İleri Düzey JPA/Hibernate Eğitimi 4

harezmi bilişim çözümleri

Native SQL Sorguları

- Session/EntityManager üzerinden **native SQL** de çalıştırılabilir
- Native SQL sorgularında istenirse **List of entity**, istenirse de **List of Object[]** dönülebilir
- Sorgu sonucu dönen entity nesneleri **Session/EntityManager ile ilişkili**dir
- Dolayısı ile bu entity nesneler üzerinde yapılan **değişiklikler otomatik olarak DB'ye yansır**

Native SQL Sorguları ve List of Object[] Dönülmesi

```
session.createSQLQuery("select * from T_PET")
        .list();

session.createSQLQuery("select ID,NAME from T_PET")
        .list();
```

Yukarıdaki native SQL sorguları sonucu List of Object[] dönülecektir. Her bir sütun değeri **ResultSetMetadata'ya bakılarak** SQL tipinden bir Java nesnesine dönüştürülecektir. Aşağıdaki yöntemle **ResultSetMetadata'ya bakmadan** her bir sütunun hangi Java tipine karşılık geldiği de belirtilebilir

```
session.createSQLQuery("select * from T_SPECIALTY")
        .addScalar("ID",Hibernate.LONG)
        .addScalar("NAME",Hibernate.STRING)
        .addScalar("BIRTH_DATE",Hibernate.DATE)
        .list();
```

Native SQL Sorguları ve Persistent Entity Dönülmesi

```
session.createSQLQuery("select * from T_PET")
        .addEntity(Pet.class)
        .list();
```

Sorgu sonucu dönen değerlerin entity nesnelere dönüştürülmesi de mümkündür. Dönüşüm sırasında sütun isimleri ile property'ler arasındaki **eşleştirme metadata üzerinden** sağlanır.

SQL sorgusu içerisinde JOIN yapmak ve parametreler tanımlamak da mümkündür. Sorgu sonucu dönen sütunlar {p.*} şeklinde placeholder alias ile yine spesifik entity'lere dönüştürülebilir.

```
session.createSQLQuery("select {p.*} from T_PET p"
           + " join T_PET_TYPE t on p.TYPE_ID = t.ID"
           + " where t.NAME = :name")
        .addEntity("p",Pet.class)
        .setParameter("name", "Kedi")
        .list();
```

Native SQL Sorguları ve ResultTransformer

```
session.createSQLQuery("select {p.*}, {v.*} from T_PET p,
T_VISIT v where p.ID = v.PET_ID and p.NAME = ?")
        .addEntity("p",Pet.class)
        .addEntity("v",Visit.class)
        .setParameter(0,"Maviş")
        .list();
```

Sorgu sonucu birden fazla entity ile de eşleştirilebilir. Yukarıdaki sorgu da List of Object[] dönülecek, Object[]'in ilk elemanı Pet, ikinci elemanı ise Visit nesneleri olacaktır.

İstenirse bunlardan herhangi birine **ResultTransformer** ile project edilebilir.

```
session.createSQLQuery("select {p.*}, {v.*} from T_PET p,
T_VISIT v where p.ID = v.PET_ID")
      .addEntity("v",Visit.class)
      .addEntity("p",Pet.class)
   .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY)
      .list();
```

Native SQL Sorguları ve Eager Fetch

```
session.createSQLQuery("select {p.*}, {v.*} from T_PET p
left outer join T_VISIT v")
      .addEntity("v",Visit.class)
      .addEntity("p",Pet.class)
      .addJoin("v","p.visits")
      .addEntity("p",Pet.class)
      .setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY)
      .list();
```

Lazy property veya collection ilişkileri EAGER FETCH de yapılabilir.
Burada dikkat edilmesi gereken nokta eğer ResultTransformer kullanılacak ise addJoin nedeni ile root entity olarak Visit dönülmektedir!
Dönülecek entity'nin Pet olması isteniyorsa addEntity ile Pet.class tekrar belirtilmelidir.

JPA'da Native SQL ile Scalar/Sütun Değerler Dönmek

- JPA’da native SQL sorguları da **belirli sütunlara project** ettirilerek çalıştırılabilir
- Sorgu sonucu **List<Object[]>** dönülecektir

```
javax.persistence.Query query =
                    entityManager.createNativeQuery(
        "select ID,NAME,BIRTH_DATE from T_PET where NAME like :name");

query.setParameter("name", "%e");

List<Object[]> resultList = query.getResultList();

for (Object[] arr: resultList) {
    System.out.println(ArrayUtils.toString(arr));
}
```

JPA'da Native SQL ile Entity Dönmek

- **Entity nesne** dönülebilmesi için için ise sorguda entity ile ilgili **bütün sütunlar mevcut olmalı**dır ve ayrıca Entity'nin tipi de belirtilmelidir

```
Query query = entityManager.createNativeQuery(
                "select * from T_PET where NAME like :name",Pet.class);

query.setParameter("name", "%e");

List<Pet> resultList = query.getResultList();

for (Pet pet : resultList) {
    System.out.println(pet);
}
```

JPA'da Native SQL ile Birden Fazla Entity Dönmek

- Eğer native SQL sorgusu birden fazla tipte entity tipi veya scalar değeri birlikte dönüyorsa explicit **resultset mapping metadata**'sına ihtiyaç vardır

```
Query query = entityManager.createNativeQuery(
"select p.ID,p.NAME,p.BIRTH_DATE,p.OWNER_ID,p.TYPE_ID,p.OPT_LOCK_VERSION,"
+ "v.ID,v.VISIT_DATE,v.DESCRIPTION,v.CHECKUP,v.PET_ID,v.OPT_LOCK_VERSION "
+ "from T_PET p left outer join T_VISIT v on p.ID = v.PET_ID where NAME like
:name",
"petsWithVisits");

query.setParameter("name", "%e");

List<Object[]> resultList = query.getResultList();

for (Object[] arr : resultList) {
    Pet p = (Pet) arr[0];
    Visit v = (Visit) arr[1];
    System.out.println(p.getName());
    System.out.println(v.getDescription());
}
```

ResultSet mapping metadata @SqlResultSetMapping anotasyonu ile ayrıca tanımlanmış olmalıdır

JPA'da Native SQL ve @SqlResultSetMapping

```
@SqlResultSetMapping(name="petsWithVisits",entities={
    @EntityResult(entityClass=Pet.class,fields={
        @FieldResult(column="ID",name="id"),
        @FieldResult(column="NAME",name="name"),
        @FieldResult(column="BIRTH_DATE",name="birthDate"),
        @FieldResult(column="OPT_LOCK_VERSION",name="version"),
        @FieldResult(column="OWNER_ID",name="owner"),
        @FieldResult(column="TYPE_ID",name="type")
    }),
    @EntityResult(entityClass=Visit.class,fields={
        @FieldResult(column="ID",name="id"),
        @FieldResult(column="VISIT_DATE",name="date"),
        @FieldResult(column="DESCRIPTION",name="description"),
        @FieldResult(column="OPT_LOCK_VERSION",name="version"),
        @FieldResult(column="CHECKUP",name="checkup"),
        @FieldResult(column="PET_ID",name="pet")
    })
})
```

Anotasyon yerine **META-INF/orm.xml** içerisinde **<sql-result-set-mapping>** elemanı ile de tanımlanabilir

JPA'da Native SQL ve <sql-result-set-mapping>

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings ...>
    <sql-result-set-mapping name="petsWithVisits">
        <entity-result entity-class="com.javaegitimleri.petclinic.model.Pet">
            <field-result name="id" column="ID" />
            <field-result name="name" column="NAME" />
            <field-result name="birthDate" column="BIRTH_DATE" />
            <field-result name="owner" column="OWNER_ID" />
            <field-result name="type" column="TYPE_ID" />
            <field-result name="version" column="OPT_LOCK_VERSION" />
        </entity-result>
        <entity-result entity-class="com.javaegitimleri.petclinic.model.Visit">
            <field-result name="id" column="ID" />
            <field-result name="date" column="VISIT_DATE" />
            <field-result name="description" column="DESCRIPTION" />
            <field-result name="checkup" column="CHECKUP" />
            <field-result name="pet" column="PET_ID" />
            <field-result name="version" column="OPT_LOCK_VERSION" />
        </entity-result>
    </sql-result-set-mapping>
</entity-mappings>
```

JPA'da Native SQL ile Scalar/Sütun Değerler Dönmek

- Entity nesneler, **scalar değer**ler veya tek tek sütun değerleri ile birlikte de döndürülebilir

```
Query query = entityManager.createNativeQuery("select p.ID,p.NAME,p.BIRTH_DATE,"
    + "p.OWNER_ID,p.TYPE_ID,p.OPT_LOCK_VERSION, count(v.ID) as VISIT_COUNT "
    + "from T_PET p left join T_VISIT v on p.ID = v.PET_ID "
    + "group by p.ID,p.NAME,p.BIRTH_DATE,p.OWNER_ID,p.TYPE_ID,p.OPT_LOCK_VERSION",
"petsWithVisitCounts");

List<Object[]> resultList = query.getResultList();
for (Object[] arr : resultList) {
    Pet p = (Pet) arr[0];Long visitCount = (Long) arr[1];
    System.out.println(p.getName() + ":" + visitCount);
}

@SqlResultSetMapping(name = "petsWithVisitCounts", entities = {
    @EntityResult(entityClass = Pet.class, fields = {
    @FieldResult(column = "ID", name = "id"),
    @FieldResult(column = "NAME", name = "name"),
    @FieldResult(column = "BIRTH_DATE", name = "birthDate"),
    @FieldResult(column = "OPT_LOCK_VERSION", name = "version"),
    @FieldResult(column = "OWNER_ID", name = "owner"),
    @FieldResult(column = "TYPE_ID", name = "type")}
)}, columns = { @ColumnResult(type = Long.class, name = "VISIT_COUNT") })
```

JPA'da Native SQL ve Constructor Result Mapping

- Native SQL sorgu sonucu persistence context ile ilişkili entity dönmek yerine **DTO (value object) dönmek** de mümkündür
- JPQL'deki **constructor expression**'lara benzer

```
Query query = entityManager.createNativeQuery(
    "select p.ID,p.NAME,count(v.ID) as VISIT_COUNT "
    + "from T_PET p left join T_VISIT v on p.ID = v.PET_ID "
    + "group by p.ID,p.NAME", "petVisitInfoList");

List<PetVisitInfo> resultList = query.getResultList();

for (PetVisitInfo petVisitInfo : resultList) {
    System.out.println(
        petVisitInfo.getName() + " :" + petVisitInfo.getVisitCount());
}
```

@SqlResultSetMapping anotasyonu ile ayrıca tanımlanmış olmalıdır

JPA'da Native SQL ve Constructor Result Mapping

JAVA Eğitimleri

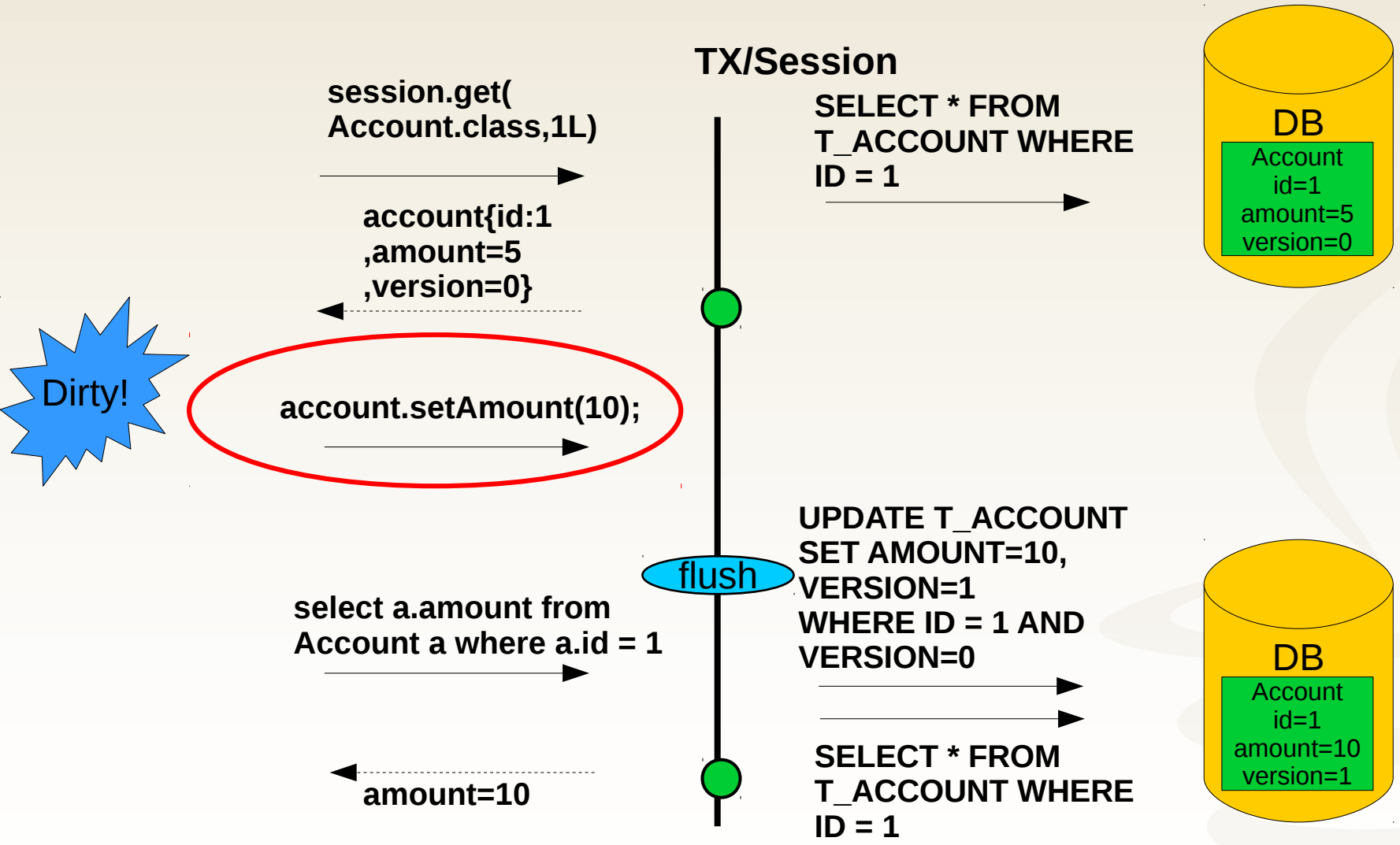
```java
public class PetVisitInfo {
    private Long id;
    private String name;
    private Long visitCount;

    public PetVisitInfo(Long id, String name, Long visitCount) {
        this.id = id;
        this.name = name;
        this.visitCount = visitCount;
    }
    //getters...
}
```

```java
@SqlResultSetMapping(name = "petVisitInfoList", classes={
    @ConstructorResult(targetClass=PetVisitInfo.class,
        columns={
            @ColumnResult(name="ID",type=Long.class),
            @ColumnResult(name="NAME",type=String.class),
            @ColumnResult(name="VISIT_COUNT",type=Long.class)
        })
})
```

Sorgular ve Flush

- HQL/JPQL ve Criteria sorguları **mutlaka DB'ye gider**
- Sorgunun çalıştırıldığı Session'da sorguda herhangi bir biçimde yer alan **entity'ler dirty vaziyette ise sorgu öncesinde flush** yapılır (FlushMode.AUTO)
- Bu sayede değişikliklerin yapıldığı TX içerisinde de **HQL/JQL sorgusunun güncel veri dönmesi** sağlanır

Sorgular ve Flush

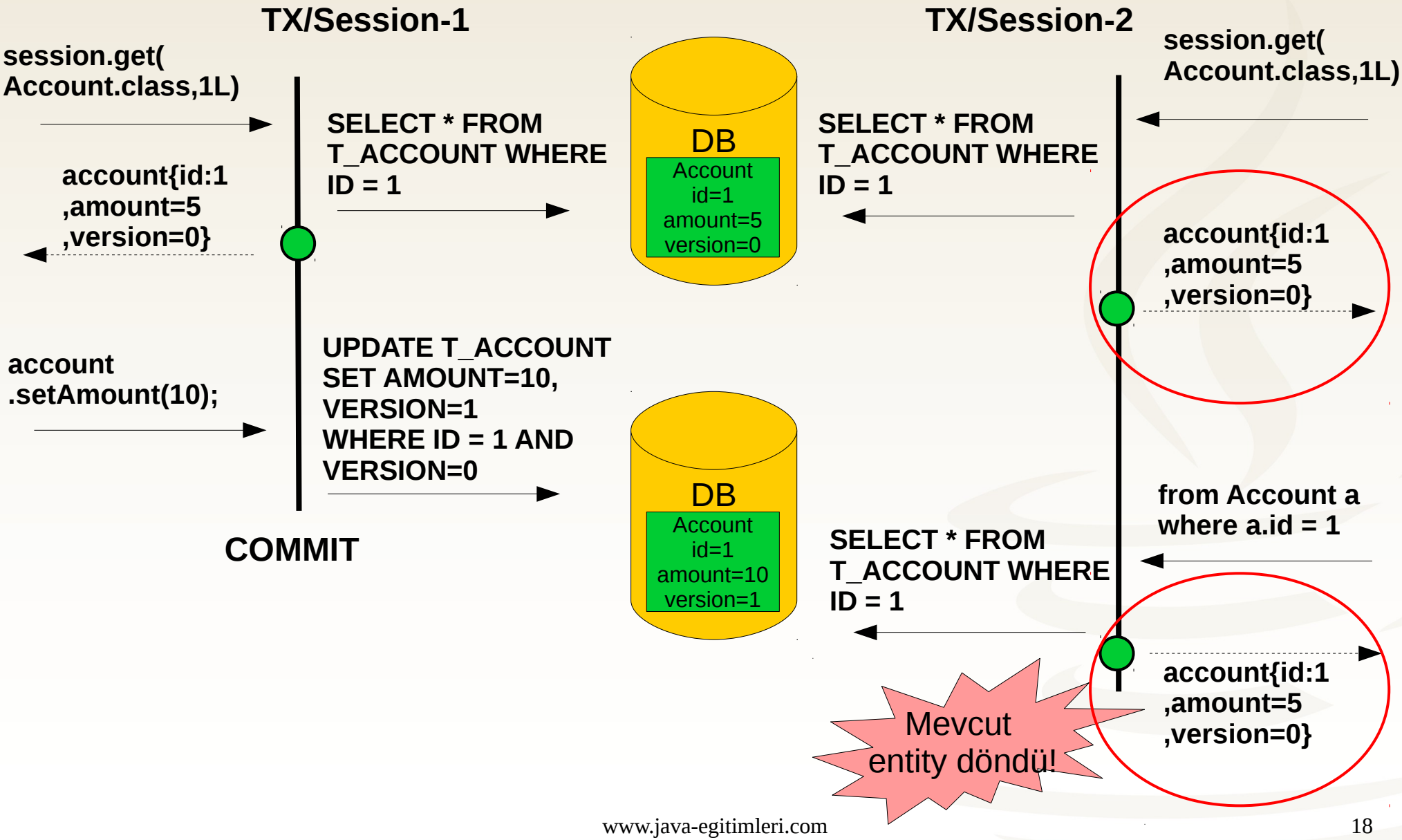


Sorgular ve Persistence Context'deki Entity'ler

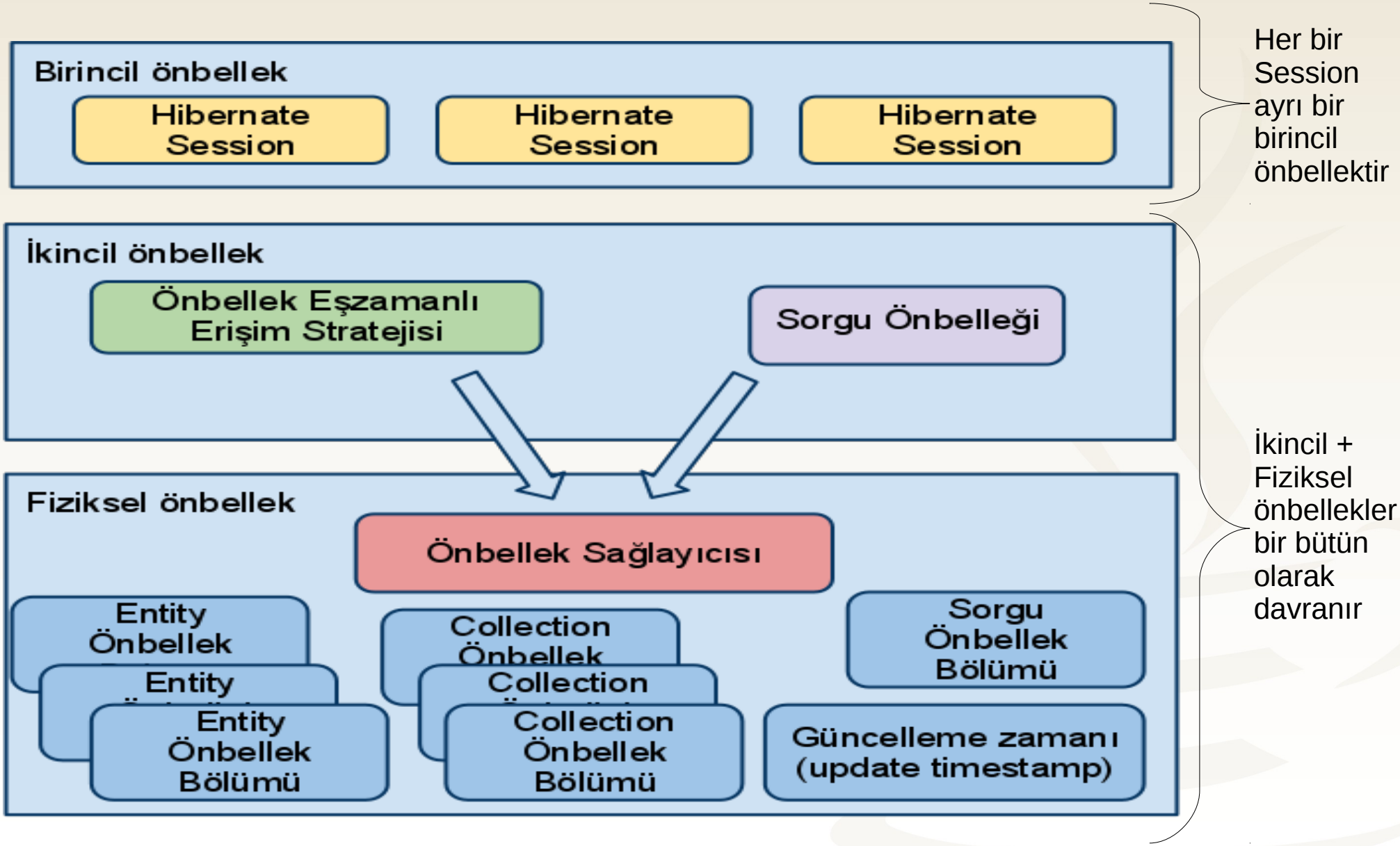
JAVA Eğitimleri

- Sorgunun çalıştırıldığı Session'da, sorgu sonucu dönen **entity halihazırda mevcut** olabilir
- Bu durumda sorgu DB'ye gidip veriyi çekse bile, DB'den dönen sonuç yerine **Session'da daha önceden yüklenmiş entity** dönülür
- Böyle bir durumda **DB'deki güncel veri yerine daha eski bir entity state'ine erişmek** söz konusu olabilir
- Session'daki entity'nin state'ini DB'deki en güncel veri ile eşitlemek için **Session.refresh(entity)** kullanılabilir

Sorgular ve Persistence Context'deki Entity'ler



Hibernate ve Ön Bellek Mimarisi



Ön Bellek Bölgelerinin İsimlendirmesi

- Entity önbellek alanlarının ismi **sınıf isminin FQN**'idir
  - com.javaegitimleri.petclinic.model.Pet
- Collection ilişkilerinin önbellek alanlarının ismi ilişkinin ait olduğu entity sınıfın **FQN + property** ismidir
  - com.javaegitimleri.petclinic.model.Pet.visits
- Sorgular için default olarak tek bir sorgu önbellek alanı bulunur

Ön Bellek Bölgelerinin İsimlendirmesi

- Belirli bir SessionFactory veya persistence unit için **hibernate.cache.region_prefix** konfigürasyon property'si ile bölge adı değiştirilebilir
- Eğer uygulama **birden fazla SessionFactory** veya persistence unit kullanıyor ise bu özellik faydalıdır
- Bu olmadan cache bölge isimleri farklı persistence unit'lerde **çakışma** yaşayacaktır

Ön Bellek Bölgelerinin Konfigürasyonu

```
<cache
name="com.javaegitimleri.petclinic.model.Owner"

maxElementsInMemory="500"
        eternal="true"
        timeToIdleSeconds="0"
        timeToLiveSeconds="0"
        overflowToDisk="false"
/>

<cache
name="com.javaegitimleri.petclinic.model.Pet"
    maxElementsInMemory="50000"
        eternal="false"

timeToIdleSeconds="1800"

timeToLiveSeconds="100000"
        overflowToDisk="false"
/>
```

- **overflowToDisk="false"** hafızadaki alan dolduğu vakit veriyi diskte tutup tutmamaya karar verir
- **eternal="true"** timeout üzerinden evict işlemini kontrol eder
- Eğer cache size nesne sayısından fazla olursa evict işlemi devre dışı kalır
- Son erişimden bu yana zaman aşımını **timeToIdleSeconds** belirler
- Veri cache'e eklendikten sonraki zaman aşımı süresini **timeToLiveSeconds** belirler

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!

İkincil Ön Belleğe Programatik Erişim

- **SessionFactory.getCache()** ile **ikincil önbelleğ**e erişilebilir
- **Entity'lerin veya collection ilişkilerin** bu önbellek alanlarında mevcut olup olmadıkları tespit edilebilir
- Önbellekte tutulan entity'ler, collection ilişkileri ve sorgu sonuçları **evict** edilebilir
- Önbellek alanları **toptan temizlenebilir**

İkinci Seviye Ön Belleğe Erişimin Kontrolü

- **CacheMode**, Hibernate'in ikincil önbellek ile etkileşimini Session veya sorgu düzeyinde kontrol etmeyi sağlar
- **NORMAL**
  - default mode
  - Veri önbellekten okunur, ön belleğe yazılır.
- **IGNORE**
  - Cache'ten okuma yaptırmaz, veri update sırasında cache invalidate ettirilir

İkinci Seviye Ön Belleğe Erişimin Kontrolü

- **GET**
  - Veri cache'den okunur, ancak cache'e put yapılmaz, veri update sırasında cache invalidate ettirilir
- **PUT**
  - Veri cache'e put yapılır, ancak cache'den okunmaz
- **REFRESH**
  - PUT moduna benzer, tek farkı **hibernate.cache.use_minimal_puts** özelliğinin gözardı edilmesidir.
  - Bu sayede önbellekteki veri her seferinde yenilenmiş olur.

İkinci Seviye Ön Belleğe Erişimin Kontrolü

```
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
session.setCacheMode(CacheMode.IGNORE);

for ( int i=0; i<100000; i++ ) {
    Item item = new Item();
    session.save(item);
    if ( i % 100 == 0 ) {
        session.flush();
        session.clear();
    }
}

tx.commit();
session.close();
```

JPA ve İkincil Önbellek Desteği

JAVA Eğitimleri

- JPA 2.0 ile gelen yeniliklerden birisi de **ikincil önbellek desteği**dir
- JPA spesifikasyonunda ikincil önbellek kabiliyeti **opsiyoneldir**, JPA provider ikincil önbellek kabiliyetine sahip olmayabilir
- Böyle bir durumda JPA 2 önbellek ayarları sessizce **göz ardı** edilir

JPA ve İkincil Önbellek Desteği

- Cache konfigürasyonu **persistence unit düzeyinde** yapılır
- Hibernate'de de konfigürasyon **SessionFactory düzeyinde** yapılmaktadır
- Genel olarak Hibernate ile ikincil önbellek kabiliyetine **benzer** bir konfigürasyon ve kullanıma sahiptir
- Ancak Hibernate'deki kadar kapsamlı bir çözüm sunmamaktadır

JPA @Cacheable Anotasyonu

- Entity düzeyinde @Cacheable anotasyonu ile yönetilir

```
@Entity
@javax.persistence.Cacheable(true)
public class Pet {
...
}
```

JPA İkincil Önbellek Konfigürasyonu

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="testjpa">
        <shared-cache-mode>ALL</shared-cache-mode>
        <properties>
            <property name="hibernate.ejb.cfgfile" value="/hibernate.cfg.xml" />
        </properties>
    </persistence-unit>
</persistence>
```

Alabileceği değerler:
NONE
ALL
ENABLE_SELECTIVE
DISABLE_SELECTIVE
UNSPECIFIED

JPA İkincil Önbellek Modları

- **ENABLE_SELECTIVE**: **@Cacheable(true)** anotasyonuna sahip **entity sınıfları** için önbellekleme aktif olur
- **DISABLE_SELECTIVE**: **@Cacheable(false)** ile işaretlenmeyen bütün **entity sınıfları** otomatik olarak önbelleklemeye tabi tutulur
- **NONE**: @Cacheable annotasyonuna bakılmaksızın ikincil önbellek **devre dışı** bırakılır
- **ALL**: @Cacheable annotasyonuna bakmaksızın **bütün entity'ler** için devreye alınır
- **UNSPECIFIED**: Persistence **provider'ın default davranışı**na bakılır

JPA İkincil Önbellek ve Eşzamanlı Erişim Stratejisi

JAVA Eğitimleri

- JPA'da Hibernate'de olduğu gibi **entity veya collection ilişkisi düzeyinde eşzamanlı erişim stratejisi belirtmek mümkün değil**dir
- **EntityManager** düzeyinde set edilebilen **retrieveMode** ve **storeMode** property'leri ile ikincil önbellek erişimi kontrol düzenlenir
  - javax.persistence.cache.retrieveMode
  - javax.persistence.cache.storeMode

Cache Retrieve Modları

- **CacheRetrieveMode.USE**:
  - Entity nesnenin mevcut ise önbellekten okunmasını söyler
  - Eğer cache'de değil ise DB'den okuma yapılır
  - EntityManager.refresh() de göz ardı edilir, çünkü bu işlemde her zaman DB'ye gidilmelidir
- **CacheRetrieveMode.BYPASS**:
  - Önbelleğin göz ardı edilip verinin doğrudan DB'den okunmasını sağlar

Cache Store Modları

- **CacheStoreMode.USE:**
  - DB'den okuma yapıldığında veya TX commit olduğunda verinin ön belleğe yazılmasını veya önbellekte güncellenmesini söyler
  - Halihazırda önbellekte bulunan bir veri ise herhangi bir şey yapmaz
- **CacheStoreMode.BYPASS**:
  - Verinin ön belleğe yazılmamasını sağlar, Önbelleğe hiç dokunulmaz

Cache Store Modları

- **CacheStoreMode.REFRESH**:
  - DB'den okuma olduğunda veya TX commit anında verinin ön belleğe yazılmasını söyler
  - Ancak USE'dan farklı olarak veritabanından okuma yapıldığında önbellekte daha evvel mevcut olan verinin güncellenmesini de sağlar

JPA ve Query Cache

- JPA 2'de **sorguların önbelleklenmesi** ile ilgili ise herhangi net bir kabiliyet ortaya konmamıştır
- Sorgularla ilgili olarak yine **ORM çözümüne spesifik özelliklerin kullanılması** söz konusudur

```
Query query = entityManager.createQuery("select o from Owner o");
query.setHint("org.hibernate.cacheable", true);
...
```

JPA İkincil Önbellek Erişim Kontrolüne Örnekler

```
EntityManager entityManager = emf.createEntityManager();
entityManager.setProperty("javax.persistence.cache.storeMode", "BYPASS");
```

```
Map<String, Object> properties = new HashMap<String, Object>();
props.put("javax.persistence.cache.retrieveMode", "BYPASS");
Owner owner = entityManager.find(Owner.class, 1L, properties);
```

```
CriteriaBuilder cb = emf.getCriteriaBuilder();
CriteriaQuery<Owner> criteriaQuery = cb.createQuery(Owner.class);

TypedQuery<Owner> query = entityManager.createQuery(criteriaQuery);
query.setHint("javax.persistence.cache.storeMode", "REFRESH");
```

Programatik JPA İkincil Önbellek Konfigürasyonu

- **EntityManagerFactory** yaratılırken programatik olarak da ikincil önbellek konfigürasyonu yapılabilir

```
Properties jpaProperties = new Properties();

jpaProperties.put("javax.persistence.sharedCache.mode","ENABLE_SELECTIVE");

EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("testjpa", jpaProperties);
```

Programatik İkincil Önbellek Erişimi

- **javax.persistence.Cache** arayüzü üzerinden de ikincil **önbellek kullanımı programatik olarak** da gerçekleştirilebilir

```
Cache cache = entityManagerFactory.getCache();

boolean contains = cache.contains(Owner.class, 1L);

cache.evict(Owner.class, 1L);

cache.evict(Owner.class);

cache.evictAll();
```

İletişim

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