Veritabanı Programlama
Bölüm 11
Transaction

Doç. Dr. Öğr. Üyesi Murat TAŞYÜREK

11 Aralık 2024

Kayseri Üniversitesi, Bilgisayar Mühendisliği Bölümü

Transaction gereksinim

- Veritabanı sunucuları, birçok işlemi doğru, eş zamanlı ve yüksek performansla gerçekleştirebilmek için optimize edilmiştir..
- 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 yönetimi, özellikle finans, sağlık ve büyük ölçekli veri işlemleri gerektiren sistemlerde veri tutarlılığını ve işlem bütünlüğünü sağlamak için kritik öneme sahiptir.**

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 özellik, veri tutarlılığının korunması açısından büyük önem taşır.

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.
- **Bir transaction'ın yaptığı değişiklikler, yalnızca tamamlandıktan sonra diğer transaction'lar tarafından görülebilir.**

ACID -Durability (Dayanıklılık)

- Transaction'lar veri üzerinde karmaşık işlemler gerçekleştirebilir.
- Bu işlemlerin bütününü 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 içinde yapılan her bir işlem tamamlandığında, işlemin başarıyla gerçekleşip gerçekleşmediği kontrol edilir. 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.

Transaction Bloğu

- 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.

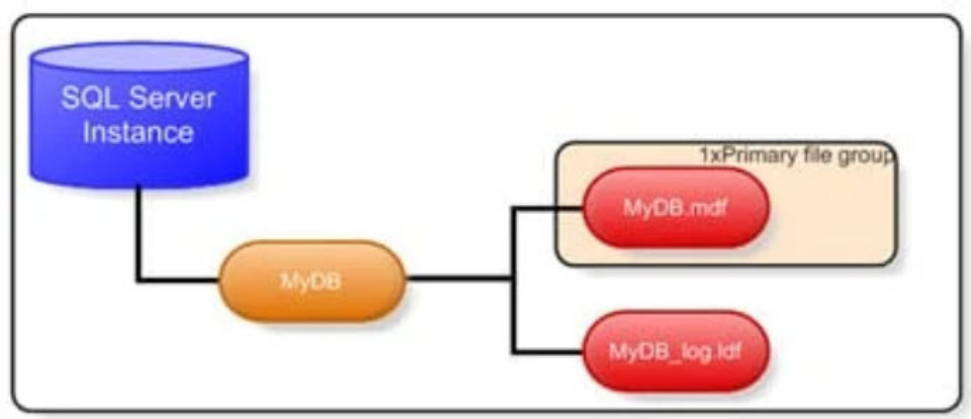
SQL Server LDF

- 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. Log dosyaları, veri kaybını önlemek amacıyla, gerçek veri sayfalarına yazılmadan önce değişiklikleri geçici olarak saklar.
- 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.

SQL Server Dosyalar



Transaction İfadeleri

- Transaction yönetimi için kullanılan dört farklı ifade vardır.
- **BEGIN**: bu ifadele 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 işlemden 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:

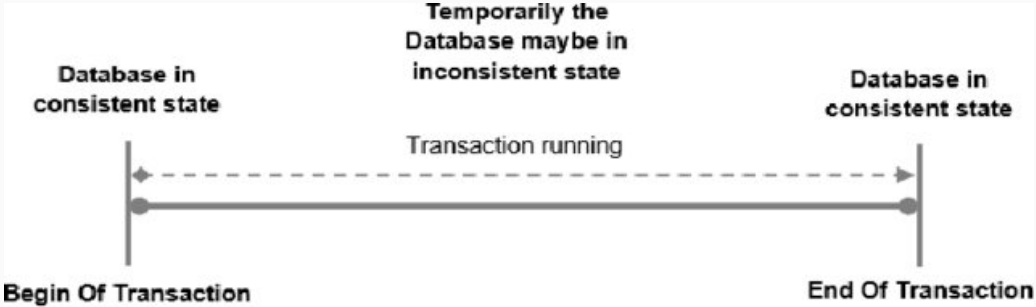
```
ROLLBACK { TRAN | TRANSACTION }
     [ transaction_name | @tran_name_variable
     | savepoint_name | @savepoint_variable ]
[ ; ]
```

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 }
[ ; ]
```

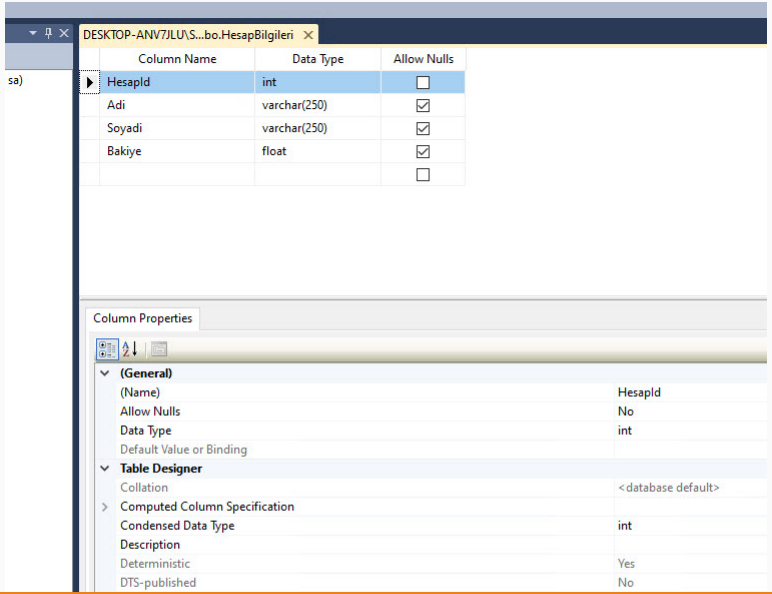
Transaction Bloğu



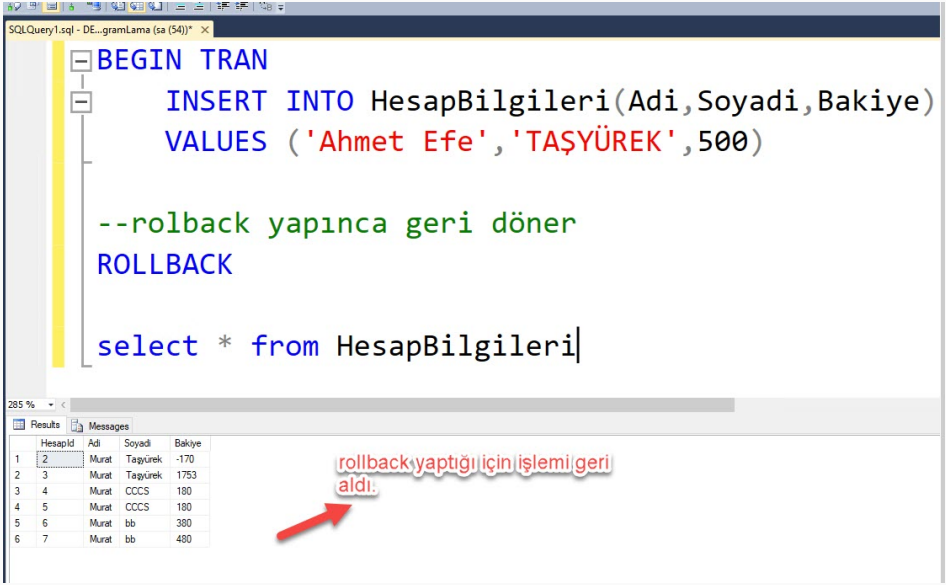
Örnek

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

Tablo


sa)
DESKTOP-ANV7JLU\S...bo.HesapBilgileri
Column Name
Data Type
Allow Nulls
HesapId
int
Adi
varchar(250)
Soyadi
varchar(250)
Bakiye
float
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 -rollback



Transaction SQL -commit

```
BEGIN TRAN
    INSERT INTO HesapBilgileri(Adi,Soyadi,Bakiye)
    VALUES ('Mehmet Yiğit','TAŞYÜREK',400)

--commit yapınca işlem bitmiş olur.
commit

select * from HesapBilgileri
```

85 %

Results | Messages

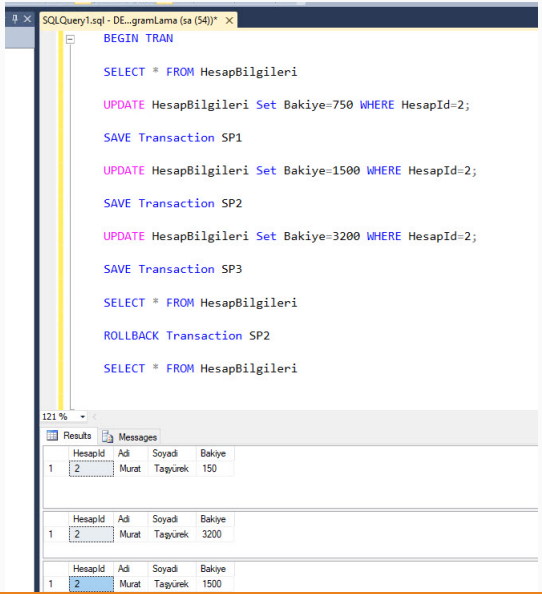
	HesapId	Adi	Soyadi	Bakiye
1	2	Murat	Taşyürek	-170
2	3	Murat	Taşyürek	1753
3	4	Murat	CCCS	180
4	5	Murat	CCCS	180
5	6	Murat	bb	380
6	9	Mehmet Yiğit	TAŞYÜREK	400
7	7	Murat	bb	480

commit olduğu için işlem tamanlandı, kayıt edildi.

Save Transaction Örnek

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

Transaction SQL



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 gerekbilir.
- Bu kırılma yazdığım kodda olacağı gibi yazdığımı koddan kaynaklı **triger** 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 başlatttıktan 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 oluşrsa ekrana hata sebenini 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	Adı	Soyadı	Bakiye
1	2	Murat	Taşyürek	0

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

	HesapId	Adı	Soyadı	Bakiye
1	2	Murat	Taşyürek	0