ONDOKUZMAYIS ÜNİVERSİTESİ
Bilgisayar Mühendisliği Bölümü

Veri Tabanı Lab. Dersi Deney Föyü-3

Konu: Saklı Yordamlar (Stored Procedures), Fonksiyonlar (Functions), İmleçler (Cursorler), Transaction Blokları

Deney Sonucu İstenen: Öncelikle aşağıdaki anlatılanları dikkatlice okuyunuz**.**

tOgrenci: ogrenciID (Primary Key), ad, soyad, dogumtarih, adres, telefon

bilgileriniz içermektedir.

- Yukarıdaki tabloya kendisine gelen parametreleri kullanarak kayıt ekleme işini yapan bir saklı yordam yazınız.
- Aynı tablodan toplam öğrenci sayısını (count,) döndüren saklı yordam oluşturunuz.
- Oluşturmuş olduğunuz saklı yordam nesnelerine herhangi bir programlama dilini kullanarak (c#, python, java vs..) bağlantı yapınız ve çalıştırınız.
- İşlem sonuçlarını kullandığınız programlama ortamında gösteriniz. Ekleme işlemi yapan saklı yordam için bir uyarı mesajı, hesaplamalı işlem yapan saklı yordam için geri dönen değerin gösterilmesini sağlayınız.
- Oluşturduğunuz öğrenci tablosu üzerine bir ekleme tetikleyicisi yazınız. Bu tabloya bir kayıt eklendiğinde togrenciYedek isimli ve aynı özelliklere sahip başka bir tabloya buraya eklenen özelliklerin aynısının eklenmesini sağlayınız.

Saklı Yordamlar: Bir tabloya bağlı olmaksızın veritabanı uzayında tanımlanan belirli bir işi yapmaya yönelik kodlardır. Başka bir değişle "Derlenmiş sql cümlecikleridir". Saklı yordam olarak tanımlanan bu yapılar kullanılan veritabanı yönetim sistemine ait programlanabilir sql dil kullanılarak gerçekleştirilir. Her veritabanı yönetim sistemi standart sql dilini desteklemekle birlikte kendi üzerinde bazı programlanabilir özelliklere sahip bir dil barındırırlar. MS SQL Server ortamında T-SQL dili, ORACLE ortamında PL-SQL PostgreSQL PL/pgSQL dili, Sybase (T-SQL)

Stored Procedure ne işe yarar?

Çalıştırmak istediğimiz sql cümleciklerini bir Saklı Yordam içine yerleştirerek, bunun bir veritabanı nesnesi haline gelmesini ve çalışıtırıldığında doğrudan, veritabanı yöneticisini üzerinde barındıran sunucu makinede işlemesini sağlarız.

Stored Procedure faydaları nelerdir?

İstemci makinelerdeki iş yükünü azaltır ve performansı arttırır (yazıldığı zaman aynı zamanda compile edildikleri için query optimizer tarafından optimize edilmiş en hızlı şekilde çalışır). Sql cümleleri, Saklı Yordam' lardan çok daha yavaş sonuç döndürür. Çok katlı mimariyi uygulamak isteğimiz projelerde faydalıdır. Networkü (Ağ Trafiğini) azaltır. Açık Sql cümleciklerine nazaran daha güvenlidir Programlama deyimlerini içerebilirler. if, next, set vs.. programlama dillerindekine benzer özellikler sunar. Gelen parametrelere göre sorgu yapılıp sonucun dönmesi sağlanabilir.

Aşağıda MS SQL Server ortamında yukarıda anlatılan yapıların kullanımını görebilirsiniz. T-SQL dilini kullanmak için bu dilin bazı özelliklerinden bahsetmek gerekir.

DECLARE : Adından da anlaşıldığı gibi bildirmek. T-sql de kullanılacak değişkenler ,cursor gibi özel yapılar vb. bildirmek ve tanıtmak için kullanılır. Tanımlama amaçlı kullanılır.

```
DECLARE @Degisken1 varchar(50)
DECLARE @Degisken2 datetime
```

SET : Değişkenlere değer atama için kullanılır.

```
DECLARE @Degisken1 varchar(20)
SET @Degisken1 = 'www.csharpNedir.com'
```

CURSOR : Çeşitli sql ifadeleri sonucunda elde edilen veri kümesi üzerinde ileri geri ilerlemek için tanımlanan yapılardır.

Fonksiyonlar (Functions) : Belli bir sonucu geri döndürmek için yapılmış bir veya birden fazla yerde kullanılan yapılardır. Programcılıkta kullanmış olduğumuz fonksiyonlar ile aynı mantıktadırlar.

```
CREATE FUNCTION FonksiyonAdı  (Parametreler AS VeriTipi )
    RETURNS GeriDönecekDeğerin VeriTipi
AS
   BEGIN
       Fonksiyonda yapılacak işer
       RETURN GeriDönenDeğer
  END
```

Örnek : Bu örneğimiz SQL cümlelerinde genellikle sorun çıkaran NULL değerleri. Eğer sql cümlesinde iki değer toplarken değerlerden biri NULL ise sonuç NULL olur. Bu fonksiyonumuzda değişkenin değerinin NULL olup olmamadığının kontrolunu yapar.

```
CREATE FUNCTION checkNULL (@pDeger decimal(20,2))
 RETURNS decimal(20,2)
 AS
 BEGIN
  DECLARE @Sayi decimal(20,2);
  SET @Sayi = @pDeger;
  if (@Sayi is NULL)
        SET @Sayi = 0;
  return @Sayi;
END
```

Örnek : Örneğimiz TahsilatToplam tablosunda bir abonenin dönem dönem harcamış olduğu su miktarının bedeleri toplamlarını bulunmakta. Fonksiyonumuz abonenin toplam tahsil edilmiş borcunu göstermekte. RETURNS ifadesi fonksiyonun geri dönüş değerini , DECLARE ifadesi de kullanılan değişkeni bildirir. Ayrıca burda ilk yaptığımız fonksiyonu yani checkNULL fonksiyonunu da kullandık. Böylece Donem değerlerinin NULL olup olmadığının kontrolunu de yapmış olduk.

.

```
CREATE  FUNCTION SuToplam(@pId numeric(18,0))
RETURNS decimal(18,2)
AS
BEGIN
   DECLARE @toplam decimal(18,2)

  SELECT  @Toplam = dbo.checkNull(Donem1) + dbo.checkNull(Donem2)
        + dbo.checkNull(Donem3) + dbo.checkNull(Donem4)
        + dbo.checkNull(Donem5) + dbo.checkNull(Donem6)
      FROM TahsilatToplam WHERE HemSehriId = @pId

   RETURN @Toplam
END
```

Saklı Yordamlar(Stored Procedurs) : Belli bir işlemi gerçekleştirmek için oluşturulan sql ifadeleri topluluğu . Örneğin projenizde sql ifadelerin olmasını istemiyorsanız ve bütün işlemleri veritabanında yapmak istiyorsanız saklı yordamları kullanabilirisinz. Belli parametreler gönderilerek kayıt ekleme, güncelleme, silme veya veriyi listeleme amaçlı saklı yordamlar oluşturabilirisiniz. Belli girdi ve çıktı parametreleri olduğu için kullanılan projenin güvenliliğini artırırlar. Ayrıca istemciden sunucuya uzun sql cümlelerinin gitmesindense saklı yordamın adı ve gerekli parametrelerinin gitmesi ağ trafiğini de azaltmış olur.

```
CREATE PROCEDURE (Parametreler  VeriTipi , DönüşDeğeriParametreler  VeriTipi OUTPUT)
AS
      Fonksiyonda yapılacak işler
```

OUTPUT parametreleri procedur çalıştıktan sonra geri dönüş değeri olan parametrelerdir. Ve parametre tanımlanırken @ karakterinde ikitane yazılarak oluşturulur. @@Param gibi.

```
CREATE PROCEDURE sp_InsSuTahakkuk(
@pHemSehriId int, @pSayacaNo char(10), @pDonem char(1), @pIlkEndex int,
@pSonEndek int, @pHarcamalar decimal(18,2), @pToplamTutar decimal(18,2))
AS
 INSERT INTO SuTahakkuk (HemSehriId, SayacaNo, Donem, IlkEndex,
             SonEndek, Harcamalar, ToplamTutar)
   VALUES (@pHemSehriId, @pSayacaNo, @pDonem ,@pIlkEndex,
             @pSonEndek ,@pHarcamalar, @pToplamTutar)
```

Saklı yordamlar MS SQL ortamında EXECUTE veya EXEC komutları ile çalıştırılırlar.

```
EXECUTE sp_InsSuTahakkuk (1, '1522', '1', 145, 156, 120, 150)
EXEC sp_EmlakBilgileri 2004
```

Örnek : Aşağıdaki örneğimiz de işlemler sonucunda dönüş değeri olarak tablo kullanan bir saklı sordam olacak. Ayrıca bu saklı yordamımızda Cursor oluşturma ve kullanımına ilişkin güzel bir örnek. Bu örneğimiz farklı tablolarda bulunan bilgileri birleştirip tek bir tablo oluşturan bir procedur oldu. EmlakBina tablosunda sadece ilgili tabloların Id'leri olan SahipSicil, Mahalle, Sokak bilgilerinin birleştirilip görsel açıdan daha kullanıştı bir tablo halenine getirilmesi.

```
CREATE   PROCEDURE sp_EmlakBilgileri  @pIslemYili int
AS
DECLARE @AdSoyad varchar(100), @Adres varchar(100)
DECLARE @Ada varchar(100),     @Pafta char(10)
DECLARE @Parsel char(10),      @RaicBedel decimal(18,0)

CREATE TABLE #TTABLES (/* Temp Tablonun oluşturulması*/
AD_SOYAD   VARCHAR(100), ADRES VARCHAR(100),ADA  CHAR(10),
PAFTA CHAR(10), PARSEL CHAR(10), RAIC_BEDEL DECIMAL(18,0),EMLAK_TIP CHAR(4))

/*İlgili veri kümesini tutmak için cursorun tanımlanması*/
DECLARE cEmlakBina CURSOR FOR  SELECT
        (HS.Adi + ' '  + HS.Soyadi) as AdSoyad,
        (M.Adi + ' ' + C.Adi + ' ' + DisKapi + ' ' + IcKapi ) as Adres,
        Ada, Pafta, Parsel, RaicBedel  FROM EmlakBina
        INNER JOIN HemsehriSicil HS ON HS.HemsehriId = SahipSicil
        INNER JOIN Mahalle M ON M.MahalleId = Mahalle_Id
        INNER JOIN Cadde C ON C.SokakId = CaddeId
        WHERE IslemYili =  @pIslemYili
OPEN cEmlakBina /*Cursorun açılması*/

FETCH NEXT FROM cEmlakBina INTO @AdSoyad, @Adres, @Ada, @Pafta, @Parsel ,@RaicBedel

WHILE (@@FETCH_STATUS = 0)
BEGIN
   INSERT INTO #TTABLES (AD_SOYAD,ADRES,ADA,PAFTA,PARSEL,RAIC_BEDEL) /*tabloya ekleme*/
        VALUES ( @AdSoyad, @Adres,@Ada, @Pafta, @Parsel ,@RaicBedel)

FETCH NEXT FROM cEmlakBina  INTO @AdSoyad, @Adres,@Ada, @Pafta, @Parsel ,@RaicBedel
END

CLOSE cEmlakBina /* Cursore kapatılır*/
DEALLOCATE  cEmlakBina /*Cursor yok edilmesi*/

SELECT * FROM #TTABLES /*geriye tablo döndüreceğimiz tablo*/
DROP TABLE #TTABLES /*oluşturulan tablonun yok edilmesi*/
```

Burada önünde # işareti ile oluşturulan TTABLES isimli tablo geçici tablodur. Geçici tablolar T-SQL ortamında oluşturulur , kullanılır işi bittiğinde DROP edilirler.

Tetikleyiciler (Triggers) : 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.

```
CREATE TRIGGER TetikleyiciAdı  ON TabloAdi veyaViewAdi (FOR AFTER , INSTEAD OF) veya (FOR {INSERT, UPDATE, DELETE})
 BEGIN
    Tetikleyicide yapılacak işler
 END
```

Örnek : Örneğimiz 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

TRANSACTIN 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:

```
BEGIN TRAN[SACTION] [transaction_adi]
```

İle bir 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
```

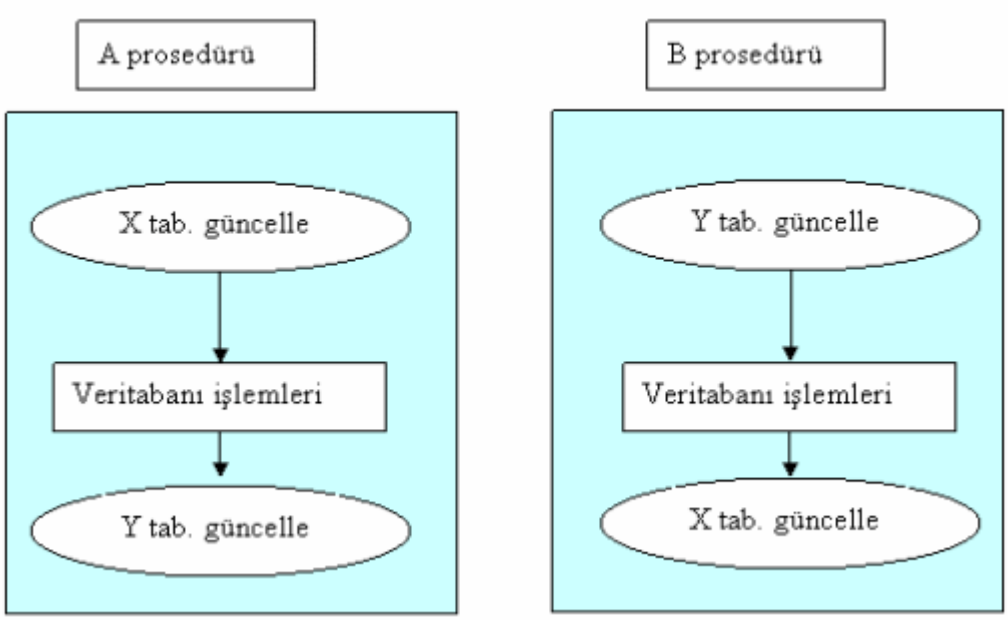
```
    WHERE hesapNo=@aliciHesap
    IF @@ERROR<>0
      ROLLBACK
      RETURN 0
  COMMIT
  RETURN 1
GO
```

Test etmek için,

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

Ölümcül Kilitlenme(DEAD LOCK) Nedir ve Nasıl Önlenir?

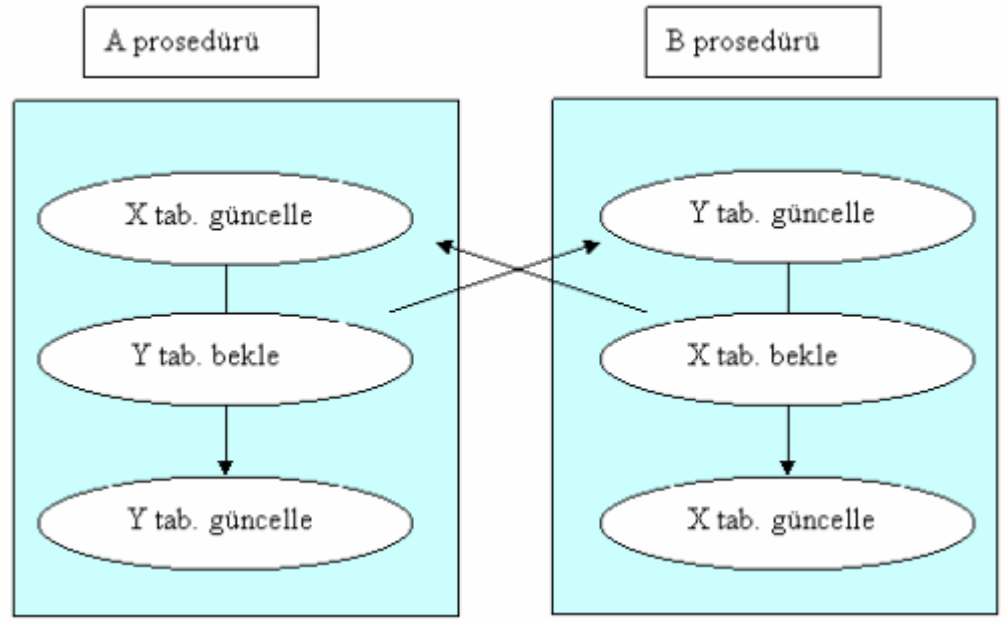
Ölümcül kilitlenme iki veya daha fazla prosesin karşılıklı olarak birbirlerinin kilitlediği kaynaklara erişmek istemesidir. Özellikle büyük çaplı projelerde çokca karşılaşılır. Her iki prosesde sürekli birbirlerini beklediği için sistem kaynakları olumsuz yönde etkilenir. Özellikle server ın CPU değeri boşuna harcanmış olur. Bu da serverımızın performansını çok kötü etkiler hatta müdahale edilmeyen ölümcül kilitlenmelerde server cevap veremez duruma bile gelebilir. Örnek bir senaryo üzerinde ölümcül kilitlenmenin nasıl gerçekleştiğini anlatalım.



Elimizde iki tane prosedürümüz var bir kullanıcı A prosedürünü çalıştırıyor diğer bir kullanıcı ise B prosedürünü çalıştırıyor. Bu iki prosedürde içerisinde birden fazla işlem update barındıran prosedürlerdir. Böyle bir durumda bu iki prosedür tüm işlemlerini yaptıktan sonra verileri veritabanına yazar. Bu prosedürler kendi içerisinde TRANSACTION gibi çalışır. Öncelikle iki prosedür ilk işlemlerini yapıyorlar.

Yani A prosedürü X tablosunun ilgili satırını güncelliyor, B tablosuda Y tablosunun ilgili satırını güncelliyor. Güncelleme yapılırken proses bitene kadar COMMIT yani tüm işlemler tamam artık bunu veritabanına yazabilirsin denilene kadar(bu iki prosedürde tüm güncellemelerini yaptıktan sonra COMMIT işlemini yapar) yapılan tüm işlemlerden sonra işlem yapılan kayıtlara kilit koyar ve başka bir prosesin bu kayıtlara erişmesini engeller. Neden olarak ise bir tabloda yaptığımız bir işlem başka prosesleri de etkileyecektir bu durumda başka prosesler o işlemden önceki verileri mi yoksa o işlem yapılırken ki verileri alacağına karar veremez. Belki prosedür içerisinde herhangi bir nedenden dolayı işlem yapılan tablodaki veriler ROLLBACK yani geri alma işlemi yapacaktır.

Şimdiki durumda A prosedürü Y tablosunda işlem yapmak için B prosedürünün kilit koyduğu kilitin kalkmasını beklemektedir yine aynı durumda B proseside X tablosunda işlem yapmak için A prosesinin X tablosuna koyduğu kiliti kaldırmasını beklemektedir. Burada bir döngü oluşmakta ve her iki prosesde birbirini beklemektedir. Bu iki prosesde serverda asılı kalmaktadır.



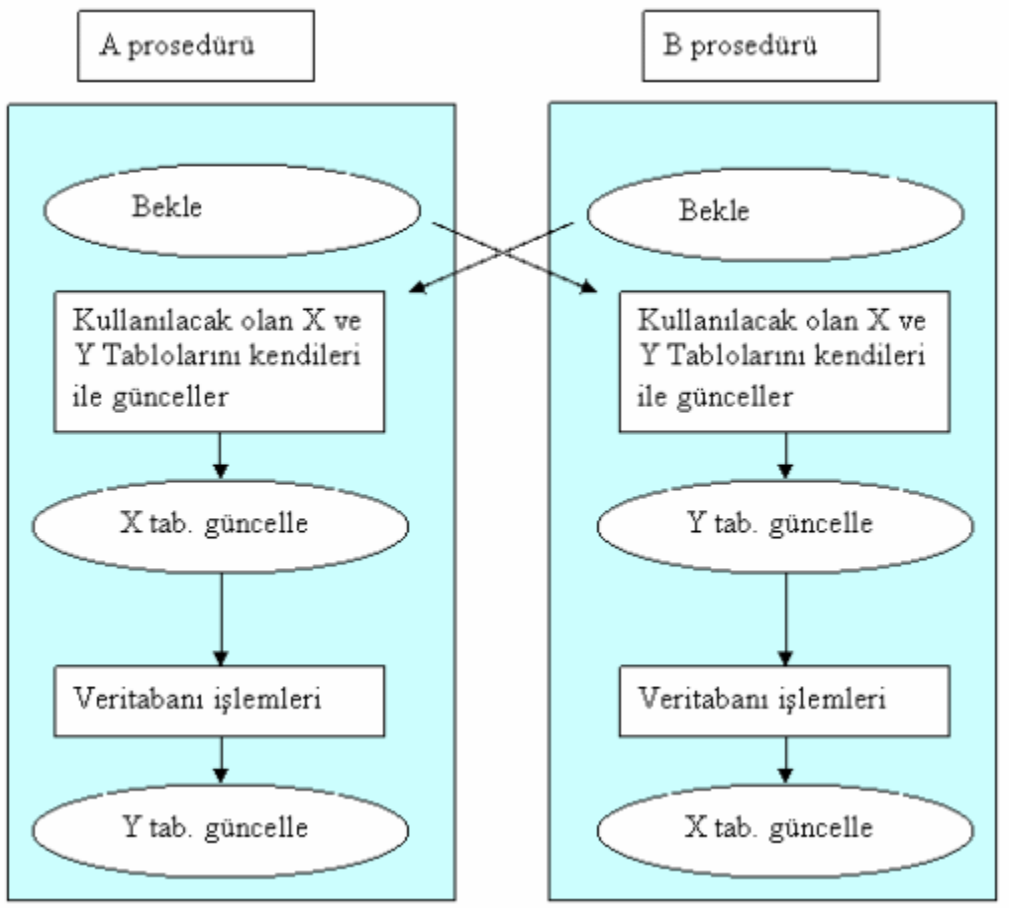
Peki bu ölümcül kilitlenme nasıl önlenir?

Öncelikle çözüm için A prosedüründe ve B prosedüründe ilk güncelleme işlemlerinden sonra Commit konulup kullanılan tablolardaki kiliti çözmek gelebilir. Fakat Commit koymak hatalı bir çözümdür. Çünkü proses bitene kadar proses içerisinde bir hata ya da istenmeyen bir durum oluştuğunda rollback(geri alma) yapıldığında ancak commit olan yere kadar olan bölümü kurtarabilirsiniz. Commit yukarısındaki bölümü kurtarmak için ayrı bir çalışma yapmak gerekmektedir. Commit in bir diğer sakıncası ise her commit işleminden sonra veritabanında rollback segmentleri oluşur yani iki commit arasında yapılan işlemler server tarafından tutulur. Çok fazla commit yapılması hem prosedürün hızını yavaşlatır hemde serverın gereğinden fazla şişmesini sağlar.

Çözüm için bir diğer yöntem ise sql server üzerinde ölümcül kilitlenme(dead lock) olan prosesleri manuel olaraka tespit edip kapatmak ile olabilir. Burada özellikle sql admine çok fazla iş düşer. Sp_who prosesini çalıştırarak ölümcül kilitlenen olan tablolar görülebilir. Management Studio/ management/ ActivityMonitor kullanılarak halen aktif olan kilitlinmeler görülüp proseslerin üzerinde kill yapılarak ölümcül kilitlenmeler çözülebilir.

Diğer bir yöntem ise sql server ın ölümcül kilitlenmeyi farketmesi halinde daha az önceliği olan prosesi kill ederek diğer bekleyen prosese yol açması şeklindedir. Sql serverın hangi prosesin öncelikli olduğunu bilmesi için DEADLOCK_PRIORITY parametresinin LOW ya da NORMAL olarak eşitlemek gerekmektedir. Bu yöntemde her zaman dead lock ın yakalanabileceği garanti değildir. Yakalansa bile bu yakalamaya kadar geçen süre çok uzun olduğu için sistem kaynakları olumsuz yönde etkilenecektir.

Tam olarak bir çözüm yöntemi olmasada en çok kullanılan ve tercih edilen yöntem ise proseslerin başlarında kullanılacak olan tabloların ilgili satırlarını bir grup biçiminde kendisi ile update edilerek lock edilmesi ve daha sonra bu tablolar üzerinde işlemlerin yapılmasıdır. Yani A prosesi düşünülünce X ve Y arasında yapılan işlemler ve bu işlemlerin gerçeklenme süreleri yok edilerek deadlock a girme olasılığının bitirilmesidir. A prosesi ilk başlandığında X ve Y tablolarının ilgili kayıtları kendileri ile güncellenerek lock koyulur aynı işlem B prosesi için de yapılır A ve B den hangisi daha önce bu ilk adımı geçerse diğer proses geçenin işlemlerini yapmasını bekleyecektir. Şekille göstermek gerekirse;



Deadlock’a düşmemek için yapılabilecek diğer işlemler ise,

- Mümkün olduğunda az sayıda tablo üzerinde işlem yapacak prosedürler yazmak. Gereksiz işlemler ile prosedürü yormamak.
- Prosedürlerin çok hızlı çalışmalarını sağlamak.
- Proses içerisinde yapılacak işlemleri mümkünse aynı tablo sırasında yapmak (Mesala, bizim örneğimiz için her iki prosesinde önce X i sonra Y yi güncellemesi.)
- WITH(NOLOCK) kullanılarak yapılan seçim işlemlerinde eğer o satır üzerinde LOCK varsa Lock eden prosedür işlemini tamamlamadan önceki veri alınır ve sorgu lock ı beklemek zorunda kalmaz. Fakat alınan veri her an değişecek olması göz önüne alınmalıdır. Kullanılışı: (SELECT * FROM TABLO_ISMI WITH(NOLOCK)) şeklindedir.