DATABASE

Öğr. Gör. Evgin GÖÇERİ

24 Kasım 2014

Ortak Zamanlılık ve Transaction

Transaction nedir?

- Bazı durumlarda, birden fazla işlem bir bütünün parçasıdır. Bu işlemlerden biri bile gerçekleşmese bütün işlemler anlamsız kalabilir
- Bu türden işlemlerin tamamının bir tek işlem gibi ele alınması gerekiyor demektir
- Parçalanamaz işlemlerin oluşturduğu yeni bir tek işleme ***transaction*** denir. O halde transaction, daha küçük parçacıklara ayrılamayan işlem bloğu demektir

Bir transaction bloğu SQL server tarafından şu şekilde ele alınır:

1. **Transaction bloğu başlatılır**. Bu andan itibaren yapılacak işlemlerin bütünsellik arzettiği ve her an tamamının geçersiz sayılabileceği belirtilmiş olur. Bu işlem transaction log'lar sayesinde desteklenir.

Transaction bloğu çeşitli şekillerde başlatılabilir. SQL server tarafından otomatik olarak veya kullanıcı tarafından BEGIN TRANSACTION ile başlatılabilir (BEGIN TRANSACTION yerine daha kısa olarak BEGIN TRAN da kullanılabilir)

Bir transaction bloğu SQL server tarafından şu şekilde ele alınır:

2. Transaction bloğu arasında yapılan her işlem bittiği anda başarılı olup olmadığına bakılır, başarılı olmadığı anda geri alım işlemine geçilir (ROLLBACK)

Başarılı ise, bir sonraki işleme geçilir. Bu işlemde bazen kullanıcı tarafından yapılsa da genellikle SQL server tarafından yapılır

Bir transaction bloğu SQL server tarafından şu şekilde ele alınır:

3. Tüm işlemler tamamlandığı anda (veritabanı tekrardan tutarlı bir hale geldiğinde) COMMIT ile yeni hali ile sabitlenir

Başarısız bir sonuç ise ROLLBACK ile en başa alınır ve bilgiler ilk hali ile sabitlenir.

Bu iki ifade de Transaction'ı bitirir

Banka işlemi örneği

Bir kullanıcı başka bir kullanıcıya havale yaptığında ne olur?

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 yada program takılabilir. Bu durumda ne olur?

Gönderenin hesabından para düşülmüştür fakat alıcının hesabına geçmemiştir. Bir kısım paranın sahibinin kimliği kaybolmuştur. *Bu da sistemin olası durumlar dışında veri kaybetmeye müsait bir hal alması demektir*

ACID (Atomicity, Consistency, Isolation, Durability)

- Bir veritabanı veriler üstünde değişiklik yaparken ACID kuralını sağlamalıdır. Bu kurallar;

1. Atomicity (Bölünmezlik): Transaction bloğu yarım kalmaz. Ya hepsi gerçekleşmiş sayılır veya hiçbir işlem gerçekleşmemiş gibi kabul edilerek en başa dönülür. Yani, transaction daha küçük parçalara ayrılamayan bir işlem birimi olarak ele alınır

2. Consistency (Tutarlılık): Transaction veritabanının yapısını bozmadan işlem bloğunu terk etmelidir. Yani ara işlemler yaparken ürettiği işlem parçacıklarının etkisini veritabanında bırakarak, transaction'ı sonlandıramaz. (Örneğin, birinci hesaptan para azaltıp ikinci hesaba eklemeden transaction sonlandırılamaz)

ACID (Atomicity, Consistency, Isolation, Durability)

3. Isolation (İzolasyon): Farklı transaction'lar birbirinden ayrık ele alınmalıdır. Her transaction için veritabanının yapısı ayrı ayrı korunmalıdır. İlk transaction tarafından yapılan değişiklikler, ikinci transaction'dan her an görülememeli, sadece bütün işlem gerçekleştiği anda ve bütünü bir anda görülmelidir

4. Durability (Dayanıklılık): Tamamlanmış transaction'ın hatalara karşı esnek olması gerekir. Elektrik kesilmesi, CPU yanması, işletim sisteminin çökmesi bu kuralları uygulamaya engel olmamalıdır. Bunun içinde gerçekleşmiş ve başarılı olarak sonlanmış transaction'ın değişikliklerinin kalıcı olarak diske yansıtılması gerekir

Transaction bloğu nasıl ele alınır?

- SQL server üstünde herhangi bir veride değişiklik yapıldığı zaman, ilgili sayfalar daha önce diskten hafızaya (RAM) çağrılmamışsa öncelikle tampon denilen belleğe çağrılır
- Daha sonra üstünde değişiklikler yapılır
- Bu değişiklikler diske hemen yansıtılmaz
- Bu şekilde içeriği değişmiş fakat diske henüz kaydedilmemiş sayfalara kirli sayfa (*dirty page*) denir
- Sayfada meydana gelen her değişiklik, *.ldb uzantılı transaction log dosyasına anlık olarak kaydedilir.
- Kirli sayfalardaki değişikliğin diske kaydedilmesi işlemi için de arındırma (flushing) terimi kullanılır

Transaction Modları

- SQL server 2 temel transaction mod için destek verir:
  Harici (Explicit) ve Dahili (Implicit) transaction.

Bunlardan hiç birisinin geçerli olmadığı durumda Otomatik (Autocommit) adı verilen modda çalışır (Bu modda tüm işlemleri SQL server denetler)

Harici Transaction, programcı tarafından başlatılan ve sonlandırılan işlem bloğu için kullanılan bir terimdir. BEGIN TRAN ile başlatılıp, ROLLBACK TRAN veya COMMIT TRAN ile sonlandırılan bloktur

Dahili Transaction ise, belli ifadelerin çalıştırılması halinde kendiliğinden devreye girebilen fakat kullanıcı tarafından sona erdirilmesi istenilen bir transaction şeklidir

Harici Transaction

- Harici transaction'lar BEGIN TRAN ile başlatılıp, COMMIT ROLLBACK ile bitirilebilirler. Ancak, herhangi bir iç hata oluşması veya güç kesilmesi gibi hallerde SQL server de otomatik olarak verilerin korunmasını destekler

- BEGIN TRAN [SACTION] [transaction_ismi]
  Transaction kod bloğu
  COMMIT TRAN | ROLLBACK

Harici transaction'lar pek tavsiye edilmese de iç içe de kullanılabilir. Bu durumda açık transaction kalmamasına dikkat edilmelidir. Aksi halde sistem kaynaklari daha fazla kullanılabilir yada bazı sistem kaynaklarına başkalarının erişimi engellenmiş olur

Açık transaction kalıp kalmadığını anlamanın yolu, @@TRANCOUNT değerine bakmaktır. @@TRANCOUNT değeri anlık olarak SQL server üstünde kaç adet açık transaction bulunduğunu gösterir

Örnek:

Aşağıda bir banka hesaplarının tutulduğu tablo yapısı mevcuttur

```
CREATE TABLE tblHesap(
        HesapNo CHAR(10) PRIMARY KEY NOT NULL,
        isim VARCHAR(55), Soyad VARCHAR(55), sube INTEGER,
    bakiye MONEY)
```

İki adet kayıt ekleyelim:

```
INSERT tblHesap VALUES ('000000023', 'Ali', 'Eryatan', 749,
    30000000)
INSERT tblHesap VALUES ('000000042', 'Ahmet', 'Eryatan', 749,
    30000000)
```

Örnek:

Bu tablo üstünde bir stored procedure tanımlayalım. Bu procedure dışardan havale yapanın hesap numarasını, havaleyi alacak olanın hesap numarasını ve havale miktarını alsın ve transaction başlattıktan sonra havaleyi yapsın. Havalede sorun çıkmazsa geçerli kabul etsin, çıkarsa her aşamada ROLLBACK ile durumu geri alsın

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

TRY – CATCH ile Transaction

- @@ERROR değerinin sıfırdan farklılığını kontrol etmek yerine TRY-CATCH yapısını tercih etmek daha pratiktir. Örnek yeniden düzenlenirse,

```
ALTER PROC SP$havale(
@aliciHesap CHAR(10), @gonderenHesap CHAR(10), miktar MONEY)
AS
  BEGIN TRY
     BEGIN TRANSACTION
     UPDATE tblHesap
     SET bakiye = bakiye - @miktar
     WHERE hesapNo = @gonderenHesap

     UPDATE tblHesap
     SET bakiye = bakiye +@miktar
     WHERE hesapNo = @aliciHesap
     COMMIT
  END TRY

BEGIN CATCH
 print @@ERROR + 'kodlu hata olduğundan havale yapılamadı '
 ROLLBACK
END CATCH
```

Sabitleme Noktaları

- Bazen, bir noktaya gelindikten sonra, işlemlerin buraya kadar olanını geçerli kabul etmek isteriz ama, bundan sonraki işlemler içinde transaction (geri alabilme seçeneğine) ihtiyaç duyarız

- Bu türden durumlarda sabitleme noktalarından faydalanılır

- Bir sabitleme noktası başlatıldığı anda, işlem bloğunda bir şeyler ters giderse *en başa dönme seçeneği saklı kalmak üzere, noktanın oluşturulduğu yere de dönme seçeneği* sunar

- Bu nedenle sabitleme noktaları kitap ipine benzetilir. Çünkü SQL Server'ın hangi aşamaya dönebileceğini tanımlamamızı sağlar

Sabitleme Noktaları

- Bir sabitleme noktası şu şekilde tanımlanır:

```
SAVE TRANSACTION sabitleme_noktasi_adi
```

Burada sabitleme noktasına bir ad vermek gerekir ki daha sonra herhangi bir anda ilgili sabitleme noktasına şu şekilde dönülebilsin:

```
ROLLBACK TRAN sabitleme_noktasi_adi
```

Dahili (Implicit) Transaction

Dahili transaction için öncelikle

IMPLICIT_TRANSACTION ON

deyimi kullanılarak sistemin belli ifadelerden önce otomatik başlatması sağlanır. Ancak, transaction işlem tarafından bitirilmez, kullanıcının bitirmesi gerekir

Transaction şu ifadelerden sonra başlatılır: ALTER TABLE, CREATE TABLE, DELETE, DROP, INSERT, SELECT, UPDATE

Daha sonra autocommit moda dönmek için,

IMPLICIT_TRANSACTION OFF

ifadesi kullanılır

Ortak Zamanlılık ve İzolasyon Seviyeleri

Kurumsal manadaki bir veritabanı yönetim sistemi yazılımının aynı anda çok sayıda kullanıcıya ortak veriler üstünde çalışma olanağı sağlaması gerekir.

Ortak zamanlı aynı anda aynı işle ilgili olmasa da, ortak zamanlarda ortak kaynaklardan iş yapmak anlamında kullanılan bir terimdir

Aynı makaleyi değerlendiren bir yazar ile bir editör ortak zamanlı çalışıyor demektir. Çünkü makale ortak kaynak konumundadır

Ancak ortak zamanlı kullanıcı desteğinde transaction blokları oluşturmanın doğurabileceği tanımlanmış sakıncalı durumlar vardır. Bu durumlar ortak zamanlılık problemleri olarak ele alınır

İzolasyon Sağlamak

Farklı transaction'ları aynı anda yönetirken, erişilen kaynaklardaki değişimlerin bir diğer transaction tarafından görünebilirliğini veya aynı anda yansıtılabilir olmasını ayarlamak için iki temel yöntem kullanılır; Kilitleme ve Satır Versiyonlama

Kilitleme temelli olarak sağlanan ortak zamanlılıkta, her bir transaction farklı büyüklükte kaynak kilitleyebilir. Bir transaction bitimine kadar, bir satırı, bir sayfayı veya bir tablonun tamamını başka transaction'ların erişimine kapatabilir

Satır versiyonlama ile izolasyon yapabilmek için önceden veritabanı seviyesinde ayarlamalar yapmak gerekir. Bu ayarlamalar yapıldıktan sonra satır seviyeli bir veri değişim versiyonlaması yapılır. SQL server, transaction başladığı anda verilerin transaction için bir mantıksal kopyasını oluşturur. Daha sonra bu veriler üstünden değişimleri takip ettiği için ayrıca gerçek verileri dışarıya karşı kilitleme gereksinimi ortadan kalkar. Böylece özellikle okumalar tarafından engellenen diğer transactionlar nedeniyle ortaya çıkacak, kilitlemelerin önüne geçilmiş olur

Ortak Zamanlı Erişim Problemleri

Kilitleme ve versiyonlamada temel amaç; Aynı verilerin farklı transactionlar tarafından aynı anda değiştirilmesi ile ortaya çıkacak kargaşayı engellemek ve kirli sayfalardaki verilerin başka transactionlar tarafından okunarak yanıltıcı sonuçlar elde edilmesinin önüne geçmektir

Veriler aynı anda birden fazla transaction tarafından değiştirilir veya değiştirmeler sonlanmadan başka transactionlar tarafından okunursa bazı sorunlar ortaya çıkması olasıdır. Bu sorunlara **ortak zamanlı erişim problemleri** denir.

1.Kayıp güncelleme (Lost Update) : Aynı anda bir çok transaction aynı kayıt üzerinde işlem yapıyorsa, transactionlardan sadece son kapanın dediği olur. Diğerlerinin yaptığı değişiklikler kaybedilir

Kayıp Güncelleme Problemi

Siz A acentasından bir uçak bileti aldınız. Aynı anda B acentasından bir başka yolcu daha bilet aldı ve sizin bilet alma işleminizi gerçekleştiren transaction biraz geç COMMIT oldu diyelim.

Neticede aynı havayolunun aynı koltuğunu satın aldınızsa ciddi sorunlar yaşayacağınız muhakkak.

Sorunun çözümü, A acentası bilet satmak için bir transaction başlattığı anda, B acentasının aynı kayıt üstünde transaction başlatmaması sağlanmalıdır.

Veya versiyonlama temelli çalışılacaksa, VTYS'nin UPDATE çakışması fark edilip, ikinci UPDATE'e hata vermesi gerekir (SQL server 2005 bu şekilde davranmaktadır)

Hayalet Okuma Problemi (Phantom Read)

Bir transaction ilk okuyabildiği verileri daha sonra tekrardan okuyabiliyordur; çünkü değişikliklere (UPDATE) karşı kilitleme yapmıştır. Ancak bunların dışında yeni veri eklenmesine (INSERT) engel olamıyordur.

Bu türden bir durumda da **yeni eklenen veriler** değerlendirme dışında kalacaktır.

Örneğin, mağazamızda bir kampanya düzenledik ve 1 adetten az satan ürünlere %4 indirim uygulamak üzere bir transaction açtık. Ancak aynı anda sistemde birileri yeni ürünler giriyorsa, bizim transaction'ımızın başlamasından sonra girilen yeni ürünler hiç satılmamış olmasına rağmen bu indirim oranlara yansıtılmayacaktır. Çözümü için yeni ürün eklenmesine de kilit koymak gerekir

Kirli Okuma (Dirty Read)

Aynı kaynağa erişen birden fazla transaction'dan biri, işlemini sona erdirmeden önce diğer transaction'lar tarafından okunan kayıtlar gerçeği yansıtmaz.

Çünkü, birinci transaction diğer transaction'lar okuma yapmadan kayıtlar üstünde okuma yaparsa ardından okuma işleminden sonrada değişikliklerden vazgeçerse, diğer transaction'lar yanlış değerler okumuş olur

Mağaza örneğinde, mağaza yönetimince fiyatlara zam yapmak için bir transaction başlatıldığını varsayalım. Aynı zamanda bir müşteride sipariş vermiş olsun.

Mağaza otomasyonu fiyatlandırmada bir aksilikle karşılaşıp, transaction'ı geri aldığı anda, sipariş alınmış ve bitmiş varsayalım. Ürün müşteriye anlık olarak çok yüksek fiyatla satılmış olur. Önlemek için, ikinci transaction'ın kirli sayfaları transaction bitene kadar okumasını engellemek gerekir

İzolasyon Seviyeleri

Aynı anda veritabanına kaynaklara erişen iki transaction'ın bir diğerinin değişim ve kaynak erişimlerinden ne derece izole tutulacağı ile ilgili çeşitli ayarlamalar yapılabilir

Read Uncommited: İzolasyon seviyesi 0'dır. Yani hiçbir izolasyon yoktur. Transaction açıkken başka transactionda açık olarak veri değiştirebilir. Bütün ortak zamanlılık problemleri görülür

Read Commited: Veri okunurken, kirli okumayı önler. Ama transaction bitmeden veri değiştirilebilir. SQL serverda default seçenek budur

Repeatable Read: Transaction içerisindeki sorguda geçen bütün veriler kilitlemeye alınır. Dirty read sorununa çözüm sağlar çünkü kirli hafızanın okunmasına müsaade etmez

Serializable: Ortak kaynaklara aynı anda bir transaction sonlanmadan, ikinci bir transaction ulaşamaz. Böyle bir durumda, hiçbir ortak zamanlılık problemi doğal olarak görülmez

İzolasyon Seviyeleri

İzolasyon Seviyesi	Açıklama	DIRTY READ	NONREPEATABLEREAD	PHANTOM READ
Read Uncommitted	Herkes Herşeyi Yapabilir	+	+	+
Read Committed	Sadece commit edilmiş verileri okuyabilir. Ama yeni veri girilmesine ve erişilen verinin değişimine kilit koymaz	-	+	+
Repeatable Read	Birinci transaction okuduğu bölgeleri değişikliğe karşı kilitler Ama yeni kayıt eklenmesine engel olmaz	-	-	+
Serializable	Ortak kaynağa aynı zamanda sadece bir transaction erişebilir	-	-	-