Chapter 13: Transactions & Concurrency

- 13.1 Introduction (Giriş)
- 13.2 **Transactions (Transaction'lar)**
- 13.3 Concurrency control (Birlikte çalışma denetimi)
- 13.4 Locks (Kilitler)
- 13.5 Summary (Özet)

Learning objectives (Öğrenme Hedefleri)

- Transaction uygulamalarının ve sunucular tarafından yönetilen paylaşılan nesnelerin concurrency denetiminin tartışılması.
- **Transaction** kavramının tanıtılması: paylaşılan bir veritabanına (db) erişen bir programın çalışması
- Concurrency control tanıtımı: paralel çalışarak ve paylaşılan veriye erişerek birbirleriyle çakışabilecek proseslerin yaptıkları işlerin koordinasyonu

Introduction (Giriş)

- Bir (server) veritabanı, adlandırılmış veri öğelerinden oluşur
- Her veri öğesinin (data item) bir değeri (value) bulunur
- Veri öğelerinin herhangi bir andaki değerleri, db'in durumu (state) olur
- Bir server uygulaması, db'e Read ve Write database işlemleriyle erişir
- Read(x) x veri öğesinde tutulan değeri okur
- Write(x, val) x'in değerini val yapar
- DB sistemi her işlemi atomik olarak yapar. Yani işlemleri sırayla yapıyor gibi davranır
- DB sistemi şu transaction işlemlerini destekler: Start (Başlat), Commit (İşle), Abort (İptal)

*

Transactions

- Suppose you are booking a flight from Sundsvall to Florida.
- You go to the travel agents they tell you that you must.
  - Fly Sundsvall to Stockholm
  - Stockholm to Chicago
  - Chicago to Florida.
- You say OK, book the flights.
- they start booking the flights one by one
- discover on the last flight that it is full. The next free flight from Chicago to Florida is the next day. So you have to wait a whole day at Chicago.

Transactions

- The situation on the previous is not optimal.
- You want to either book all the flights or book none of them
- You don’t want a half completed booking.
- Especially if the half completed booking commits you to something that you don’t want (spend a night in Chicago)
- A transaction is a sequence of operations with an **all or nothing** behavior.
- The problems comes is that other people might be booking flights at the same time and hence some flights might get full while you think they are empty.

Transaction Concept (Transaction Kavramı)

- **Transaction** çeşitli veri öğelerine erişen veya değiştiren, program çalışma *birim*idir.
- Örn. $50'ı A hesabından B'ye aktarmak için yapılan transaction:

  1.**read**(*A*)

  2.*A*:= *A* –50

  3.**write**(*A*)

  4.**read**(*B*)

  5.*B*:= *B* + 50

  6.**write**(*B)*
- İki temel mesele: 1) Başarısızlık çeşitleri: donanımsal hatalar ve sistem çökmeleri, 2) birden fazla transaction'ın birlikte çalışması

Properties of Transactions (Transaction'ların Özellikleri)

Atomicity requirement (Atomik çalışma gereksinimi)

- 3. adımdan sonra ve 6. adımdan önce transaction başarısız olursa, para "kaybı" olur ve veritabanı tutarsız duruma (state) düşer
- Başarısızlık sebebi yazılım da olabilir, donanım da
- Sistem, kısmen çalıştırılmış bir transaction'ın yaptığı değişikliklerin veritabanına yansıtılmayacağını garantilemelidir

Durability requirement (Kalıcılık gereksinimi)

- Transaction'ın (yani $50'ın aktarımının) tamamlandığı kullanıcıya bildirildikten sonra, yazılım veya donanım hataları olsa bile, transaction'ın veritabanında yaptığı değişiklikler kalıcı olmalıdır.

*

Properties of Transactions (Transaction'ların Özellikleri)

Bu örnekte **tutarlılık gereksinimi**:

- Transaction'dan sonra A+B sabit kalır

Genel olarak tutarlılık gereksiniminde

- Bütünlük kısıtlamaları açık olarak belirtilmiştir

Bir transaction tutarlı bir veritabanı görmelidir. Transaction çalışma esnasında veritabanı geçici olarak tutarsız olabilir. Transaction başarıyla tamamlandığında, database tutarlı olmalıdır.

Assumption? Logic of T... (Varsayım? T mantığı...)

*

Properties of Transactions (Transaction'ların Özellikleri)

Isolation (Yalıtım): 3. ve 6. adımlar arasında, başka bir transaction T2'nin kısmen güncellenmiş veritabanına erişimine izin verilirse, tutarsız bir veritabanıyla karşılaşır (*A* + *B* olması gereken değerden küçük olur).

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*	

- Transaction'lar **seri (serially)** çalıştırılarak doğruluk sağlanabilir**:** birinci bittikten sonra ikinci.
- Ancak, daha sonrada göreceğimiz gibi, birden fazla transaction'ın birlikte çalıştırılmasının belirgin faydaları bulunmaktadır.

ACID Properties (ACID Özellikleri)

Veri bütünlüğünü korumak için, veritabanı sistemi şunları sağlamalıdır:

- **Atomicity (Atomiklik).** Ya transaction etkisi tamamıyla veritabanına yansıtılır, ya da hiç etkilemez. a transaction must be all or nothing;
- **Consistency (Tutarlılık).** Transaction'ın yalıtık olarak çalıştırılması veritabanının tutarlılığını korur. a transaction takes the db from one consistent state to another consistent state
- **Isolation (Yalıtım).** Birden fazla transaction birlikte uygulansa da, her bir transaction birlikte çalıştığı diğer transaction'lardan habersiz olmalıdır. Ara transaction sonuçları, birlikte çalışan diğer transaction'lardan saklanmalıdır. Her bir transaction çifti T_i ve T_j için, T_i'ye göre ya T_j, T_i başlamadan bitmiştir, ya da bittikten sonra başlamıştır. transactions should be run as if they are the only transaction running
- **Durability(Kalıcılık).** Transaction başarıyla tamamlandıktan sonra, sistem hataları olsa bile, veritabanında yapılan değişiklikler kalıcı olur. if the server crashes after I have booked my flight, the flight is still booked.

Problems with Transactions - Concurrency Control

- The problem with transaction is that we want to do them **efficiently**.
- An **inefficient** way of doing things would be to stop access to the server by anybody else while when a transaction has started and only grant access when this person has finished.
- Instead we allow many transactions to go on at the same time and try to stop bad things happening when we do things at the same time.

Problems with Transactions

- Two things can go wrong when we merge transactions.
  - The Lost Update Problem
  - Inconsistent retrievals
- To illustrate what is going on we look at a bank account example.

The problem

- Consider two transaction *T* and *U* where *A* = $100, *B* = $200 and *C* = $300.

Transaction T:

- balance = b.getbalance();
- b.setBalance(balance*1.1);
- a.withdraw(balance/10);

Transaction U:

- balance = b.getBalance();
- b.setBalance(balance*1.1);
- c.withdraw(balance/10);

- Executing *T* and *U* should result in *b* being increased by 10% and 10% again so *b* should have the value $242 dollars afterwards. we lost one of the updates to *b*. The sum will be the wrong value.

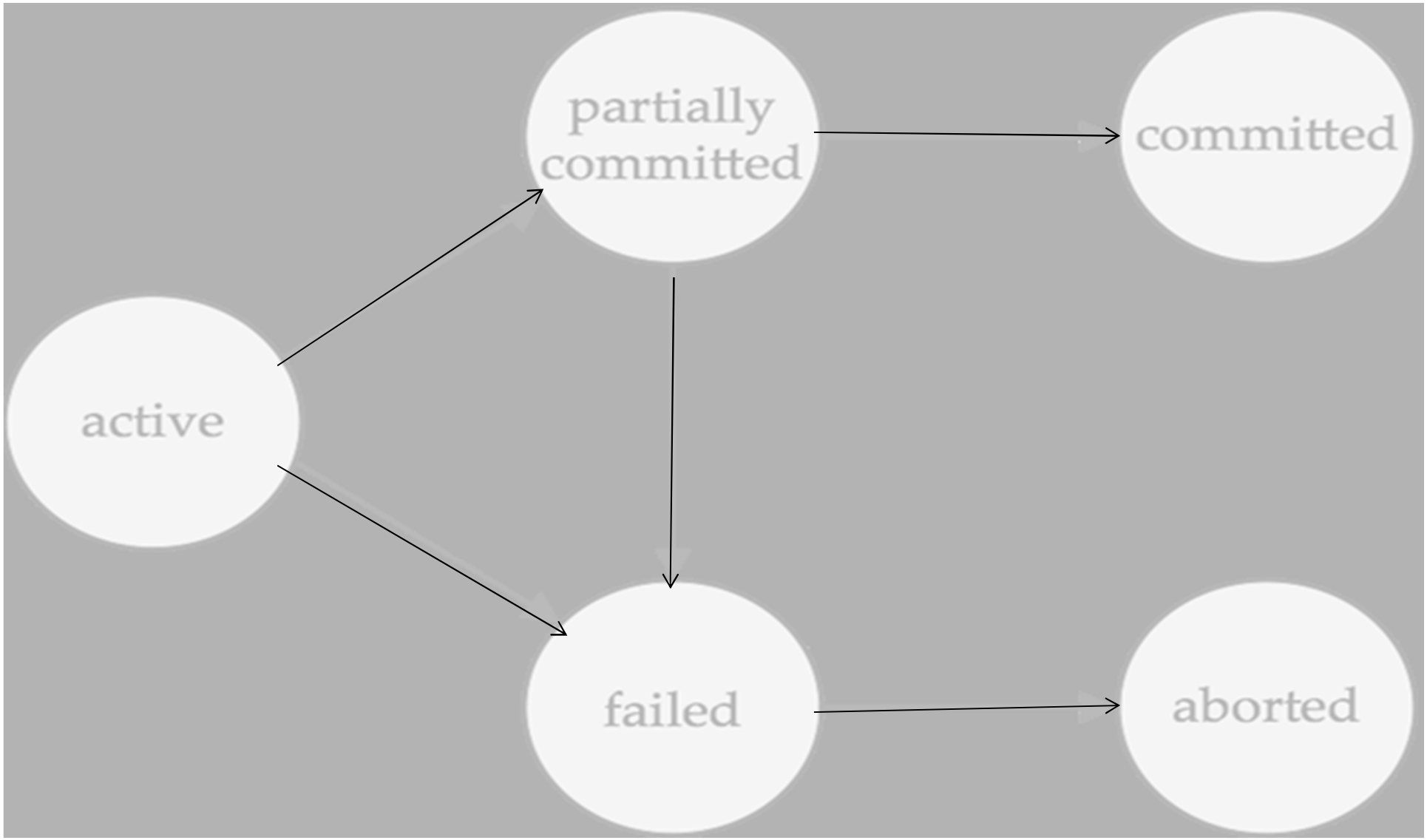
Transaction State (Transaction Durumu)

- **Active (Aktif)** – başlangıç durumu; transaction çalışırken bu durumda kalır
- **Partially committed (Kısmen işlenmiş)** – final statement (son ifade) çalıştırıldıktan sonra.
- **Failed (Başarısız) –** Normal çalışmanın devam edemeyeceği anlaşıldıktan sonra.
- **Aborted (İptal)** – transaction geri çevrildikten ve veritabanı transaction başlangıcındaki durumuna döndürüldükten sonra. İptalden sonra iki seçenek vardır:

  dahili mantık hatası yoksa transaction yeniden çalıştırılır (restart)

  transaction sonlandırılır (kill)
- **Committed (İşlenmiş)** –Başarıyla tamamlandıktan sonra.

Transaction State (Transaction Durumu)



Concurrent Executions (Birlikte Uygulama)

- 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.
- **Concurrency control schemes** – Yalıtımı sağlama mekanizmaları: veritabanının tutarlılığını bozmalarını önlemek için, birlikte çalışan (concurrent) transaction'lar arasındaki etkileşimin denetlenmesi

Example (Örnek)

- İki *xact* düşünün (*xact=transaction*):
  - 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 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** – 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

Schedule Examples:

- Girişik bir *schedule* düşünün:

  T1: A=A+100, B=B-100

  T2: A=1.06*A, B=1.06*B

- Bunda sorun yok. Peki ya bu:

  T1: A=A+100, B=B-100

  T2: A=1.06*A, B=1.06*B

- DBMS'in ikinci schedule'ı görüşü:
  - T1: R(A), W(A), R(B), W(B)
  - T2: R(A), W(A), R(B), W(B)

Serializability (Serileştirilebilme)

- **Basic Assumption (Temel Kabul)** – Her bir transaction veritabanı tutarlılığını korur.
- Böylece bir küme transaction'ın seri çalıştırılması veritabanı tutarlılığını korur.
- (concurrent) bir schedule, seri bir schedule'a denk ise serileştirilebilirdir.

Conflicting Instructions (Çakışan Komutlar)

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

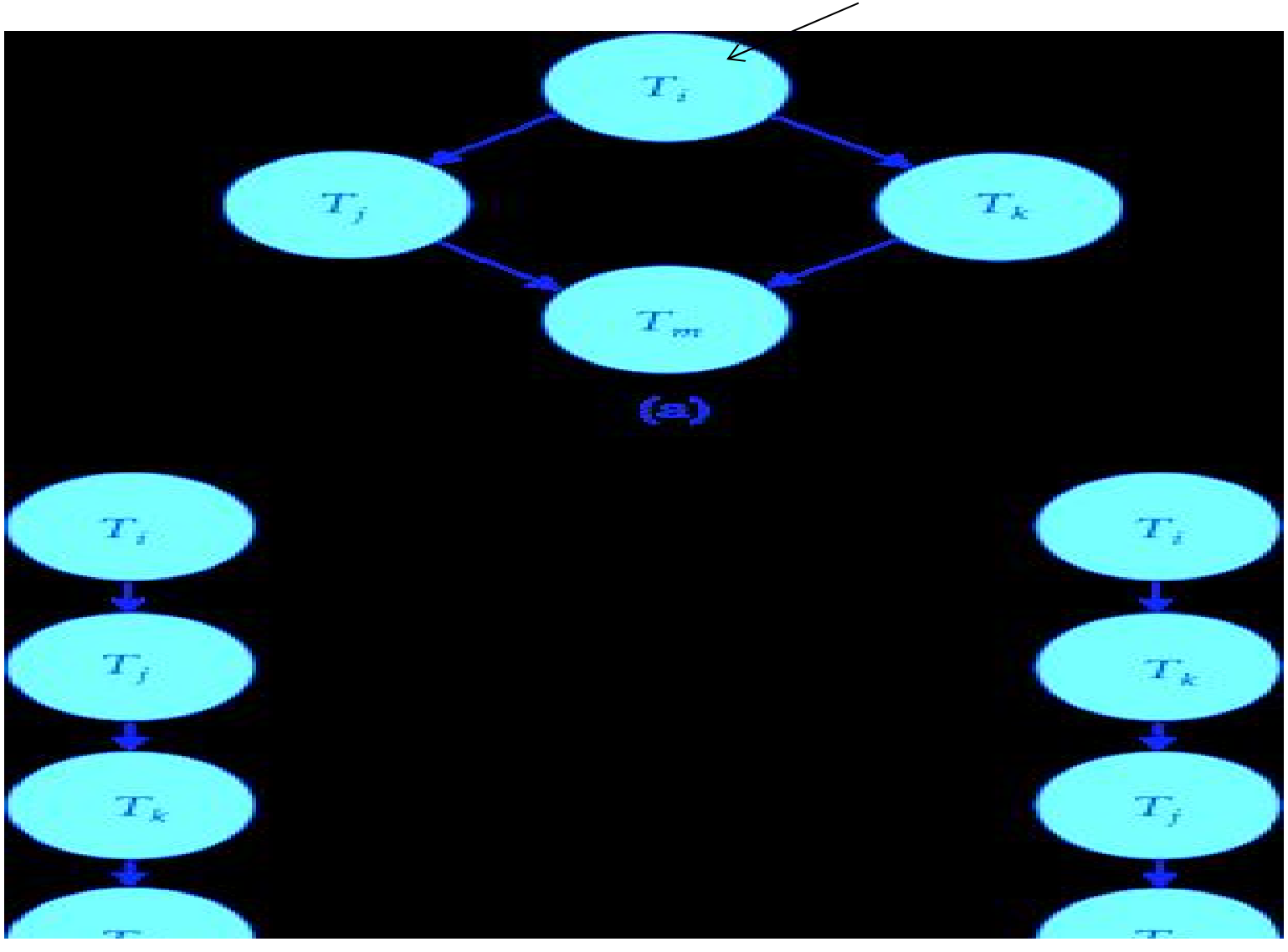
- 1. *Ii*= **read**(*Q*), *Ij*= **read**(*Q*). *Ii* ve *Ij* çakışmaz.
- 2. *Ii*= **read**(*Q*), *Ij*= **write**(*Q*). çakışır
- 3. *Ii*= **write**(*Q*), *Ij*= **read**(*Q*). çakışır
- 4. *Ii*= **write**(*Q*), *Ij*= **write**(*Q*). çakışır

Anomalies with Interleaved Execution
Girişik Çalışmadaki Uyumsuzluklar

- İşlenmemiş verinin okunması (WR çakışmaları, “kirli okuma”):
  - T1: R(A), W(A), R(B), W(B), Abort
  - T2: R(A), W(A), C
- Tekrarlanamaz okumalar (RW çakışmaları):
  - T1: R(A), R(A), W(A), C
  - T2: R(A), W(A), C
- İşlenmemiş verinin üzerine yazılması (WW çakışmaları):
  - T1: W(A), W(B), C
  - T2: W(A), W(B), C

Testing for Serializability (Serileştirilebilme ölçümü)

- Bir küme transaction'a ait bir schedule düşünün *T*1, *T*2, ..., *Tn*
- **Precedence graph (Öncelik graph'ı)** — köşelerin transaction'ları (isimleri) gösterdiği yönlü bir graph.
- iki transaction çakışıyorsa, ve *Ti* üzerinde daha önce çakışma olan bir veri öğesine eriştiyse, *Ti*'den *Tj* 'ye bir ok çiziyoruz.
- Ok'u erişilen öğe ile etiketleyebiliriz.


T_i
T_j
T_k
T_m
(a)
T_i
T_j
T_k
T_i
T_k
T_j

topological sorting (topolojik sıralama)

- Bir schedule, ancak öncelik (precedence) graph'ı döngüsüz (acyclic) ise serileştirilebilir.
- Cycle-detection (Döngü-tespit) algoritmaları mevcuttur
- Eğer precedence graph, acyclic ise, serileştirme düzeni, graph'ın *topolojik sıralanmasıyla* elde edilebilir.
- Bu, graph'ın kısmi düzeni ile tutarlı olan, doğrusal bir düzendir

Lock-Based Concurrency Control
Kilit-Tabanlı Birlikte İşleme Denetimi

Strict Two-phase Locking (Strict 2PL) Protocol:

Tam İki-aşamalı Kilitleme (Strict 2PL) Protokolü:

- Her bir xact okuma yapmadan önce nesne üzerinde bir S (*shared*) lock, ve yazmadan önce bir X (*exclusive*) lock almalıdır.
- Bir transaction'ın tuttuğu tüm kilitler transaction'lar tamamlandıktan sonra bırakılır
- Bir xact bir nesne üzerinde bir X lock tutuyorsa, başka hiçbir xact o nesne üzerinde kilit alamaz (S veya X)
- Strict 2PL sadece serileştirilebilir schedule'lara izin verir

The Two Phase Locking Protocol
İki Aşamalı Kilitleme Protokolü

- Phase 1: Growing Phase
- Aşama 1: Büyüme Aşaması
  - transaction may obtain locks (transaction kilit alabilir)
  - transaction may not release locks (transaction kilit bırakamaz)

- Phase 2: Shrinking Phase
- Aşama 2: Küçülme Aşaması
  - transaction may release locks (transaction kilit bırakabilir)
  - transaction may not obtain locks (transaction kilit alamaz)

Aborting a Transaction (Transaction Durdurulması)

- Transaction *Ti* iptal edilirse, yaptığı her şey geri alınmalıdır. Bu kadarla kalmayıp, eğer *Tj*, en son *Ti* tarafından yazılan bir nesneyi okuyorsa, *Tj* de durdurulmalıdır!
- Birçok sistem, bir xactin kilitlerini sadece işleme (commit) zamanında bırakarak (release), bu gibi *katlanan iptallerden* (*cascading aborts*) sakınır.
  - Eğer *Ti* bir nesneye yazarsa, *Tj* bu yazılanı sadece *Ti* işledikten (commit) sonra okuyabilir

The Log: to *undo* actions of an aborted transaction
Kayıt: iptal edilen transaction'ın yaptıklarını *geri almada* kullanılır

- Her yazma (write) işleminin kaydedildiği bir *log* tut. Bu mekanizma, sistem çökmelerinden dönüş için de kullanılır: sistem uyandığında, çökme anındaki bütün aktif xact'lar durdurulur.
- Kayıtta (log) tutulan değerler:
  - *Ti bir nesneye yazar*: eski ve yeni değer
  - *Ti işler(commit)/durur(abort)*: bu durumu belirten bir kayıt
- Log kayıtları, xact id ile birlikte tutulduğundan belirli bir xact'in geri alınması (undo) çok kolaydır

Current State

- Current DML: bir küme işlem bir transaction oluşturur.
  - SQL'de, bir transaction dolaylı olarak (implicitly) başlar.
  - **Commit work**, şimdiki transaction'ı işler ve yeni birisini başlatır.
  - **Rollback work**, şimdiki transaction'ı durdurur.
- Hemen hemen tüm database sistemlerinde, ön-tanımlı olarak (by default), her SQL ifadesi (statement) başarılı olursa dolaylı olarak (implicitly) işlenir (commit).
- Concurrency control (birlikte çalışma denetimi) ve recovery (geri kazanma), DBMS'in en önemli işlevlerindendir.
- Kullanımda olan bir DBMS şu işlevleri temin eder
  - Sistem, lock/unlock isteklerini (request) otomatik olarak yerleştirir
  - Farklı xact'lerin işlerinin zamanlamasını, elde edilen sonucun, aynı xact'lerin sırayla çalıştırılarak elde edilecek sonuçla aynı olmasını garantileyecek şekilde yapar.

*

Summary (Özet)

- Transactions: birlikte çalışan diğer transaction'lar server çökmelerine rağmen atomik olarak çalışacak bir dizi iş
- İşlenen xact'in etkisi: kalıcı olarak depolanan bir kayıt
- Operation conflicts (İşlem çakışmaları): concurrency control mekanizmalarına temel oluşturur
- Concurrency control: 2-phase locking (2-aşamalı kilitleme)
- 2PL ile schedule serializability (serileştirilebilirliği)