Transaction Isolation Düzeyleri

Eşzamanlı Erişim ve Transaction Isolation Problemleri

- DB'deki ortak bir veriye **birden fazla transaction**'ın **eş zamanlı erişimi** senkronizasyon problemleri ortaya çıkarır
- Temel olarak dört ana **senkronizasyon problemi** mevcuttur
  - Lost update
  - Dirty read
  - Unrepeatable read
  - Phantom read

Lost Update

TX 1	TX 2
Read bakiye 100	
	Read bakiye 100
Bakiye = bakiye + 100	
	Bakiye = bakiye - 100
Write bakiye 200	
	Write bakiye 0

Problem: TX 1'in bakiyede yaptığı güncelleme TX 1 aktif iken kaybolmuş oldu!

Çözüm: Read uncommitted

- Her TX istediği satırı **okuyabilir**
- Bir TX commit etmeden diğer TX aynı satırı **güncelleyemez**
- Ancak bir TX commit etmeden diğer TX aynı satırı okuyabilir **(dirty read)**

Dirty Read

TX 1	TX 2
Read bakiye 100	
Bakiye = bakiye + 100	
Write bakiye 200	
	Read bakiye 200
	Bakiye = bakiye + 200
	Write bakiye 400
ROLLBACK	

Problem: TX 1 yaptığı değişikliğin geri alınmasını istedi, ancak TX2 bundan evvel TX 1'in yaptığı değişikliği okumuş oldu!

Çözüm: **Read committed**

- **Salt okuma yapan TX, diğer TX'i engellemez**
- Ancak değişiklik yapıp **commit etmeyen** TX, diğer TX'in değişen satırı okumasını engeller

Unrepeatable Read

TX 1	TX 2
Read bakiye 100	
	Read bakiye 100
	Bakiye = bakiye + 200
	Write bakiye 300
	Commit
Read bakiye 300	

Problem: TX 1 bakiye değerini iki sefer okudu, ancak her ikisinde de farklı değer elde etti.

Oysa TX 1 boyunca bakiyenin her okunuşta aynı değeri vermesi belirli hesap işlemlerinde önem arz edebilir!

Çözüm: **Repeatable read**

- **Salt okuma** yapan TX, değişiklik yapacak diğer TX'i bloklar, fakat diğer TX'de salt okuma yapıyor ise engellemez
- **Değişiklik** yapan TX diğer bütün TX'leri engeller

Phantom Read

TX 1	TX 2
Read hesaplar H1,H2	
	Insert new hesap H3
	Commit
Read hesaplar H1,H2H3	

Problem: Müteakip iki okumada TX1 farklı sayıda hesap kaydı ile karşılaştı! İki okuma arasında yeni bir kayıt TX2 tarafından eklendi!

Çözüm: **Serializable**

- Concurrent TX'leri birbiri ardında çalışıyormuşlar gibi davrandırır
- Concurrent çalışan TX1 ve TX2, ardışık TX1,TX2 şeklinde çalışıyormuş gibi ele alınır

TX Isolation Düzeyinin Belirlenmesi

	Lost update	Dirty read	Unrepeatable read	Phantom read
READ_UNCOMMITTED				
READ_COMMITTED				
REPEATABLE_READ				
SERIALIZABLE				

TX Isolation Düzeyinin Belirlenmesi

JAVA Eğitimleri

- **Read uncommitted** isolation düzeyi kurumsal uygulamalarda hemen her zaman **elenebilir**
- **Phantom read**'ler de pek çok kurumsal uygulama için bir **problem oluşturmaz**
- Bu durumda seçenek olarak **read committed** veya **repeatable read**'den birisi kalmaktadır
- Coğu veritabanı **default "read committed"** düzeyindedir
- İstenirse bu Hibernate tarafında da değiştirilebilir

Veritabanlarının Default Transaction Isolation Düzeyleri

Veritabanı	Default İsolation Düzeyi
Oracle	READ_COMMITED
MS SQL	REPEATABLE_READ
MySQL	READ_COMMITED
PostgreSQL	READ_COMMITED
H2	READ_COMMITED
Apache Derby	READ_COMMITED
DB2	READ_COMMITED

TX Isolation Düzeyinin Değiştirilmesi

- Hibernate tarafında **hibernate.connection.isolation** konfigürasyon property'si ile bu düzey değiştirilebilir
- Alabileceği değerler
  - 1:read uncommitted
  - 2:read committed
  - 4:repeatable read
  - 8:serializable

Lost Update Problemi ve Isolation Düzeyleri

JAVA Eğitimleri

- **Eşzamanlı transaction**'larda repeatable read veya serializable isolation düzeyleri lost update'i engeller
- Ancak **detached entity**'ler ile çalışılıyor ise bir önceki oturumda yüklenmiş ve güncel olmayan bir entity reattach edilerek, üzerinde yapılacak bir güncelleme isolation düzeyi ne olursa olsun **lost update problemine yol açacak**tır
- Bu problemi önlemek için **optimistic lock mekanizması** kullanılabilir

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com