BİL3002

VERİTABANI TASARIM VE UYGULAMA

12. Hafta --> Transactions

II. Kısım

SQL: Transactions

Transactions (İşlemler)

- **Transaction**'lar, VYS'nin kullanıcılar için *kurtarma* ve *eşzamanlılığı* işlemesini sağlayan bir programlama soyutlamasıdır.
- Uygulamada: **Transaction**'lar kullanıcılar için kritik öneme sahiptir.
  - Veri işleme sistemlerinin sıradan kullanıcıları bile!
- **Temeller: Transaction**'ların nasıl çalıştığına dair temel bilgiler ve bunları kullanmayla ilgili temel problemler

Transactions

- **Transaction**'lar, VYS'nin bir kullanıcı programının soyut görünümüdür
  - Bir dizi okuma ve yazma
  - Birden çok kez yürütülen aynı program, farklı işlemler olarak kabul edilir.
  - VYS, CREATE TABLE ifadelerinde belirtilen IC'lere bağlı olarak bazı Bütünlük Kısıtlamalarını (IC'ler) uygulayacaktır.
  - Bunun ötesinde, VYS verilerin anlamını gerçekten anlamıyor
    - Ör. Bir banka hesabındaki faizin nasıl hesaplandığını anlamıyor

Örnek

T1:	BEGIN	A=A=A+100,	B=B-100 END
T2:	BEGIN	A=1.06*A,	B=1.06*B END

- Sezgisel olarak, ilk işlem B'nin hesabından A'nın hesabına 100 ABD Doları aktarılmasıdır.
- İkincisi, her iki hesabı da % 6 faiz ödemesiyle alacaklandırmaktır.
- VYS'nin bu işlemlere ilişkin görünümü:

T1:	R(A), W(A), R(B), W(B)
T2:	R(A), W(A), R(B), W(B)

Transactions: Tanımlama

- Bir Transaction (TXN), tek bir gerçek-dünya geçişini yansıtan bir veya daha fazla işlem (okuma veya yazma) dizisidir.
- Gerçek dünyada, bir TXN ya tamamen gerçekleşir ya da hiç gerçekleşmez
  - Ör.
    - Banka hesapları arasındaki para transferi
    - Bir grup ürünün satın alınması
    - Bir ders için kayıt olma

```
START TRANSACTION
      UPDATE Product
      SET Price = Price – 1.99
      WHERE pname = 'Gizmo'
COMMIT
```

Transactions: Tanımlama

- SQL'de:
  - Varsayılan: Her sorgu bir işlem (transaction)'dir
- Bir uygulamada ise, birden çok ifade bir işlem olarak birlikte gruplanabilir:

```
START TRANSACTION
      UPDATE Bank SET amount = amount – 100
      WHERE name = 'Bob'
      UPDATE Bank SET amount = amount + 100
      WHERE name = 'Joe'
COMMIT
```

Transactions'lar için motivasyon

- Kullanıcı eylemlerini (okuma (read) ve yazma (write)) işlemlere göre gruplamak iki hedefe yardımcı olur:

1. **Kurtarma ve Dayanıklılık:** VYS verilerini çökmeler, iptaller, sistem kapanmaları, donanım arızaları vb. Karşısında tutarlı ve dayanıklı tutmak.
    - Genellikle «logging» ile uygulanır

2. **Eşzamanlılık:** Anomaly'ler yaratmadan TXN'leri paralelleştirerek daha iyi performans elde etme
    - Genellikle «locking» ile uygulanır

I. Kurtarma ve Dayanıklılık:

- Kullanıcı verilerinin kurtarılması ve dayanıklılığı, güvenilir VYS kullanımı için çok önemlidir
  - VYS'ler çökmeler yaşayabilir (ör. Elektrik kesintileri, vb.)
  - Bireysel TXN'ler iptal edilebilir (ör. Kullanıcı tarafından)

- **Fikir:** TXN'lerin ya tam olarak dayanıklı bir şekilde depolandığından ya da hiç saklanmadığından emin olun; TXN'leri "geri alabilmek" için log'lar tutun

Çökmelere/İptallere karşı koruma

```
Client 1:
     INSERT INTO SmallProduct(name, price)
          SELECT pname, price
          FROM Product
          WHERE price <= 0.99
```

Crash / abort!

```
     DELETE Product
          WHERE price <=0.99
```

Ters giden ne olur?

Çökmelere/İptallere karşı koruma

- Bir kullanıcı bir SQL ifadesini yürüttüğünde otomatik olarak bir işlem başlatılır
- **COMMIT** komutu TXN'yi işler
  - Etkileri nihai hale getirilir ve sonraki işlemlere görünür hale gelir
- **ROLLBACK** komutu TXN'yi iptal eder
  - Etkileri geri alınır

```
Client 1:
     START TRANSACTION
          INSERT INTO SmallProduct(name, price)
               SELECT pname, price
               FROM Product
               WHERE price <= 0.99

          DELETE Product
               WHERE price <=0.99
     COMMIT OR ROLLBACK
```

2. Eşzamanlılık

- İyi bir VYS performansı için kullanıcı programlarının **eşzamanlı** yürütülmesi önemlidir
  - Disk erişimi sık ve yavaş olabilir
  - Kullanıcılar, TXN'leri tek başlarına ve tutarlılık korunacak şekilde yürütebilmelidir.
  - Eşzamanlı yürütmeye izin verirken işlem izolasyonunu sağlamak zordur ancak performans nedenleriyle gereklidir
    - İşlemci ve disk kullanımında artış, daha fazla işlem hacmi sağlar (belirli bir sürede tamamlanan ortalama işlem sayısı)
      - bir işlem CPU'yu kullanırken diğeri diskten okuma veya diske yazma durumunda olabilir
    - İşlemler için azaltılmış ortalama yanıt süresi
      - Kısa işlemlerin uzun olanların arkasında beklemesi gerekmez
- ***Temel Fikir!*** CPU'ların canlı kalmasını sağlamak için VYS'nin aynı anda birkaç kullanıcı TXN'sini çalıştırmasını sağlayın ...

Birden çok kullanıcı: tek sorgular

```
Client 1: UPDATE Product
          SET Price = Price – 1.99
          WHERE pname = 'Gizmo'

Client 2: UPDATE Product
          SET Price = Price * 0.5
          WHERE pname = 'Gizmo'
```

İki yönetici aynı anda ürün indirimi yapmaya çalışıyor. Ne yanlış gidebilir?

Birden çok kullanıcı: tek sorgular

```
Client 1: START TRANSACTION
              UPDATE Product
              SET Price = Price – 1.99
              WHERE pname = 'Gizmo'
          COMMIT

Client 2: START TRANSACTION
              UPDATE Product
              SET Price = Price * 0.5
              WHERE pname = 'Gizmo'
          COMMIT
```

Transactions: Tanımlama

- Şema işlemleri (ör. CREATE TABLE) geçerli işlemi örtük olarak taahhüt eder
  - Çünkü; Bir şema işlemini geri almak genellikle zordur
- Birçok VYS, her bir ifadeyi otomatik olarak işleyen bir **AUTOCOMMIT** özelliğini destekler
  - API aracılığıyla açıp kapatılabilir

Transaction Özellikleri: ACID

- Bir TXN'deki DB işlemleri aşağıdaki özellikleri (ACID) izlemelidir. VYS'in eşzamanlılık kontrolü ve kurtarma yöntemleri tarafından uygulanmalıdır.
- **A**tomic:
  - Durum, TXN'nin tüm etkilerini gösterir veya hiçbirini göstermez
  - Bir TXN'nin işlemleri "yarım bitmiş" şeklinde ASLA bırakılmaz
- **C**onsistent:
  - TXN, bütünlüğün korunduğu bir durumdan bütünlüğün geçerli olduğu başka bir duruma geçer
- **I**solated
  - TXN'lerin etkisi, birbiri ardına çalışan TXN'ler ile aynıdır
- **D**urable:
  - Bir TXN gerçekleştirildikten sonra etkileri veritabanında kalır
  - Bir TXN işleme koyulduktan sonra VYS çökerse, VYS geri geldiğinde TXN'nin tüm etkileri veritabanında kalmalıdır.

ACID: Atomicity

- TXN'nin faaliyetleri atomiktir: *ya hep ya hiç*
  - Sezgisel olarak: Gerçek-dünyada, bir işlem ya tamamen gerçekleşecek ya da hiç gerçekleşmeyecek bir şeydir.
- Bir TXN için iki olası sonuç
  - (**Commit**) Taahhüt eder: tüm değişiklikler yapılır
  - (**Abort**) İptal edilir: hiçbir değişiklik yapılmaz

ACID: Atomicity

- Bir işlemin kısmi etkileri şu durumlarda geri alınmalıdır:
  1. Kullanıcı işlemi açıkça **ROLLBACK** kullanarak iptal eder
     Örneğin, uygulama son adımda kullanıcıdan onay ister ve yanıta bağlı olarak COMMIT veya ROLLBACK yayınlar.
  2. VYS, bir işlem tamamlanmadan önce çöker
  3. Herhangi bir kısıtlama ihlal edilir

  Bazı sistemler yalnızca bu ifadeyi geri alır ve işlemin devam etmesine izin verir; diğerleri tüm işlemi geri alır

Atomicity nasıl sağlanır:

- VYS tarafından
- Logging ile («geri alma işlemi»)

ACID: Consistency

- Tablolar her zaman kullanıcı tanımlı bütünlük kısıtlamalarına uymalıdır
  - Örnek:
    - Hesap numarası benzersizdir
    - Stok miktarı negatif olamaz
    - Borç ve kredi toplamı 0'dır
    - Tasarruf hesabından çekilip başka hesaba para transfer ederseniz, toplam tutar hala aynı kalır

Consistency nasıl sağlanır:

- Bu özelliğin her işlemde kullanıcının sorumluluğunda olmasını sağlamak
  - Programcı, bir TXN'nin tutarlı bir durumu tutarlı bir duruma almasını sağlar
  - Veritabanının tutarlılığı, veritabanında belirtilen kısıtlamalar ve tetikleyiciler ve / veya işlemlerin kendisinde garanti edilir.
  - Tutarsızlık ortaya çıktığında, ifadeyi veya işlemi iptal edin veya (ertelenmiş kısıtlama kontrolü veya uygulama tarafından zorlanan kısıtlamalarla) işlem içindeki tutarsızlığı düzeltin

ACID: Isolation

- Bir işlem diğer işlemlerle eşzamanlı olarak yürütülür
- Isolation: Sanki her bir işlem diğerlerinden ayrı olarak yürütülür.
  - Bir işlem, taahhüt edilene kadar güncellemelerini diğer işlemlere görünür hale getirmemelidir
  - Örneğin. Çalıştırma sırasında diğer işlemlerden gelen değişiklikleri gözlemleyememelidir

ACID: Isolation

- Performans için VYS, işlemleri serileştirilebilir bir çizelge (serializable scheduling) kullanarak yürütür
  - Bu çizelgede, farklı işlemlerden operasyonlar eşzamanlı olarak araya eklenebilir ve yürütülebilir.
  - Ancak çizelgenin, seri programla aynı efektleri üretmesi garanti edilir.

Isolation nasıl sağlanır:

- Kilitleme, çok sürümlü eşzamanlılık kontrolü vb.

ACID: Durability

- VYS, kullanıcıyı bir işlemin başarıyla tamamlandığı konusunda bilgilendirdikten sonra, TXN'nin etkileri TXN'den sonra var olmaya ("kalıcı") devam etmelidir.
  - Ve tüm program sona erdikten sonra
  - Ve elektrik kesintileri, çökmeler vb. Olsa bile.
  - Ve benzeri...
- Yani, diske yazılmalıdır.

Örn. Para Transferi

- A hesabından B hesabına 50$ aktarma işlemi:
  1. read(A)
  2. A := A − 50
  3. write(A)
  4. read(B)
  5. B := B + 50
  6. write(B)
- **Atomicity (Atomik çalışma) gereksinimi:**
  - İşlem 3. adımdan sonra ve 6. adımdan önce başarısız olursa, tutarsız bir veritabanı durumuna yol açacak şekilde para "kaybedilir«
    - Başarısızlık yazılım veya donanımdan kaynaklanabilir.
  - Sistem, kısmen yürütülen bir işlemin güncellemelerinin veritabanına yansıtılmamasını sağlamalıdır.
- **Durability (Kalıcılık) gereksinimi** — kullanıcıya işlemin tamamlandığı bildirildiğinde (yani, 50$'ın transferi gerçekleştiğinde), işlem tarafından veritabanında yapılan güncellemeler, yazılım veya donanım arızaları olsa bile devam etmelidir.

Örn. Para Transferi

- Yukarıdaki örnekte **Consistency** (tutarlılık) gereksinimi:
  - A ve B'nin toplamı, işlemin yürütülmesiyle değişmez
- Genel olarak, **Consistency** gereksinimleri şunları içerir:
  - Birincil anahtarlar ve yabancı anahtarlar gibi açıkça belirtilen bütünlük kısıtlamaları
  - Örtük bütünlük kısıtlamaları
    - örneğin, tüm hesapların bakiyelerinin toplamı eksi kredi tutarlarının toplamı eldeki nakit değerine eşit olmalıdır
- Bir transaction tutarlı (consistent) bir veritabanı görmelidir.
- İşlem yürütme sırasında veritabanı geçici olarak tutarsız (inconsistent) olabilir.
- İşlem başarıyla tamamlandığında, veritabanı tutarlı (consistent) olmalıdır
- Hatalı işlem mantığı tutarsızlığa (inconsistency) yol açabilir

Örn. Para Transferi

- **Isolation (Yalıtım)** gereksinimi — 3. ve 6. adımlar arasında başka bir T2 işleminin kısmen güncellenmiş veritabanına erişmesine izin verilirse, tutarsız bir veritabanı görecektir (A + B toplamı olması gerekenden daha az olacaktır).

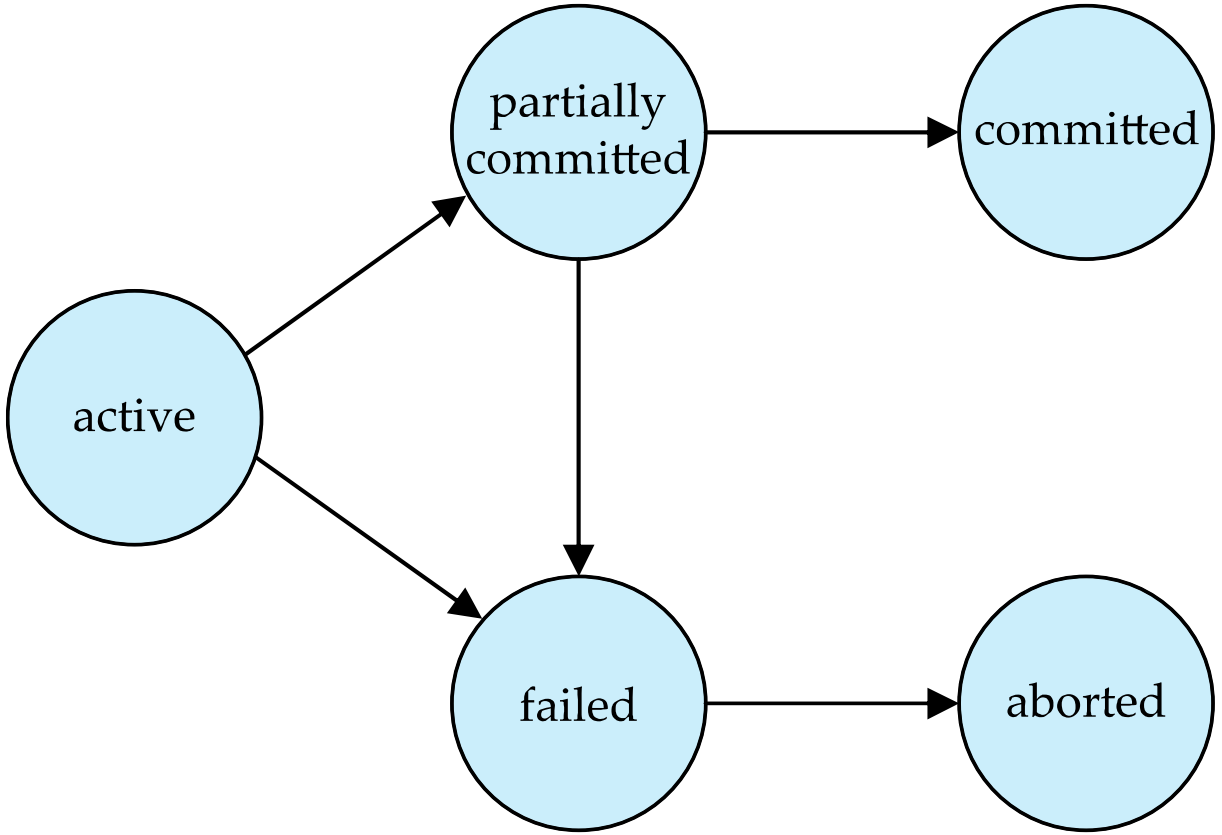
T1	T2
1. read(A)	
2. A := A − 50	
3. write(A)	
	read(A), read(B), print(A+B)
4. read(B)	
5. B := B + 50	
6. write(B)	

- **Isolation** önemsiz bir şekilde işlemleri seri olarak çalıştırarak sağlanabilir. Yani birbiri ardına.

Transaction State (Transaction Durumu)

- Active (Aktif) – başlangıç durumu; işlem yürütülürken bu durumda kalır.
- Partially committed (Kısmen işlenmiş) – son ifade yürütüldükten sonra.
- Failed (Başarısız) – normal yürütmenin artık devam edemeyeceği keşfedildikten sonra.
- Aborted (İptal) – işlem geri alındıktan ve veritabanı işlem başlamadan önceki durumuna geri yüklendikten sonra. İptal edildikten sonra iki seçenek:
  - Transaction'u yeniden başlat
    - Yalnızca dahili mantıksal hata yoksa yapılabilir
  - Transaction'u sonlandır
- Committed (İşlenmiş) - başarıyla tamamlandıktan sonra.

Transaction State (dvm.)



Eşzamanlı (Concurrent) Çalıştırmalar

- Sistemde birden fazla transaction birlikte çalışabilir.
- Avantajları:
  - artırılmış işlemci ve disk kullanımı, daha iyi transaction verimliliği (throughput)
    - Örn. Bir transaction CPU kullanırken, başka biri disk okuma yazma yapabilir
  - azaltılmış ortalama transaction yanıt süresi: kısa transaction'ların uzun olanları beklemesine gerek yok.
- Eşzamanlılık kontrol şemaları (Concurrency control schemes) - Yalıtımı sağlama mekanizmaları: veritabanının tutarlılığını bozmalarını önlemek için, birlikte çalışan (concurrent) transaction'lar arasındaki etkileşimin denetlenmesi

Örnek

- İki transaction düşünün:
  - T1: BEGIN A=A+100, B=B-100 END
  - T2: BEGIN A=1.06*A, B=1.06*B END
- Anlaşıldığı gibi, ilk transaction A hesabından B hesabına 100 dolar aktarıyor. İkincisi de, her iki hesaba %6 ekliyor.
- İkisi birlikte başlatıldığında, T1'in T2'den önce çalışacağının, veya tam tersinin, bir garantisi yok. Ancak görülecek net etki, bu iki transaction'ın sırayla çalıştırılmasına denk olmalıdır.

Schedules (Zamanlamalar)

- (Schedule) Zamanlama - Birlikte çalışan transaction'ların işleyeceği komutların zamansal sıralamasını belirleyen bir dizi komut
  - Bir küme transaction için belirlenen bir schedule, o transaction'lardaki tüm komutları (instruction) bulundurmalıdır
  - Herbir transaction'da bulunan komutların sırası korunmalıdır.
- Çalışmasını başarıyla tamamlayan bir transaction, son ifade (statement) olarak instruction'ları işle (commit instructions)'yi bulundurur
  - ön-tanımlı olarak (by default) transaction'ın son adımda commit instruction yapacağı varsayılır.
- Çalışmasını başarıyla tamamlayamayan bir transaction, son ifade olarak instruction'ları iptal et (abort instructions)'i bulundurur

Çizelge 1 (Schedule 1)

- T1'in A'dan B'ye 50$ transfer etmesine ve T2'nin A'dan B'ye bakiyenin %10'unu transfer ettiğini düşünelim. T1'in ardından T2'nin geldiği bir seri çizelge (serial schedule):

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
	$B := B + temp$
	write (B)
	commit

Çizelge 2 (Schedule 2)

- T2'yi T1'in takip ettiği bir seri çizelge (serial schedule):

T_1	T_2
	read (A) $temp := A * 0.1$ $A := A - temp$ write (A) read (B) $B := B + temp$ write (B) commit
read (A) $A := A - 50$ write (A) read (B) $B := B + 50$ write (B) commit	

Çizelge 3 (Schedule 3)

- T1 ve T2 önceden tanımlanan işlemler olsun. Aşağıdaki çizelge bir seri çizelge (serial schedule) değildir, ancak Çizelge 1'e eşdeğerdir.

T_1	T_2
read (A)	
$A := A - 50$	
write (A)	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
read (B)	
$B := B + 50$	
write (B)	
commit	
	read (B)
	$B := B + temp$
	write (B)
	commit

- Çizelge 1, 2 ve 3'te A + B toplamı korunur.

Çizelge 4 (Schedule 4)

- Aşağıdaki eşzamanlı çizelge (concurrent schedule), (A + B) değerini korumaz.

T_1	T_2
read (A)	
$A := A - 50$	
	read (A)
	$temp := A * 0.1$
	$A := A - temp$
	write (A)
	read (B)
write (A)	
read (B)	
$B := B + 50$	
write (B)	
commit	
	$B := B + temp$
	write (B)
	commit

Conflicting Instructions (Çakışan Komutlar)

- Ti ve Tj transaction'larının li ve lj instruction'ları, ancak li ve lj tarafından erişilen bir Q öğesi varsa, ve bu komutlardan en az birisi write ise, çakışır (conflict).

1. li= read(Q), lj= read(Q). li ve lj çakışmaz.
2. li= read(Q), lj= write(Q). çakışır
3. li= write(Q), lj= read(Q). çakışır
4. li= write(Q), lj= write(Q). çakışır

Serileştirilebilirlik - Serializability

- Temel Varsayım – Her işlem veritabanı tutarlılığını korur.
- Böylece, bir dizi işlemin seri olarak yürütülmesi, veritabanı tutarlılığını korur.
- Bir (muhtemelen eşzamanlı) çizelge, seri çizelgeye eşdeğer ise seri hale getirilebilir. Çizelge denkliğinin farklı biçimleri aşağıdaki kavramların ortaya çıkmasına neden olur:

1. Conflict serializability
2. View serializability

İşlemlerin(Transaction) basitleştirilmiş görünümü

- Read ve write talimatları dışındaki işlemleri göz ardı ederiz
- İşlemlerin, read ve write işlemleri arasında yerel arabelleklerdeki veriler üzerinde keyfi hesaplamalar yapabileceğini varsayıyoruz.
- Basitleştirilmiş programlarımız yalnızca okuma ve yazma talimatlarından oluşur.

Conflict Serializability

- Eğer bir S çizelgesi, bir dizi çelişkili olmayan talimat değiş tokuşu ile bir S' programına dönüştürülebilirse, S ve S'nin conflict equivalent olduğunu söyleriz.
- Bir seri çizelge conflict equivalent ise, bu çizelgeye S'nin çakışma seri hale getirilebilir (Conflict Serializability) olduğunu söylüyoruz.

Conflict Serializability

- Çizelge 3, Çizelge 6'ya dönüştürülebilir; bu, T2'nin T1'i takip ettiği seri bir çizelgedir ve birbiriyle çelişmeyen talimatların değiş tokuş edilmesiyle gerçekleşir. Bu nedenle Çizelge 3 çakışma seri hale getirilebilir.

T_1	T_2
read (A)	
write (A)	
	read (A)
	write (A)
read (B)	
write (B)	
	read (B)
	write (B)

Schedule 3

T_1	T_2
read (A)	
write (A)	
read (B)	
write (B)	
	read (A)
	write (A)
	read (B)
	write (B)

Schedule 6

Conflict Serializability

- Conflict Serializability olmayan bir çizelge örneği:

T_3	T_4
read (Q)	
	write (Q)
write (Q)	

- < T3, T4 > seri çizelgesini veya < T4, T3 > seri çizelgesini elde etmek için yukarıdaki programdaki talimatları değiştiremiyoruz.

View Serializability

- S ve S' aynı işlem (transaction) kümesine sahip iki çizelge olsun. S ve S', her Q veri öğesi için aşağıdaki üç koşulun karşılanması durumunda View Serializability'dir:

1. S çizelgesinde Ti işlemi Q'nun başlangıç değerini okursa, S' programında da Ti işlemi Q'nun başlangıç değerini okumalıdır.
2. S çizelgesinde Ti işlemi read(Q) gerçekleştirirse ve bu değer Tj işlemi (varsa) tarafından üretilmişse, S' programında Ti işlemi de Tj'nin write(Q) işlemi tarafından üretilen Q değerini okumalıdır.
3. S çizelgesinde son read(Q) işlemini gerçekleştiren işlem (varsa), S' programında son read(Q) işlemini de gerçekleştirmelidir.

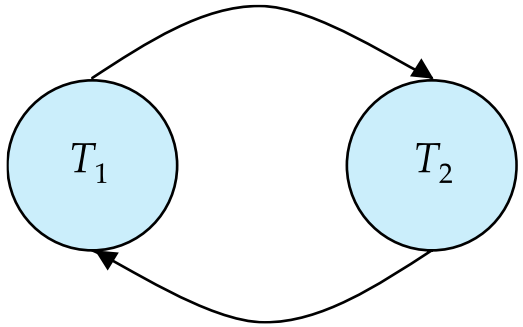
View Serializability

- Bir S çizelgesi, bir View Schedule'e eşdeğer bu çizelge seri hale getirilebilir (view serializable).
- Her conflict serializable çizelge aynı zamanda view serializable.
- Aşağıda, view serializable ancak conflict serializable olmayan bir program bulunmaktadır.

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

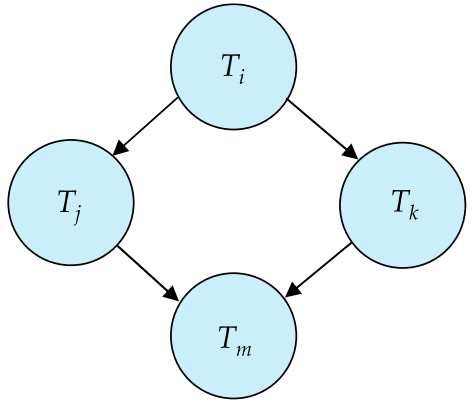
Serileştirilebilirlik (Serializability) Testi

- Bir dizi T1, T2, ..., Tn transaction çizelgesini düşünün.
- Precedence graph (Öncelik graph'ı) - köşelerin transaction (isimler) olduğu doğrudan bir graph.
- İki işlem çakışıyorsa Ti'den Tj'ye bir yay çizeriz ve Ti, conflict'in daha önce ortaya çıktığı veri öğesine erişir.
- Yayı, erişilen öğeye göre etiketleyebiliriz.
- Precedence graph örneği:

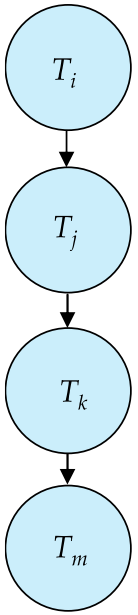


Conflict Serializability Testi

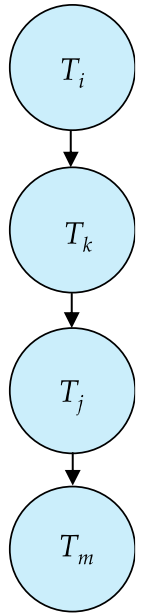
- Bir çizelge, ancak ve ancak predence graph'ı döngüsel değilse conflict serializable'dir.
- Predence graph'ı döngüsel değilse, serileştirilebilirlik sırası, grafın topolojik sıralamasıyla elde edilebilir.
  - Bu, grafın kısmi düzeniyle tutarlı doğrusal bir düzendir.



(a)



(b)



(c)

Sorular

Aşağıdaki çizelgelerin (S) conflict serializable olup olmadığını belirleyiniz:

1. S: $R_1(A), R_2(A), R_1(B), R_2(B), R_3(B), W_1(A), w_2(B)$
2.

T1	T2	T3	T4
	R(X)		
		W(X)	
		Commit	
W(X)			
Commit			
	W(Y)		
	R(Z)		
	Commit		
			R(X)
			R(Y)
			Commit

Sorular

3.

T1	T2	T3	T4
			R(A)
	R(A)		
		R(A)	
W(B)			
	W(A)		
		R(B)	
	W(B)		

4.

T1	T2
R(A)	
A = A-10	
	R(A)
	Temp = 0.2 x A
	W(A)
	R(B)
W(A)	
R(B)	
B = B+10	
W(B)	
	B = B+Temp
	W(B)

Örnek

```
Buket'den vahdete para gönderilecek
BEGIN TRY
      UPDATE      dbo.Hesap      SET      Bakiye-=100      WHERE
TCKimlikNo='23456789101'
      RAISERROR('Elektrikler Kesildi',16,2)
      UPDATE      dbo.Hesap      SET      Bakiye+=100      WHERE
TCKimlikNo='12345678910'
END TRY
BEGIN CATCH
      PRINT 'Beklenmedik bir hata olustu'
END CATCH
```

Eksiltme işlemi gerçekleşti fakat ekleme işlemi tamamlanmadı. Ortada bir para kaybı söz konusu. Nerede bu para?

"BEGIN TRAN" ile "transaction" işlemini başlatıyoruz. ("BEGIN TRANSACTION" da yazabilirdik.)

Ard arda, tek bir işlemmiş gibi çalışmasını istediğim kod bloğumu yazdıktan sonra, "transaction" işlemini "COMMIT TRAN" (işlemi doğrulama anlamı var) ile sonlandırıyorum.

"COMMIT TRAN" komutunu görene kadar, işlemlerimde hata olsa da olmasa da, işlem sonuçlarım tabloya yansımayacaktır.

Daha sonra "try" bloğumu kapatıyorum.

Bir hata ile karşılaşılması sonucunda "Catch" bloğu çalışacağından dolayı, "Catch" bloğunda da "ROLLBACK TRAN" komutu ile yapılan işlemleri geri alınmasını sağlıyoruz.

```
CREATE TABLE Hesap(
id integer identity(1,1),
TCKimlikNo varchar(11),
Ad varchar(30),
Hesap money)
```

```
BEGIN TRY
   BEGIN TRAN
      UPDATE      dbo.Hesap      SET      Bakiye-=100      WHERE
TCKimlikNo='23456789101'
      RAISERROR('Elektrikler Kesildi',16,2)
      UPDATE      dbo.Hesap      SET      Bakiye+=100      WHERE
TCKimlikNo='12345678910'
   COMMIT TRAN
END TRY
BEGIN CATCH
   PRINT 'Beklenmedik bir hata olustu'
   ROLLBACK TRAN
END CATCH
```

Sorular....