

Veritabanı Programlama

Bölüm 11

Transaction

Dr. Öğr. Üyesi Murat TAŞYÜREK (kayubmprogramlama1@gmail.com)

21 Aralık 2023

Kayseri Üniversitesi, Bilgisayar Mühendisliği Bölümü

Transaction gereksinim

- Veritabanı sunucusu birçok işlemi doğru, eş zamanlı ve performanslı sorgulamak için optimize edebilecek yeteneklere sahiptir (sahip olmalıdır).
- Bir sorgunun işlem yaptığı satırlara başka kullanıcılar ya da sorgular da ulaşmak ve üzerinde değişiklik yapmak isteyebilir.
- Az sayıda kullanıcısı olan veya küçük uygulamalarda bu tür durumlar sorun oluşturmayabilir.
- Ancak bankacılık ya da benzeri büyük veri işlemleri gerçekleştiren çok istemcili veritabanı mimarilerinde aynı anda çok yoğun sorgular gerçekleştirilir.

Transaction

- Bir ya da birkaç tablo üzerinde veri değiştirme işlemi gerçekleştirilecekse ve değişiklik yapmak isteyen sorgu sayısı dakikada milyonları buluyorsa, belirli kriterler oluşturulması ve işlemlerin belirli iş blokları halinde gerçekleşmesi gerekir.
- **Transaction**, veritabanı işlemlerinde başarı ve hatayı garanti eden operasyonlar listesi olarak tanımlayabiliriz.
- **ACID** (Atomicity, Consistency, Isolation, Durability) (elektrik kesintisi, sistem çökmesi gibi durumlarda geçerliliği garanti etmeye yarayan veritabanı özelliklerinin bir dizi özelliği, detaylıca inceleyeceğiz)
- **ACID** özelliklerini karşılayan veritabanı işlemler dizisi (kod bloğu), **transaction** olarak da adlandırılır.

ACID -Atomicity (Bölünemezlik)

- **Atomicity** transaction işleminin ana özelliği olarak ifade eder.
- Bir transaction bloğu yarım kalamaz.
- Yarım kalan transaction bloğu veri tutarsızlığına neden olur.
- Ya tüm işlemler gerçekleştirilir, ya da transaction başlangıcına geri döner.
- Diğer bir ifade ile transaction'ın gerçekleştirdiği tüm değişiklikler geri alınarak gerçekleşmeden önceki haline döner.

ACID -Consistency (Tutarlılık)

- **Consistency**, bölünemezlik kuralının alt yapısını oluşturduğu bir kuraldır.
- **Transaction** veri tutarlılığı sağlamalıdır.
- Yani bir transaction içerisinde güncelleme işlemi gerçekleştiyse ve ya kalan tüm işlemler de gerçekleşmeli ya da güncelle işlemi de geri alınmalıdır.
- Bu veri tutarlılığı açısından çok önemlidir.

ACID -Isolation (İzolasyon)

- Her transaction veritabanı için bir istek (iş) paketidir.
- Bir iş paketi (transaction) tarafından gerçekleştirilen değişiklikler tamamlanmadan bir başka transaction tarafından **görülememelidir**.
- Her transaction ayrı olarak işlenmelidir.
- Transaction'ın tüm işlemleri gerçekleştikten sonra bir başka transaction tarafından görülebilmelidir.

ACID -Durability (Dayanıklılık)

- Transaction'lar veri üzerinde karmaşık işlemler gerçekleştirebilir.
- Bu işlemlerin bütünü güvence altına almak için transaction hatalara karşı dayanıklı olmalıdır.
- SQL Server'da meydana gelebilecek sistem sorunu, elektrik kesilmesi, işletim sisteminden ya da farklı yazılımlardan kaynaklanabilecek hatalara karşı hazırlıklı ve dayanıklı olmalıdır.

Bir transaction bloğunun işlenişi

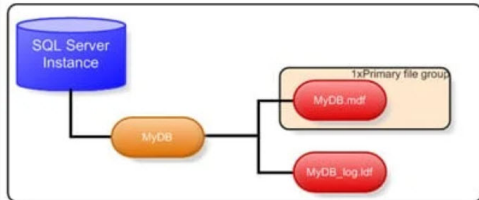
- **Transaction** bloğu başlatılır. Transaction içerisindeki tüm iş parçacıkları bir bütün olarak tamamlanmak ya da bütün olarak iptal edilmek zorundadır. 6 işlem parçası içeren bir transaction bloğu içerisinde 5 işlem gerçekleşse de 1 işlemde hata alınırsa başarılı olan diğer 5 işlem geri alınır ve blok geçersiz olur. Transaction bloğu otomatik ya da kullanıcı tarafından başlatılabilir. Transaction'ı başlatmak için **BEGIN TRANSACTION** ya da kısa olarak **BEGIN TRAN** kullanılabilir
- Transaction bloğu içerisinde yapılan her bir işlem bittiğinde işlemin başarılı olup olmadığına bakılır. İşlem başarısız ise, geri alma (**ROLLBACK**) gerçekleştirilir. İşlem başarılı ise diğer işlemlere devam edilir.
- Transaction, içerisindeki işlemler başarılı olarak tamamlandığında **COMMIT** ile bitirilir. Başarısız olunursa **ROLLBACK** ile geri alma işlemi gerçekleştirilir. **ROLLBACK** kullanıldığında, tüm veriler transaction önceki haline geri döner.

- SQL Server, diğer tüm büyük veritabanı yönetim sistemlerinde olduğu gibi veri işlemlerini birçok farklı katman ve işlem ile gerçekleştirir.
- SQL Server üzerinde **transaction** ile herhangi bir veri değiştirme işleminde değişiklik anında veritabanına yansımaz.
- Üzerinde değişiklik yapılan sayfalar daha önce diskten hafızaya alınmamış ise öncelikle tampon hafıza (**buffer cache**) denilen hafıza bölgesine çağrılır.
- Hafıza bölgesindeki sayfalar üzerinde değişiklikler gerçekleştirilir ve veritabanına hemen yansıtılmaz.
- Hafızaya çağrılarak üzerinde değişiklik yapılan sayfalara kirli sayfa (**dirty page**) denir.

- Kirli sayfalarda tutulan değişiklikler gerekli iş süreçleri tamamlandıktan sonra veritabanına kaydedilir.
- Kirli sayfaların veritabanına kaydedilme işlemine ise arındırma (**flushing**) denir.
- Veri tutarlılığı açısından ve veri kaybını önlemek için yapılan değişiklikler ***.LDF** uzantılı transaction log dosyasına kaydedilir.
- Bu işlem, verinin veritabanına kaydedilmeden önce yapılması önemlidir. Bu log dosyaları disk üzerinde gerçek verinin bulunduğu veri sayfalarına yazılmadan önce, verinin kaybolmasını önlemek amacıyla kullanılır.
- Herhangi bir olası sorun karşısında, tutulan bu loglar, kaybolan verinin geri getirilmesini sağlar.

SQL Server Dosyalar

- SQL Server veritabanı **MDF** (master data file) ve **LDF** (log data file) olmak üzere iki dosyadan oluşur.
- **MDF** – Ana Veritabanı Dosyası anlamına gelir. Sunucunun parçası olan veritabanının tüm ana bilgilerini içerir.
- **LDF** – Bu dosya, ana veri dosyası için işlem günlükleriyle ilgili bilgileri saklar. Yapılan tüm değişikliklerin kaydını tutar. Bu dosya değişiklik tarihi/saati, yapılan değişikliklerin ayrıntıları ve değişiklikleri kimin yaptığıyla ilgili bilgileri içerir.



Transaction İfadeleri

- Transaction yönetimi için kullanılan dört farklı ifade vardır.
- **BEGIN**: bu ifade ile transaction başlatılabilir
- **ROLLBACK**: işlemler geri alınabilir
- **COMMIT**: transaction bitirilebilir
- **SAVE**: kayıt noktaları oluşturulabilir.

Transaction Başlatmak

- **BEGIN TRAN:** Transaction'ın başlangıcını belirtir.
- Bu kısımdan sonraki tüm işlemler **transaction**'ın bir parçasıdır.
- İşlem sırasında oluşabilecek olası sorunlarda geri alma ya da transaction'ın sonlandırılması gerçekleştirilebilir.
- Sade:

```
BEGIN { TRAN | TRANSACTION }  
[ ; ]
```

- Detaylı :

```
BEGIN { TRAN | TRANSACTION }  
    [ { transaction_name | @tran_name_variable }  
      [ WITH MARK [ 'description' ] ]  
    ]  
[ ; ]
```

Transaction Tamamlamak

- **COMMIT TRAN:** Transaction'ın tamamlandığını ve gerçekleştirilen transaction işlemlerinin kalıcı olarak veritabanına yansıtılması için kullanılır.
- Transaction tarafından etkilenen tüm değişiklikler, işlemlerin tamamı gerçekleşmese bile, bu işlemten sonra kalıcı hale gelir.
- **COMMIT** işleminden sonra gerçekleşen değişikliklerin geri alınması için, bu işlemleri geri alacak yeni bir transaction oluşturulmalıdır.
- Syntax:

```
COMMIT [ { TRAN | TRANSACTION } [ transaction_name | @tran_name_variable ] ]  
[ ; ]
```

Transaction'ı Geri Almak

- **ROLLBACK TRAN:** Transaction'ın gerçekleştirdiği tüm işlemleri geri almak için kullanılır.
- Yapılan tüm işlemler transaction'ın başlangıcındaki haline geri döner.
- **ROLLBACK** ile gerçekleştirilen tüm işlemler geriye alınarak transaction sonucunun tutarlılığı garanti edilir.
- **ROLLBACK** işlemi, oluşturduğunuz transaction mimarisine bağlı olarak, kayıt noktalarına (**save points**) geri dönüş için de kullanılabilir
- Syntax:

```
ROLLBACK { TRAN | TRANSACTION }  
    [ transaction_name | @tran_name_variable  
    | savepoint_name | @savepoint_variable ]  
[ ; ]
```

Sabitleme Noktaları

- **ROLLBACK** işlemi transaction'da en başa dönmek için kullanılır.
- Bazen de belirli bir noktaya kadar gerçekleşen işlemlerin geçerli kalması istenebilir. Bu durumda **SAVE TRAN** komutu ile sabitleme noktası oluşturulur.
- Bu işlemlerden sonra gerçekleşecek işlemler için **ROLLBACK**'e ihtiyaç duyulabilir.
- Sabitleme noktaları oluşturulması, transaction içerisinde en başa dönmek yerine, belirlenen bir işlem noktasına dönmek için kullanılır.
- Syntax:

```
SAVE { TRAN | TRANSACTION } { savepoint_name | @savepoint_variable }  
[ ; ]
```


- Veritabanında HesapBilgileri isminde bir tablo oluşturalım.
- HesapID isminde bir sütunumuz olsun ve **auto increment** özelliğinde olsun.
- Adi ve Soyadi özellikleri olsun ve metin değerler alsın.
- Bakiye adına bir sütunumuz olsun ve ondalıklı değerler alabilsin.
- Bu tabloya **transcation** ile veri ekleme yaparak **commit** ve **rollback** komutlarını deneyelim.

sa)

DESKTOP-ANV7JLU\S...bo.HesapBilgileri

Column Name	Data Type	Allow Nulls
HesapId	int	<input type="checkbox"/>
Adi	varchar(250)	<input checked="" type="checkbox"/>
Soyadi	varchar(250)	<input checked="" type="checkbox"/>
Bakiye	float	<input checked="" type="checkbox"/>
		<input type="checkbox"/>

Column Properties

(General)

(Name)	HesapId
Allow Nulls	No
Data Type	int
Default Value or Binding	

Table Designer

Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No

Transaction SQL

a)

```
SQLQuery1.sql - DE...gramLama (sa (54))* X
BEGIN TRAN
    INSERT INTO HesapBilgileri (Adi,Soyadi,Bakiye)
    VALUES ('Murat','Taşyürek',0)

    COMMIT
    --ROLLBACK |

    SELECT * FROM HesapBilgileri
```

121 %

Results Messages

	HesapId	Adi	Soyadi	Bakiye
1	2	Murat	Taşyürek	0

Save Transaction Örnek

- HesapBilgileri tablosunda transaction başlatarak güncelleme işlemi yapalım.
- Transaction başlattıktan sonra tablonun o anki bilgini select ile gösterelim.
- İlk güncelleme işleminde Bakiye değerini 750'ye set edelim.
- Daha sonra SP1 isimli save point (save transaction) noktası oluşturalım.
- Daha sonra güncelleme işleminde Bakiye değerini 1500'ye set edelim.
- Daha sonra SP2 isimli save point (save transaction) noktası oluşturalım.
- Daha sonra güncelleme işleminde Bakiye değerini 3200'ye set edelim.
- Daha sonra SP3 isimli save point (save transaction) noktası oluşturalım.
- Daha sonra ROLLBACK ile SP2 noktasına geri dönelim.

Transaction SQL

The screenshot shows a SQL query editor window with a tab titled "SQLQuery1.sql - DE...gramLama (sa (54))". The query text is as follows:

```
BEGIN TRAN

SELECT * FROM HesapBilgileri

UPDATE HesapBilgileri Set Bakiye=750 WHERE HesapId=2;

SAVE Transaction SP1

UPDATE HesapBilgileri Set Bakiye=1500 WHERE HesapId=2;

SAVE Transaction SP2

UPDATE HesapBilgileri Set Bakiye=3200 WHERE HesapId=2;

SAVE Transaction SP3

SELECT * FROM HesapBilgileri

ROLLBACK Transaction SP2

SELECT * FROM HesapBilgileri
```

Below the query editor, the "Results" tab is active, displaying three tables. The first table shows the state before the transaction, with a balance of 150. The second table shows the state after the first two updates, with a balance of 3200. The third table shows the state after the rollback, with a balance of 1500.

HesapId	Adi	Soyadi	Bakiye	
1	2	Murat	Tagyurek	150

HesapId	Adi	Soyadi	Bakiye	
1	2	Murat	Tagyurek	3200

HesapId	Adi	Soyadi	Bakiye	
1	2	Murat	Tagyurek	1500

Try Catch

- Kodlama yapılırken beklenmeyen durumların karşılaştığında programın hataya düşmemesi için **try-cath** bloğu kullanılır.
- T-SQL'de **try-cath** bloğu aşağıda gösterildiği gibi kullanılır.

```
BEGIN TRY
    { sql_statement | statement_block }
END TRY
BEGIN CATCH
    [ { sql_statement | statement_block } ]
END CATCH
[ ; ]
```

Try Catch ile Transaction

- Transaction işlemlerinde beklenmeyen durumlarda bir bütünlüğün sağlanması için olası kırılmaların yakalanması ve **rollback** yapılması gerektirir.
- Bu kırılma yazdığım kodda olacağı gibi yazdığımı koddan kaynaklı **trigger** vb. çalışan fonksiyonlarda da olabilir.
- Bu nedenle **transaction'ın tr-catch ile birlikte kullanılması** her zaman daha iyidir. T-SQL'de **try-cath transaction** bloğu aşağıda gösterildiği gibi kullanılır.

```
BEGIN TRY
    BEGIN TRANSACTION

    { sql_statement | statement_block }

    COMMIT
END TRY
BEGIN CATCH
    ROLLBACK

    [ { sql_statement | statement_block } ]
END CATCH
```

Try Catch ile Transaction Örneği

- HesapBilgileri tablosunda **try-catch bloğu ile transaction** başlatarak güncelleme işlemi yapalım.
- Transaction başlattıktan sonra tablonun o anki bilgini select ile gösterelim.
- İlk güncelleme işleminde Bakiye değerini 150 artıralım.
- Daha sonraki güncelleme işleminde bakiye değerini '250a' artıralım.
- Bu işlemlerden sonra **COMMIT** yapalım.
- **Catch** bloğunda hata oluşursa ekrana hata nedenini ve hata kodunu yazdıralım ve **ROLLBACK** işlemi yapalım.

Try Catch Transaction Sonuç

SQLQuery1.sql - DE...gramLama (sa (54))

```
BEGIN TRY
    BEGIN TRANSACTION
    SELECT * FROM HesapBilgileri

    UPDATE HesapBilgileri set Bakiye=Bakiye+150 where HesapId=2

    UPDATE HesapBilgileri set Bakiye=Bakiye+'250a' where HesapId=2

    COMMIT
END TRY
BEGIN CATCH
    ROLLBACK
    PRINT 'Hata oluştuğu için rollback yapıldı'
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_MESSAGE() AS ErrorMessage;
END CATCH

SELECT * FROM HesapBilgileri
```

121 %

Results Messages

	HesapId	Adi	Soyadi	Bakiye
1	2	Murat	Taşyürek	0

	ErrorNumber	ErrorMessage
1	8114	Error converting data type varchar to float.

	HesapId	Adi	Soyadi	Bakiye
1	2	Murat	Taşyürek	0