Hibernate ve İkincil Ön Bellek (Caching)

harezmi bilişim çözümleri

Ön Bellek Nedir?

- Önbelleğin yaptığı iş, mevcut veritabanı state'ini **uygulamaya daha yakın bir yer**e getirmektir
- Bu yer **hafıza** veya **disk** olabilir
- Böylece **daha önce erişilmiş veri**, tekrar veritabanına gitmeden daha yakındaki bu alandan elde edilebilecektir
- Dolayısı ile önbellek işlemi tamamen **performans optimizasyonu** ile alakalıdır

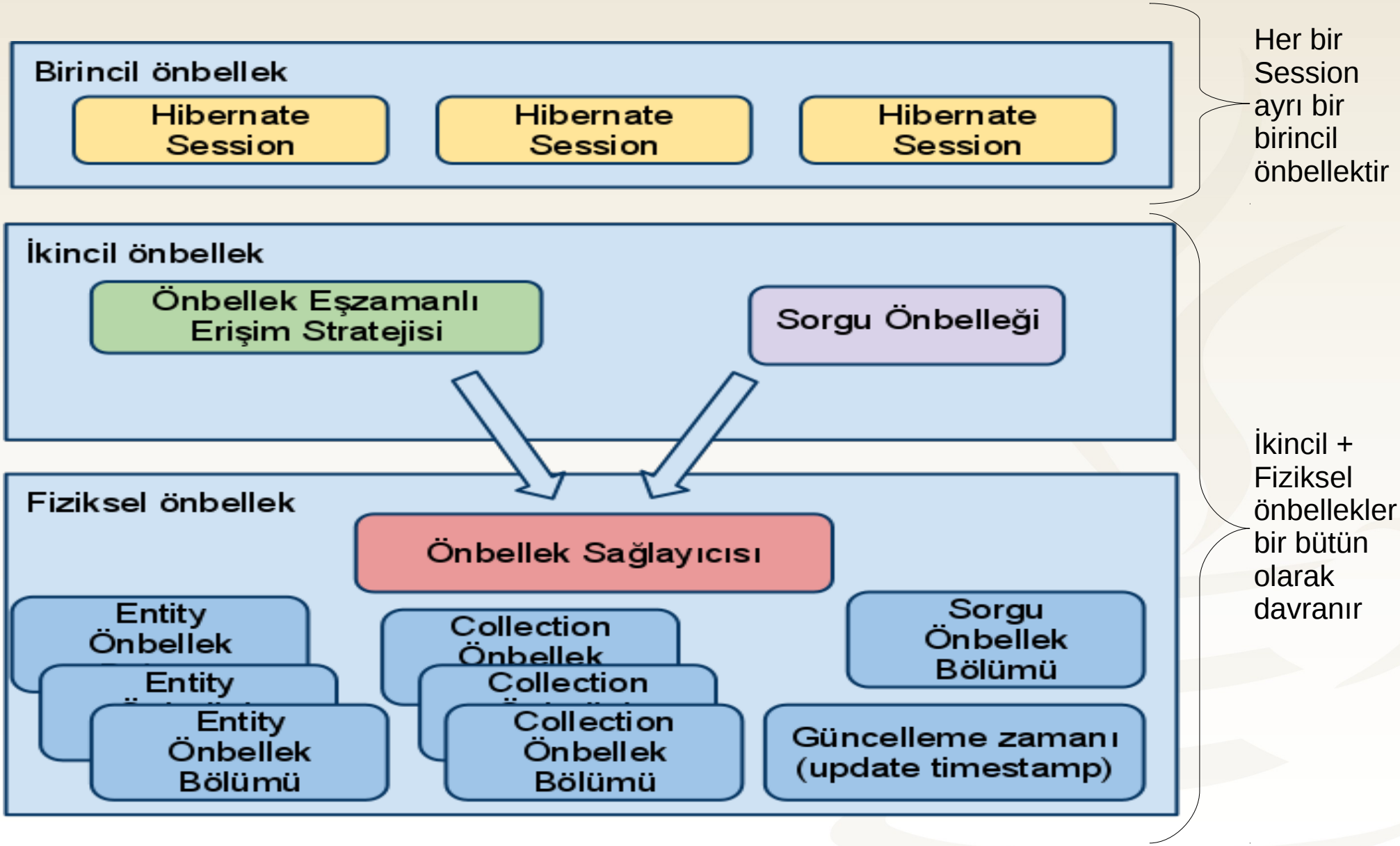
Hibernate ve Ön Bellek Mimarisi

- Hibernate **iki seviyeli önbellek mimarisi**ne sahiptir
- **İlk düzey** önbellek persistence context'dir (first level cache)
  - Zorunludur
  - Ömrü request veya conversation boyuncadır
- **İkinci düzey** önbellek ise process scope cache'dir (secondary cache)
  - Opsiyoneldir
  - Ömrü uygulamanın ömrü boyunca olabilir, yada cluster genelinde olabilir

Hibernate ve Ön Bellek Mimarisi

- Aynı **SessionFactory** üzerinden başlatılan bütün persistence context'ler **aynı ikincil cache**'i paylaşırlar
- Veri cache'de **disassembled formatt**a saklanır
- Önbellek tek tek **sınıf ve collection tabanlı ilişkiler düzeyinde** aktive edilebilir
- **Sorgu sonuçları** da önbellekte tutulabilir

Hibernate ve Ön Bellek Mimarisi



İkincil Ön Bellek Nerelerde Kullanılmalı?

JAVA Eğitimleri

- Cache'leme için **en iyi aday sınıfları**n özellikleri
  - Veri çok nadir değişir
  - Kritik veri içermez
  - Veri uygulamaya özeldir, paylaşılmaz
- İkincil cache'leme için **kötü adayların özellikleri**
  - Veri sık sık güncellenir
  - Finansal yani kritik veri içerir
  - Veri legacy bir uygulama ile paylaşılır
- **Stale veri** kabul edilemez ise **ikincil önbelleği devre dışı bırakmak** en iyi seçenektir

İkincil Ön Belleğin Devreye Alınması

- İkincil önbellek hibernate konfigürasyon dosyaları içerisinde aşağıdaki property'ler ile devreye alınır

Entity ve collection düzeyinde ikincil önbelleği aktive der

```
hibernate.cache.use_second_level_cache=true

hibernate.cache.use_query_cache=true

hibernate.cache.region.factory_class=org.hibernate.cache
.ehcache.EhCacheRegionFactory
```

HQL ve Criteria sorgularında Ikincil önbelleği aktive eder

Fiziksel önbellek sağlayıcısının sınıfı belirtilir. Bu sınıf üzerinden önbellek bölgeleri yönetilmektedir. JVM genelinde tek bir CacheManager instance'ı ile çalışmak için SingletonEhCacheRegionFactory kullanılabilir

Entity'lerin Ön Bellekte Tutulması

- Entity sınıfların önbellek alanında entity'nin bütün property değerleri ve **M:1 ve 1:1 ilişkilerinin sadece PK değerleri** tutulur

com.javaegitimleri.petclinic.model.Pet

1:{
name:"maviş",
birthDate:01.01.1970,
owner:1,
type:2
}

```
@Entity
@org.hibernate.annotations.Cache(usage =
org.hibernate.annotations.CacheConcurrencyStrategy.READ_WRITE)
public class Pet {
}
```

Entity'lerin Ön Bellekte Tutulması

- Bir entity önbellekten yüklenirse **1:1 ve M:1 ilişkilerinin gösterdiği entity'leri** Hibernate aşağıdaki sıra ile elde etmeye çalışır
  - Varsa Hibernate session'dan
  - İlişkinin hedef entity'si ikincil önbellekte tutuluyor ise ikincil önbellekten
  - Veritabanına ID ile bir sorgu yaparak

Entity'lerin Ön Bellekte Tutulması

- Dolayısı ile entity'ler önbellekte tutulacak ise bunları M:1 ve 1:1 ilişkili entity'lerinin de önbellekte tutulduğundan emin olunmalıdır
- Aksi takdirde bir ilişkiyi yüklemek için ilişkili her bir entity için **ayrı ayrı DB'ye sorgu atılması** söz konusu olacaktır!
- Embeddable **bileşenler** de ise **PK olmadığı için** bunlar önbellekte **dehydrated formda tutulmaktadır**

1:M ve M:N İlişkilerin Ön Bellekte Tutulması

- 1:M veya M:N bir ilişki için ikincil önbellek devreye alınırsa o ilişki için ayrılmış önbellek alanında **collection'daki elemanların sadece ID'leri** tutulacaktır

com.javaegitimleri.petclinic.model.Owner.pets

1:{101,102,103}
2:{104,105}
3:{}

```
@Entity
public class Owner {
    @OneToMany(mappedBy = "owner")
     @org.hibernate.annotations.Cache(usage =
org.hibernate.annotations.CacheConcurrencyStrategy.READ_WRITE)
    private Set<Pet> pets = new HashSet<Pet>();

}
```

1:M ve M:N İlişkilerin Ön Bellekte Tutulması

- Hibernate ilişkiyi yüklerken ilişkinin içerisinde yer alan **her bir Entity'yi ID'si üzerinden** aşağıdaki sıra ile elde etmeye çalışır
  - Varsa Hibernate session'dan
  - İlişkinin hedef entity'si ikincil önbellekte tutuluyor ise ikincil önbellekten
  - Veritabanına ID ile bir sorgu yaparak

1:M ve M:N İlişkilerin Ön Bellekte Tutulması

- Dolayısı ile ilişkileri ikincil önbellekte tutarken mutlaka ilişkinin **hedef entity'sinin de ön bellekte** tutulduğundan emin olunmalıdır
- Aksi takdirde bir ilişkiyi yüklemek için ilişkili her bir entity için **ayrı ayrı DB'ye sorgu atılması** söz konusu olacaktır!
- Embeddable **bileşenler** de ise **PK olmadığı için** bunlar önbellekte **dehydrated formda tutulmaktadır**

1:M ve M:N İlişkilerin Ön Bellekte Tutulması

- Collection ilişkilerinde **herhangi bir değişiklik** (eleman ekleme veya çıkarma) collection'ın önbellekten **invalidate**(evict) edilmesine neden olacaktır
- Evict işlemi eşzamanlı erişim stratejisi
  - READ_WRITE ise TX commit sonrasında
  - READ_WRITE_NONSTRICT ise TX commit öncesinde
  - TRANSACTIONAL ise tam TX commit anında gerçekleşir

1:M ve M:N İlişkilerin Ön Bellekte Tutulması

- **HQL** BULK UPDATE ve DELETE işlemleri de **ilgili collection önbellek bölgesi**nin invalidate edilmesine neden olur
- Eğer **Native SQL** çalıştırılır ise **ilgili önbellek bölgesinin sorguda belirtilmesi** gerekir
- Aksi takdirde **bütün önbellek bölgeleri invalidate** edilecektir

Sorgu Sonuçlarının Ön Bellekte Tutulması

- Hibernate sorguları da önbellekte tutulabilir
- Sorgu **ifadesi** ve **sorgu parametreleri** birlikte önbellekte saklanır
- Dolayısı ile bir sorgu ancak **aynı parametreler ile tekrar çalıştırıldığı vakit** sonucu önbellekten getirilecektir
- Eğer uygulama **çok sık olarak yazma işlemi** yapıyorsa sorgu önbelleği anlamlı değildir
- Sorgu cache'indeki değerlerle ilişkili insert, update, delete olduğunda entity ile ilgili sorgu **ön belleğindeki değerler invalidate** olur

Sorgu Sonuçlarının Ön Bellekte Tutulması

- **hibernate.cache.use_query_cache = true** sorgu önbelleğini devreye sokar
- Ancak ayrıca Query veya Criteria nesnesi üzerinde **setCacheable(true)** metodu çağrılarak spesifik sorgunun önbellek ile ilişkilendirilmesi sağlanmalıdır
- JPA'da ise **Query.setHint("org.hibernate.cacheable", true)** ile sorgu cache'i aktive edilebilir

Sorgu Sonuçlarının Ön Bellekte Tutulması

JAVA Eğitimleri

```
Query ownerByName = session.createQuery(
"select o.firstName, o.lastName from Owner o
where o.username = :username");

ownerByName.setString("username", username);
ownerByName.setCacheable(true);
```

```
Criteria criteria = session.createCriteria(User.class);

criteria.add( Restrictions.naturalId()
.set("username", "ksevindik")
.set("emailAddress", "ksevindik@gmail.com"));

criteria.setCacheable(true);
```

Sorgu Sonuçlarının Ön Bellekte Tutulması

- Sorgu önbellek alanında sorgu sonuçları eğer **scalar ise değerleri**, persistent entity ise **sadece entity PK değerleri** tutulur

Default query cache region

(from Pet p where p.birthDate = ?,01.01.2001) : {1,2,3,4,5,6}

select p.name,p.birthDate from Pet p where p.id in ?,(1,2,3) :{
[“maviş”,01.01.1970],
[“cimcime”,01.02.1980],
[“karabaş”,01.03.1990]
}

select p.name,p.birthDate from Pet p where p.id in ?,(1,2) :{
[“maviş”,01.01.1970],
[“cimcime”,01.02.1980],
}

Sorgu Sonuçlarının Ön Bellekte Tutulması

- Eğer cacheable sorgu sonucu dönen değer(ler) entity ise **her bir Entity, ID'si üzerinden** aşağıdaki sıra ile elde edilmeye çalışılır
  - Varsa Hibernate session'dan
  - İlişkinin hedef entity'si ikincil önbellekte tutuluyor ise ikincil önbellekten
  - Veritabanına ID ile bir sorgu yaparak
- Bu nedenle cacheable **sorguların döndüğü entity'lerin de cache'lenmesi** önemlidir!

Ön Bellek ve Eşzamanlı Erişim

- İkincil önbellek veriyi aynı anda **birden fazla Session** tarafından erişilebilir kılar
- Birden fazla Session, ikincil önbellek içerisindeki aynı nesneye **eşzamanlı biçimde erişebilir** ve bu veri üzerinde **değişiklik yapabilir**
- Dolayısı ile önbellekteki persistent nesnelere **erişimin senkronize edilmesi** gerekir
- Eş zamanlı **önbellek erişim stratejileri** ile önbellekteki veriye erişim ve verinin güncellenmesi düzenlenir

Eşzamanlı Erişim Stratejileri

- **Eşzamanlı erişim stratejisi** verinin önbellek içerisinde saklanmasından, güncellenmesinden ve erişilmesinden sorumludur
- Her bir **entity ve collection ilişki** için hangi önbellek eşzamanlı erişim stratejisinin kullanılacağı tanımlanmalıdır

Eşzamanlı Erişim Stratejileri

- **Read-only**
  - Verinin **hiç değişmediği** durumlar içindir
  - Salt okunur entity'ler **dirty-check** işlemine tabi tutulmazlar
  - Bu entity'ler **güncellenemezler**, sadece okunabilir ve silinebilirler

Eşzamanlı Erişim Stratejileri

- **Read-write**
  - Sık okunan ve yazılan veride kullanılabilir
  - Veri üzerindeki değişiklik **önbelleğin de güncellenmesi**ni tetikler
  - Değişiklikler önbelleğe **TX commit sonrası** yansıtılır

Eşzamanlı Erişim Stratejileri

- **Nonstrict-read-write**
  - Sıklıkla okunan, **az da olsa değişen veri** için daha uygundur
  - Eğer veri üzerinde güncelleme yapılırsa **önbellek invalidate** edilir
  - Ancak veritabanı TX ve önbellek invalidasyonu senkron yürütülmez
  - Bu durumda azda olsa **stale veri**ye erişim ihtimali ortaya çıkar
  - Veri üzerinde değişiklik yapılmaz ise **zaman aşım periyodu**na kadar veri güncel kabul edilir

Eşzamanlı Erişim Stratejileri

- **Transactional**
  - Sadece **transactional önbellek sağlayıcı**ları ile kullanılabilir
  - **EhCache** desteklemektedir
  - Uygulama içerisinde **JTA** ile birlikte kullanılmalıdır
  - Transaction commit sırasında **önbellek senkronizasyonu** da gerçekleştirir

Uygun Eşzamanlı Erişim Stratejisinin Seçilmesi

```
@Entity
@org.hibernate.annotations.Cache(usage =
org.hibernate.annotations.CacheConcurrencyStrategy.READ_WRITE)
public class Pet {

}

@Entity
public class Owner {

    @OneToMany(mappedBy = "owner")
    @org.hibernate.annotations.Cache(usage =
org.hibernate.annotations.CacheConcurrencyStrategy.READ_WRITE)
    private Set<Pet> pets = new HashSet<Pet>();


}
```

Entity düzeyinde ikincil önbellek tanımı

İlişki düzeyinde ikincil önbellek tanımı

İletişim

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