Transaction ve Ortak Zamanlılık

Sibel SOMYÜREK

Nedir?

* Daha küçük parçalara ayrılamayan en küçük işlem yığını
* Hepsi tek bir SQL ifadesiymiş gibi yürütülen SQL komutları kümesidir.

Nedir?

* Transaction'ın tamamlanabilmesi ve verilerdeki değişikliklerin kaydedilebilmesi için transaction'da yer alan tüm SQL komutlarının gerçekleştirilmesi gerekir.

Nedir?

* Eğer bir transaction herhangi bir nedenle tamamlanmazsa:
  * Transaction'dan ötürü veri kümesinde meydana gelen değişiklikler iptal edilir ve
  * Veritabanı transaction başlamadan önceki orijinal durumuna geri döndürülür.

Neden kullanılması gerekir?

* Bankamatikten para çekeceğimiz zaman çekilecek para miktarı yazılıp işlem başlatıldığında bankamatiğin bozulması veya merkezle bağlantısının kesilmesi gibi herhangi bir nedenle işlem yarıda kesilirse ne olur?
  - İhtimallerden birisi, para hesaptan düşülmüştür ama bankamatik tarafından verilmemiştir olabilir.

Neden kullanılması gerekir?

* Bu durumda bir kısım paranın sahibinin kimliği kaybedilmiş olur.
* Bu da sistemin olası durumlar dışında veri kaybetmeye müsait bir hal alması demektir ve bu gibi durumların önlenmesi gerekir.

Ne işe yarar?

* 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 gibi kabul edilir.

Transaction ve Ortak Zamanlılık

- Transaction tanımına girebilen işlemler UPDATE, INSERT ve DELETE işlemleridir.

Transaction Mantığı

* Bir Transaction bloğunu işletmenin temel mantığı şu şekildedir:
  1. transaction bloğu başlatılır. Böylece yapılan işlemlerin geçersiz sayılabileceği Veri Tabanı Yönetim Sistemi'ne bildirilmiş olur. Böylelikle VTYS işlemlere başlamadan önceki halin bir kopyasını tempdb 'de tutmaya başlar.

Transaction ve Ortak Zamanlılık

2) Transaction bloğu arasında yapılan her bir işlem bittiği anda başarılı olup olmadığına bakılır.

3) Tüm işlemler

* başarıyla tamamlandığı anda **COMMIT** ile bilgiler yeni hali ile sabitlenir
* başarısız bir sonuç ise **ROLLBACK** ile en başa alınır ve bilgiler ilk hali ile sabitlenir.

Transaction Oluşturma

* Transaction başlatmak için:
  * START TRANSACTION;
  * BEGIN
  * BEGIN WORK
* Transaction'ı sonlandırmak için:
  * Commit(başarılı)
  * Rollback(başarısız)

Transaction Oluşturma Örnek-1 1/3

```
Create table satis_bilgileri(
    id INT(11) Primary key,
    adet Smallint(3),
    Tarih timestamp
)
INSERT INTO satis_bilgileri (id, adet) VALUES (1,3), (2, 1);
```

Transaction Oluşturma Örnek-1 2/3

```
Create table stok(
    id INT(11) Primary key,
    ad Varchar(25),
    adet Smallint(3)
)
INSERT INTO stok (id, ad, adet) VALUES (1, 'Lenova ', 125), (2, 'Sony', 432);
```

Transaction Oluşturma Örnek-1 3/3

* Örnek:

```
START TRANSACTION;
    UPDATE satis_bilgileri SET adet = adet+3 WHERE id = 1;
    UPDATE stok SET adet = adet-3 WHERE id = 1;
COMMIT;
```

Transaction Oluşturma Örnek-2

```
CREATE TABLE ogrenciler (
    id INT PRIMARY KEY AUTO_INCREMENT,
    ad VARCHAR(8)
) ENGINE=InnoDB;
```

Transaction Oluşturma Örnek-2

INSERT INTO ogrenciler (ad) VALUES ('Ayşe'), ('Fatma');

START TRANSACTION;

UPDATE ogrenciler SET ad='Defne' WHERE id = 1;

UPDATE ogrenciler SET ad='Elif' WHERE id = 2;

ROLLBACK

-> Hiçbir işlem gerçekleşmez

Transaction Oluşturma Örnek-2

INSERT INTO ogrenciler (ad) VALUES ('Ayşe'), ('Fatma');

START TRANSACTION;

UPDATE ogrenciler SET ad='Defne' WHERE id = 1;

UPDATE ogrenciler SET ad='Elif' WHERE id = 2;

COMMIT

->İşlem gerçekleşir

ROLLBACK

* Bazı SQL komutları (DDL) rollback ile geri alınamaz:
  * CREATE DATABASE,
  * CREATE TABLE,
  * DROP DATABASE,
  * DROP TABLE
  * ALTER TABLE

Sabitleme Noktaları

* Bir noktaya gelindikten sonra, işlemlerin buraya kadar olanını geçerli kabul etmek amacıyla sabitleme noktalarından faydalanılır
  * SAVE TRANSACTION sabitleme_noktasi_adi
  * ROLLBACK TRAN sabitleme_noktasi_adi

Örnek

```
CREATE TABLE trans_test (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(8)
) ENGINE=InnoDB;
```

Örnek

```
INSERT INTO trans_test (name) VALUES ("a"), ("b");

START TRANSACTION;
    UPDATE trans_test SET name="z" WHERE id = 1;
    SAVEPOINT savepoint_one;
    UPDATE trans_test SET name="y" WHERE id = 2;
    ROLLBACK TO SAVEPOINT savepoint_one;
COMMIT
```

Transaction Desteği

* Mysql çeşitli depolama motorlarını destekler. Depolama motorları veritabanı tablolarını oluşturmak ve desteklemek için kullanılan sunucu bileşenleridir.
  * MyISAM- transactionları desteklemez
  * InnoDB ve Falcon- transactionları destekler

ACID (Atomicity, Consistency, Isolation, Durability)

* Güvenli bir veritabanı veriler üstünde değişiklik yaparken ACID kuralını sağlamalıdır. Bu kurallar;
  1. Atomicity (Bölünmezlik)
  2. Consistency (Tutarlılık)
  3. Isolation (İzolasyon)
  4. Durability (Dayanıklılık)

ACID (Atomicity, Consistency, Isolation, Durability)

* 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

ACID (Atomicity, Consistency, Isolation, Durability)

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

ACID (Atomicity, Consistency, Isolation, Durability)

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

AUTOCOMMIT

* Start Transaction gibi geleneksel bir ifadeyle transaction başlatmak yerine AUTOCOMMIT ifadesi kullanılabilir.
* Eğer autocommit aktifleştirildiyse(autocommit=1) her ifade bir transaction olarak algılanır.
  * Yeni bir transaction başlamadıkça önceki SQL komutlarının tamamı bir transaction olarak işlem görür.
* Autocommit'i iptal etmek için:
  * SET autocommit=0

AUTOCOMMIT

* Autocommit ifadesi kullanıldığında ayrıca Start ya da Begin Transaction ifadesinin kullanılmasına gerek kalmaz.

```
SET AUTOCOMMIT=0;
UPDATE trans_test SET name='a' WHERE id='1';
COMMIT;
```

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 zamanlılık, ortak zamanlarda ortak kaynaklardan iş yapmak anlamında kullanılan bir terimdir

Ortak Zamanlılık ve İzolasyon Seviyeleri

* Örneğin:
  * Aynı rapor üzerinde çalışan 2 öğrenci ortak zamanlı çalışıyor demektir. Çünkü rapor ortak kaynak konumundadır.

Transaction ve Ortak Zamanlılık

* Ortak zamanlarda aynı kaynağa erişim, yeterli önlemler alınmadığında çeşitli sakıncalar doğurabilir. Bu sakıncalar 4 ana problem olarak standardize edilmiştir:
    - Güncellemede Kayıp (Lost Update)
    - Tekrarlamasız Okuma (Non-Repeatable Read)
    - Hayalet Okuma (Phantom Read)
    - Kirli Okuma (Dirty Read)

Transaction ve Ortak Zamanlılık

Güncellemede Kayıp (Lost Update)

* Örneğin, bir iş anlaşmasını ele elalım.
* İki yetkili tarafından açılmış ve düzenlemeler yapılıyor olsun.
* Yetkililerden biri düzenlemeler yapıp kaydettiğinde, ikinci yetkili bu düzenlemeleri asla fark etmeyecektir ve ilk kaydetme işleminde birinci yetkilinin yaptığı düzenlemeler kaybolacaktır.
* Bu sorunu engellemek için, ortak zamanlılığı, birinci yetkilinin işi bitinceye kadar kaynağa erişimi kısıtlayacak şekilde düzenlemek(kilitlemek) gerekir.

Transaction ve Ortak Zamanlılık

Kirli Okuma (Dirty Read)

* Bir transaction tarafından değiştirilmiş ama henüz kalıcı hale getirilmemiş bir bilginin başka bir transaction tarafından gerçek kayıtmış gibi okunmasıdır.
* Örneğin, iş anlaşma taslağını ele alacak olursak, yetkililerden biri üstünde değişiklik yapıp henüz işini bitirmediği halde değişikliği kaydettiği anda başka biri tarafından sözleşmenin taraflara iletilmesi, bir kirli okuma problemi doğuracaktır.
* Bu sorunu engellemek için de yine birinci yetkilinin bütün işlemleri yapıp sözleşmenin son halini kaydetmesine kadar, anlaşmaya erişimi ve dağıtılmasını engellemek gerekir.

Transaction ve Ortak Zamanlılık

Tekrarlamasız Okuma (Non-Repeatable Read)

* İkinci bir transaction bir satıra her erişiminde farklı bir değer okuyorsa tekrarlamasız okuma sorunu ortaya çıkar.
* İş anlaşma metni örneğine dönecek olursak, yetkililerden biri sözleşmenin son halini görmek istediğinde bir göz atıp, kapatıyor.
* Ardından tekrar bakıyor ve sözleşmenin değiştiğini görüyorsa bu türden bir sorun ortaya çıkacaktır.
* Sorunu çözmek için, ortak zamanlılığı şu şekilde düzenlemek gerekir: Sözleşme taslak halde iken, yani üstünde biri düzeltme yaparken, sözleşmenin son halini görmek isteyen erişimleri engellemek(kilitleme) gerekir.

Transaction ve Ortak Zamanlılık

Hayalet Okuma (Phantom Read) :

* Transaction'lardan biri silme veya ekleme (Insert-Update) işlemleri gerçekleşirken diğer bir transaction bu işlemleri içine alan aralıktaki satırlar üstünde okuma yapıyorsa, ya elde ettiği satırlardan bir kısmı artık yoktur veya elde ettiği satırlar, bazı satırları eksik bulunduruyordur.
* İş anlaşmasındaki hali ile sözleşme taslağı üstünde taraflardan biri değişiklikler yapıp avukatına gönderdiğinde, avukatı bu değişikliklerin asıl sözleşme ile uyuşmadığını belirtir.
* Durum ayrıntılı olarak incelendiğinde, diğer taraftan yeni bir sözleşme metni maddesi eklendiği veya bir maddenin çıkartıldığı için böyle bir sorunun oluştuğu anlaşılırsa hayalet okuma sorunu ortaya çıkmış olur.

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
  * Read Commited
  * Repeatable Read
  * Serializable

İzolasyon Seviyeleri

* **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

İzolasyon Seviyeleri

* **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 Sağlama

* 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 (Locking)

* 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 kapatılabilir

Satır versiyonlama

* Satır versiyonlama ise satırların eski versiyonlarının tempdb veritabanında saklanmasıdır.
* Satır güncellenmeden önce eski versiyon transaction tarafından okunabilir.
* Böylece bazı verileri kilitlemeye gerek duyulmaz.
* Satır versiyonlama

Kaynakça

* Sheeri K. Cabral & Keith Murphy (2009). MySQL Administrator's Bible . Wiley Publishin, Inc.
* Gözüdeli, Y. (2004). SQL Server'de Ortak Zamanlılık ve İzolasyon. http://www.csharpnedir.com/articles/read/?id=406&title=SQL%20Server'de%20Ortak%20Zamanl%C4%B1l%C4%B1k%20ve%20%C4%B0zolasyon