız olduğunun anlaşılması haline ROLLBACK komutu ile transaction başarısız olarak bitirilebilir.(Yani en baştaki duruma geri dönülür)

```
COMMIT
```

Sabitleme Noktaları:

Bazen, bir noktaya kadar gelindikten sonra, işlemlerin buraya kadar olanını geçerli kabul etmek isteriz ama bundan sonraki işlemler için de transaction(geri alabilme seçeneği)ne ihtiyaç duyarız. Bu türden durumlarda sabitleme noktalarından faydalanılır.
Bir sabitleme noktası başlatıldığı anda, en başa dönme seçeneği saklı kalmak üzere, noktanın oluşturulduğu yere de dönme seçeneği sunar.

Genel yapısı şu şekildedir:

```
SAVE TRANSACTION sabitleme_notkasi_adi
```

Örnek:

```
BEGIN TRANSACTION UPDATE tblHesap
SET bakiye = 5000000
WHERE hesapNo='1'
SAVE TRANSACTION svp_kaydet
DELETE FROM tblHesap
WHERE  HesapNo='1';
ROLLBACK TRAN svp_kaydet;
SELECT * FROM tblHesap;
ROLLBACK TRAN ;
SELECT * FROM tblHesap;
COMMIT
```

İPUCU: Bir uygulamanın parçası olarak görev yapan transaction'lar, gerçek uygulamalarda genellikle stored procedure'ler içerisinde başlatılırlar. Bu durumda, bir hesaptan başka bir hesaba havale işlemini gerçekleştirecek bir stored procedure , dışarıdan, üç parametre alır:

havale eden hesap numarası, havaleyi alacak hesap numarası ve havale miktarı. RETURN parametresi ile de işlem başarılı ise 1, değil ise 0 döndürülebilir.

```
CREATE PROC SP$havale(@aliciHesap CHAR(10),@gonderenHesap CHAR(10),@miktar MONEY) AS
  BEGIN TRANSACTION
    UPDATE tblHesap
    SET bakiye=bakiye - 20000000
    WHERE hesapNo=@gonderenHesap
    IF @@ERROR<>0
      ROLLBACK
      RETURN 0
    UPDATE tblHesap
    SET bakiye=bakiye + 20000000
    WHERE hesapNo=@aliciHesap
    IF @@ERROR<>0
      ROLLBACK
      RETURN 0
  COMMIT
  RETURN 1
GO
```

Test Etmek İçin;

```
DECLARE @sonuc TINYINT
EXEC @sonuc =  SP$havale('1111122132113', '1111122132112'  ,20000000)
```

TETİKLEYİCİLER (TRİGGERS)

Tetikleyiciler sadece tablolarda veya görünümlerde(views) Insert, Update ve Delete komutları çalıştırılırken başka işlerin yapılması için tanımlanan sql cümleleridir. Mantık olarak saklı yordamlara benzerler. Fakat bunlar ilişkili oldukları tablo veya gönünümde ilgili işlem gerçekleştirilirken otomatik olarak çalışmalarıdır. T-SQL iki farklı tetikleyici vardır; bunlar AFTER ve INSTEAD OF tetikleyicileridir. AFTER tetikleyiciler ilgili işlemleri gerçekleştirildikten hemen sonra yapılırlar. INSTEAD OF tetikleyicileri ise işlem yapılırken araya girip öncesinde veya sonrasında başka işlemleri yapabilme yeteneğine sahiptir.

Trigger lar;

- Tablo,
- Database,
- Server

bazında olabilir.

I - DML (Data Manuplation Language)

Tablo üzerinde işlem yapıldığında

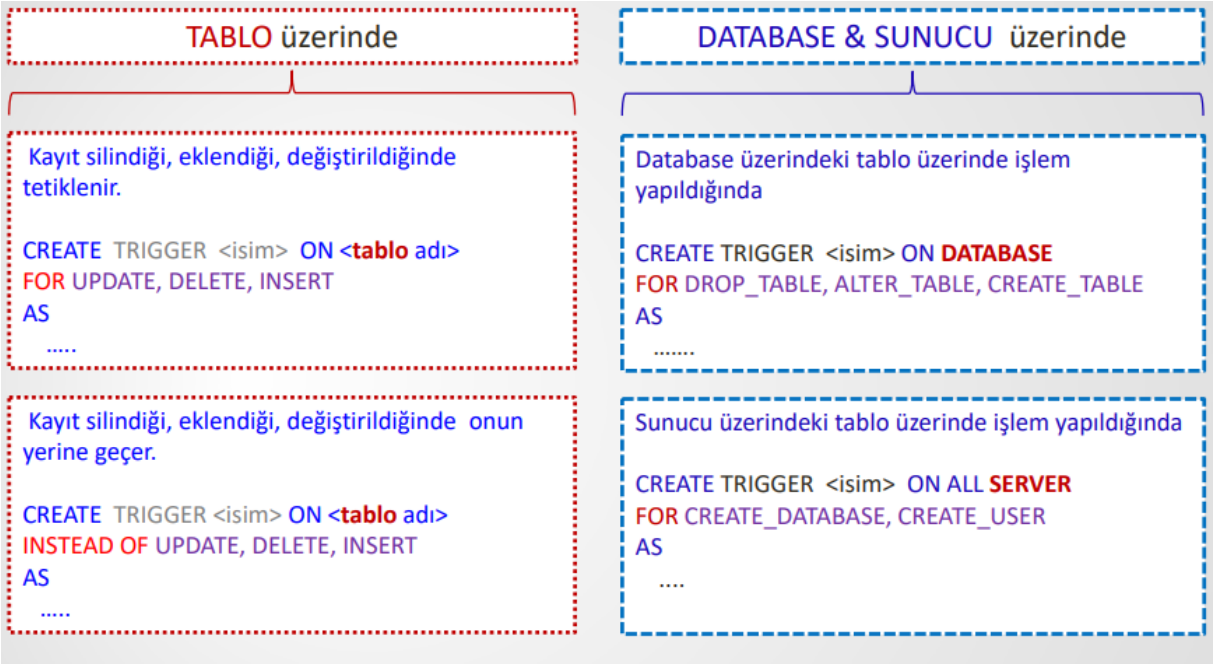
- Tabloya Kayıt EKLENDİĞİNDE (INSERT)
- Tabloya Kayıt SİLİNDİĞİNDE (DELETE)
- Tabloya Kayıt GÜNCELLENDİĞİNDE (UPDATE)

II - DDL (Data Definition Language)

Sunucu üzerinde işlem yapıldığında

- TABLO Oluşturulduğunda (CREATE_TABLE)
- TABLO Silindiğinde (DROP_TABLE)
- TABLO Oluşturulduğunda (ALTER_TABLE)
- DATABASE Oluşturulduğunda (CREATE_DATABASE)
- DATABASE Silindiğinde (DROP_DATABASE)
- DATABASE Oluşturulduğunda (ALTER_DATABASE)

Benzer şekilde; CREATE_INDEX, CREATE_USER, …



NOT: Trigger başarısız olursa onu tetikleyen komut da başarısız olur.

Örnek: SuTahakkuk tablosuna veri kaydedilirken Tahsilat toplam tablosuna ilgili HemsehriId ve döneme ait toplam tutarın değerini güncellemek olsun. Bir insert tetikleyicisi örneği olarak düşünebilirsiniz. Gördüğünüz gibi parametre alırlar ve bu parametredeki değerlere göre işlem yaparlar. Saklı yordamlarda olduğu gibi geri değer döndürmezler.

```
CREATE  TRIGGER  ti_SuTahakkuk ON SuTahakkuk FOR INSERT AS
DECLARE @pDONEM char(1)
DECLARE @pTUTAR decimal(18,2)
DECLARE @pHEMSEHRI decimal(18,2)

SELECT @pDONEM=Donem, @pTUTAR=ToplamTutar,@pHEMSEHRI = HemSehriId    FROM INSERTED

BEGIN
 IF (@pDONEM = '1') BEGIN
   UPDATE TahsilatToplam SET Donem1 = Donem1 + @pTUTAR WHERE HemSehriId = @pHEMSEHRI
 END
 ELSE IF (@pDONEM = '2') BEGIN
   UPDATE TahsilatToplam SET Donem2 = Donem2 + @pTUTAR WHERE HemSehriId = @pHEMSEHRI
 END
 ELSE IF (@pDONEM = '3') BEGIN
   UPDATE TahsilatToplam SET Donem3 = Donem3 + @pTUTAR WHERE HemSehriId = @pHEMSEHRI
 END
 ELSE IF (@pDONEM = '4') BEGIN
   UPDATE TahsilatToplam SET Donem4 = Donem4 + @pTUTAR WHERE HemSehriId = @pHEMSEHRI
 END
 ELSE IF (@pDONEM = '5') BEGIN
   UPDATE TahsilatToplam SET Donem5 = Donem5 + @pTUTAR WHERE HemSehriId = @pHEMSEHRI
 END
 ELSE IF (@pDONEM = '6') BEGIN
   UPDATE TahsilatToplam SET Donem6 = Donem6 + @pTUTAR WHERE HemSehriId = @pHEMSEHRI
 END

END
```

Kaynak: SQL Server Books Online

YAPILMASI İSTENENLER

1- **tOgrenci: ogrenciID (Primary Key), ad, soyad, dogumtarih, adres, telefon, bolumID, vizenotu, finalnotu** bilgilerini içermektedir. Oluşturduğunuz öğrenci tablosu üzerine bir silme tetikleyicisi yazınız. Bu tablodan bir kayıt silindiğinde togrenciYedek isimli ve aynı özelliklere sahip başka bir tabloya silinen kaydın aynısının eklenmesini sağlayınız.

2- **Personel(Id,Ad,Maas)** ve **ZamListesi(Id,Ad,EskiMaas,YeniMaas,Tarih)** adında iki tablomuz olsun. Bir personelin maaşına zam yapıldığında ZamListesi

tablosuna ilgili personel bilgilerini, eski maaşını ve yeni maaşını ekleyin. (trigger kullanın)

3- **Oyuncu(Id,Ad,Takım,GüncellemeTarihi)** adında bir tablomuz olsun. Bu tablodaki herhangi bir kayıt güncellendiğinde GüncellemeTarihi otomatik olarak anlık tarihe update olsun. Bu işlemi gerçekleştiren bir trigger yazınız.

4- **Kitap(Id,Ad,Yazar)** ve **KitapArsiv(Id,Ad,Yazar,ArsivTarihi)** adında iki tablomuz olsun. Kitap tablosundan bir kayıt silindiğide otomatik olarak arşiv tablosuna bir kayıt eklensin. Bu işlemi gerçekleştiren bir trigger yazınız.

5- **siparis(id,ürünId,miktar,tarih** ve **ürün(id,ad,fiyat,stok)** adında iki tablo yaratalım. sipariş tablosuna bir insert yapıldığında, ilgili ürünün stok miktarını da güncelleyin.