Persistence İşlemleri

harezmi bilişim çözümleri

Persistence Context Nedir?

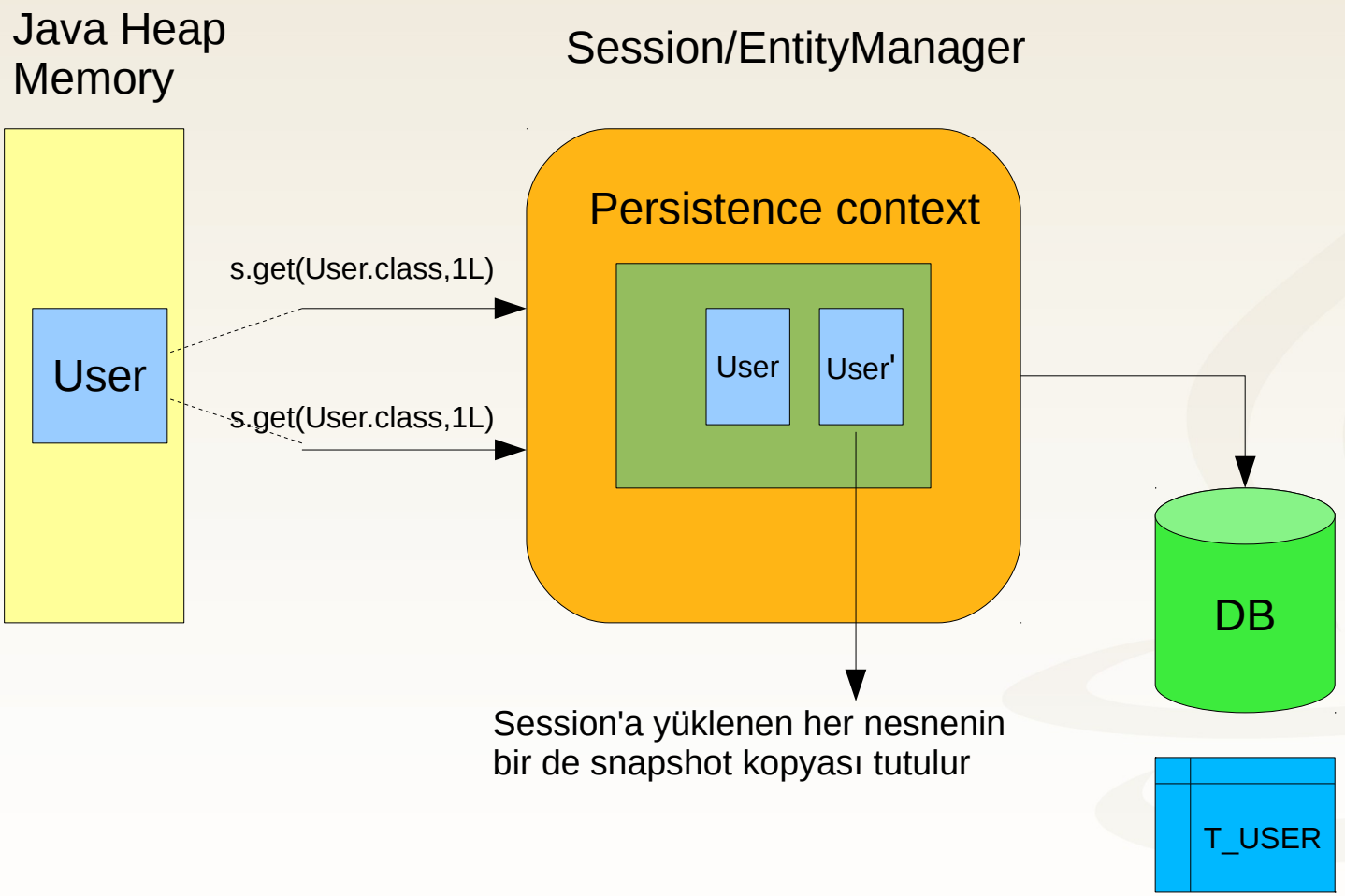
- Her **Session/EntityManager nesnesi** kendi içinde bir **persistence context** barındırır
- Persistence context, nesneler üzerinde yapılan **değişiklikleri** ve **nesnelerin state'lerini** takip eder
- **Doğrudan erişilemez**, işlemler Session üzerinden gerçekleştirilir
- Aynı Session'da birden fazla kez aynı entity'ye herhangi biçimde erişim **aynı nesne referansını** döndürür
- Bu yüzden **first-level cache** olarak da bilinir

Persistence Context ve Snapshot Kopyalar

JAVA Eğitimleri

- Persistence context'de **her nesnenin bir kopyası** tutulur
- Bu kopyaya **snapshot** adı verilir
- Hibernate **nesneler üzerindeki değişiklikleri tespit etmek** için asıl nesneyi ve snapshot kopyasını karşılaştırır
- **Uygulamanızdaki nesnelerin hafızada kapladığı alan** Hibernate persistence context'deki snapshot'larla birlikte yaklaşık x2 olmaktadır
- Bu durum göz ardı edilirse fazla sayıda entity yüklendiğinde **OutOfMemoryError** ile karşılaşabilirsiniz

Persistence Context ve Snapshot Kopyalar



Object Identity/Equality ve DB Identity Kavramları

- **Object identity** (==): iki nesne referansı aynı hafıza lokasyonunu gösteriyorsa bu nesneler "**identical**"dır
- **Object equality** (equals()): iki farklı hafıza lokasyonundaki nesnenin aynı değere sahip olmasına "**equivalence**" adı verilir
- **Database identity**: veritabanında saklanan nesneler aynı kayda karşılık geliyorlarsa db açısından "**identical**"dır

Persistent Nesnelere Erişim

- **Session.get(Class,Serializable)** ile PK kullanarak nesneler Session'a yüklenebilir
- Eğer entity halihazırda **Session'da mevcut ise** SELECT atmaz, Session'daki **mevcut entity nesneyi döner**
- PK'sı belirtilen entity **Session'da mevcut değil** ise kaydı DB'den **SELECT** eder
- Kayıt **DB'de de yok ise NULL** dönülür
- JPA'da get()'in karşılığı **find**(Class,Serializable) metodudur

Persistent Nesnelere Erişim ve Proxy

JAVA Eğitimleri

- Session.load() PK'sı belirtilen entity Session'da mevcut ise o entity nesneyi döner
- Entity Session'da mevcut değil ise **DB'ye hemen SELECT atmaz**, bunun yerine bir **proxy nesne döner**
- Proxy nesne **asıl nesnenin yerine** geçer
- Asıl nesneye gerçekten erişim ihtiyacı doğana kadar **veritabanı erişimini öteler**
- Asıl nesneye **sadece refer etmek** gerekiyorsa çok faydalıdır, (örneğin M:1 ve 1:1 ilişkiler)

Persistent Nesnelere Erişim ve Proxy

JAVA Eğitimleri

- Proxy nesnenin **herhangi bir alanına erişilmeye çalışıldığı vakit** DB den **SELECT** ile kayıt getirilir
- Bu aşamada belirtilen PK'ya karşılık gelen bir **kayıt DB'de yok ise** proxy nesne **ObjectNotFoundException** fırlatır
- Hibernate "**class proxy**" üretmektedir
- Dolayısı ile entity sınıfları ve içerisindeki public metotlar **final tanımlanmamalı**dır

Persistent Nesnelere Erişim ve Proxy

JAVA Eğitimleri

- Final tanımlanmış sınıflar için **proxy kabiliyeti devre dışı** bırakılır
- Proxy nesnenin yaratılacağı sınıf **SessionFactory build aşaması**nda üretilmektedir
- JPA'da load()'ın karşılığı ise **getReference**(Class,Serializable) metodudur
  - getReference **EntityNotFoundException** fırlatabilir

Transient Nesneler

- Bir nesne **new operatörü** ile uygulama içerisinde yaratıldığı vakit transient olarak kabul edilir
- Transient nesnenin **DB'de kayıt olarak karşılığı yok**tur
- ID değeri **NULL**'dır
- Ömürleri **process scope** ile aynıdır
- Uygulama kapatıldığında veya nesne referansı scope'dan çıktığında GC yapılabilir

Nesnenin Persistent Yapılması

- “Transitent” bir nesne **save** veya **persist** metotları ile “persistent” yapılır
- Yapılan işlemlerin DB'ye yansıtılması genellikle **TX commit** sırasında olur
- Hibernate'de save ve persist metotları her ikiside INSERT için kullanılabilir
- **Save** metodu INSERT edilen **entity'nin PK'sını döner**
- **Persist** metodunun **return tipi ise void**'tir

Save vs Persist

- **Save**
  - Entity'ye PK değerinin atanmasını hemen gerçekleştirir
  - Bunun için gerekiyorsa INSERT ifadesini TX içinde veya dışında da çalıştırabilir
  - Atanan PK değerini döner
- **Persist**
  - JPA'ya göre PK değerinin atamasını hemen gerçekleştirmeyip, TX commit'e kadar bekleyebilir
  - Ancak Hibernate impl'da persist metodu çağrıldığı vakit entity'ye PK değerini atamaktadır
  - INSERT ifadesi de TX commit anında çalıştırılır
  - Return tipi void'dir

Managed Nesneler

- Persistent nesnelerin mutlaka **veritabanı kimliği var**dır
- Başka bir ifade ile ID değeri mevcuttur
- Bu nesneler mutlaka bir **persistence context ile ilişkili**dir
- Uygulamada bu nesneler üzerinde yapılan **değişiklikler Hibernate tarafından takip edilir**

Managed Nesnenin Değiştirilmesi

- Öncelikle **get()**, **load()** veya **sorgu** ile nesnenin persistence context'e yüklenmesi gerekir
- Nesne persistence context ile ilişkili olduğu zaman müddetince üzerinde değişiklik yapılabilir
- Persistence context otomatik **dirty check** kontrolü yapar
- Tespit edilen değişiklikler veritabanına **TX commit** edildiği vakit otomatik olarak yansıtılır

Managed Nesnenin Değiştirilmesi

- Eğer bir property Session içerisinde birkaç defa değiştirilmiş ise sadece **bir UPDATE** ifadesi çalıştırılır
- UPDATE ifadesinde persistent entity'nin **bütün alanları güncellenir**

Detached Nesneler

- Session.**close()** metodu Session'ı ve içindeki persistence context'i kapatır
- Nesneler ait oldukları persistence context kapatıldığı vakit **detached** olurlar
- Nesne state'i ile DB state'i artık **senkronize edilmez**
- Detached nesnelerle uygulama içerisinde çalışmaya ve bunlar üzerinde **değişiklik yapmaya devam** edilebilir
- Ancak belli bir aşamada detached nesneler üzerindeki **değişiklikleri DB'ye yansıtmak gerekecektir**

Session Kapatmadan Detach Yapmak

- Session.**close() yapmadan** bütün persistent entity'leri detach yapmak için Session.**clear()** metodu kullanılabilir
- Session.clear() metodu aynı zamanda **snapshot kopyaları** da bırakır
- Mevcut bir transaction başlatılmış ise bu **transaction aynen devam** eder

Tek Bir Persistent Nesne'nin Detach Yapılması

JAVA Eğitimleri

- Sadece spesifik bir entity detach yapılabilir
- **Session.evict(entity)** metodu ile gerçekleşir
- JPA'daki karşılığı **detach** metodudur
- Session daki bir **entity'nin detach yapılması**nı sağlar
- Session entity üzerindeki **değişikliklerin takibini bırakır**

Detached Nesneleri Tekrar Session ile İlişkilendirmek

JAVA Eğitimleri

- Detached nesnedeki değişiklikleri DB'ye yansıtmak için **reattach** veya **merge** şarttır
- Değişikliğe uğramış detached nesnenin reattach edilmesi için **update()** metodu kullanılır
- update() metoduna parametre olarak verilen entity'ye Session "**dirty**" olarak davranır ve mutlaka **SQL UPDATE** ifadesini çalıştırır

Detached Nesneleri Tekrar Session ile İlişkilendirmek

JAVA Eğitimleri

- Değişikliğe uğramamış detached nesnenin reattach edilmesi için **lock()** metodu kullanılır
- Detached nesneyi **UPDATE ifadesi tetiklemeden** Session ile ilişkilendirir
- **lock() metodu çağırılmadan önce** nesne üzerinde yapılmış değişiklikler varsa bu **değişiklikler kaybolabilir**
- Lock metodu nesnenin sadece **detached state'den persistent state'e** geçmesini sağlar

Detached Nesneleri Tekrar Session ile İlişkilendirmek

- Eğer Session'da **aynı DB PK değeri ile başka bir nesne referansı** varsa eldeki detached nesnenin **reattachment'ı mümkün değil**dir
- Hem lock() hem de update() metotları böyle bir durumda **NonUniqueObjectException** fırlatılır
- Böyle bir durumda nesneyi Session'a reattach yapmak için **merge()** kullanılır
- Nesne merge() metodu çağırıldığında **session'da mevcut değil ise SELECT ile DB'den yüklenir**

Detached Nesneleri Tekrar Session ile İlişkilendirmek

JAVA Eğitimleri

- Sonra **değişiklikler** detached nesneden persistent nesneye **kopyalanır**
- Merge işlemi sırasında detached nesneden persistent nesneye yapılan **değişiklik aktarımında**
  - bütün **value tipli property**'ler
  - **Collection'lara yapılan ekleme ve silmeler** dikkate alınır
- **Detached nesne atılır**, session'daki **persistent nesne** metod değeri olarak **dönülür**

SelectBeforeUpdate

- **Detached** bir nesne update() veya saveOrUpdate() yapıldığında entity'nin eski halinin **snapshot'u Session'da yok**tur
- Bu durumda Hibernate mutlaka update yapacaktır
- Bu da **gereksiz UPDATE**'lere yol açabilir
- Örneğin, UPDATE sırasında çalışan trigger'lar varsa, bunlar gereksiz UPDATE nedeni ile çalışabilirler
- Bunun önüne geçmek için **@SelectBeforeUpdate** anotasyonu kullanılabilir

SelectBeforeUpdate

- Detached entity session'a **reattach edilirken** DB'den bir **SELECT** yapılır
- SELECT sonucuna göre entity üzerinde flush sırasında bir **UPDATE yapılıp yapılmayacağı**na karar verilir
- Çok fazla **SelectBeforeUpdate** bir performans problemi yaratabilir

```
@Entity
@org.hibernate.annotations.SelectBeforeUpdate(value = true)
public class Item {
//...
}
```

Transient Nesnenin Merge Edilmesi

- Eğer merge() metoduna input olarak verilen entity transient ise **entity DB'ye insert edilir**
- Eğer entity yeni yaratılmış ise **TX commit'de INSERT**, diğer durumda **UPDATE** ifadesi çalıştırılır
- **Assigned PK** yöntemi kullanılan entity'ler için merge() ile INSERT **gereksiz bir SELECT** işlemi tetikleyecektir
- Yeni bir entity yaratmak için en **iyi tercih persist()** metodudur

Hibernate saveOrUpdate

- Hibernate **saveOrUpdate()** verilen nesne üzerinde hangi işlemi gerçekleştireceğini bilir
- Bunun için **id property**'sine bakar, id NULL ise save, NOT NULL ise update yapar
- Eğer **doğal bileşke PK** kullanılıyor ve version/timestamp property mevcut değil ise Hibernate aynı PK'lı başka bir kayıt olup olmadığını tespit için **SELECT** gerçekleştirir
- Hemen her durumda, **ayrı save() ve update() metodları yerine saveOrUpdate()** metodu kullanılmalıdır

Detached Nesneler ve equals & hashCode Metotları

- **Detached nesneler**le çalışırken **equals()** ve **hashCode()** metotlarının yazılması önemlidir
- equals() metodunun **business key**'i oluşturan atribute'ları kullanarak yazılması en sağlıklı yaklaşımdır
- equals() ve hashCode() metotları içinde property değerlerine erişim **getter metotları** ile olmalıdır
- Çünkü proxy nesne söz konusu ise, **initialize** olması için property değerine getter metodu ile erişilmesi gerekir

Entity State'nin Tekrar DB'den Yüklenmesi

- **Session.refresh(entity)** metodu entity'nin **state**'ini **DB'den tekrar** yükler
- Entity **persistent** veya **detached** olabilir
- **JPA**'da ise entity **attached** vaziyette olmalıdır
- Buna optimistic lock **(version) alanı da dahil**dir
- Genellikle trigger, toplu güncelleme vb nedenlerle Hibernate'den bağımsız **değişiliklerin uygulama tarafına yansıtılması** için kullanılır

Persistent Entity'nin Silinmesi

JAVA Eğitimleri

- **delete()** ve **remove**() metotları ile gerçekleştirilir, JPA'da karşılığı **remove()**'dur
- Hibernate'de nesnenin önce **attached** olması şart değildir
- JPA'da ise silinecek entity **mutlaka attached** vaziyette olmalıdır
- delete() metodu iki şey yapar
  - Nesneyi detached ise **Session'a reattach** eder
  - **TX commit aşamasında DELETE** ifadesini tetikler

Persistent Entity'nin Silinmesi

- DELETE SQL ifadesi çalışırken DB’de belirtilen **entity mevcut değilse** OptimisticLockException fırlatılır
- Bu hata **version alanına sahip olsun olmasın** bütün entity’ler için geçerlidir

Persistent Entity'nin Silinmesi

- Eğer bir nesne session sonunda silinmek için bekliyorsa “**removed**” state'indedir
- Fakat nesne, session aktif olduğu müddetçe persistence context tarafından yönetilmektedir
- **hibernate.use_identifier_rollback** konfigürasyon değeri **true** ise Hibernate silinen nesnenin id property'sini **NULL**'a set eder

İletişim

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