JPA Entity Graph Kabiliyeti

harezmi bilişim çözümleri

Entity Graph Nedir?

- Normalde entity metadata içerisinde attribute ve ilişkilerin ne zaman yüklenecekleri **FetchType** ile tanımlanmaktadır
- Default durumda
  - @Basic, @ManyToOne, @OneToOne tanımlar **FetchType.EAGER**
  - @OneToMany, @ManyToMany, @ElementCollection tanımları **FetchType.LAZY**
- İstenirse metadata içerisinde bu tanımlar değiştirilebilir, ancak **çalışma zamanında senaryo özelinde değiştirilemez**

Entity Graph Nedir?

- JPA 2.1 ile gelen **entity graph kabiliyeti** ile çalışma zamanında entity attribute ve ilişkilerin FetchType tanımlarının **senaryoya göre değiştirilmesi** sağlanmıştır
- **@NamedEntityGraph** tanımı ile entity yüklenirken hangi attribute ve ilişkilerin yükleneceği tanımlanabilir
- Tanımlanan bu **entity graph çalışma zamanında aktive edilerek** entity'nin ilgili fetch tanımlarına göre yüklenmesi sağlanır

NamedEntityGraph

```
@NamedEntityGraphs({
     @NamedEntityGraph(
      name = "petsWithVisits",
      attributeNodes = {@NamedAttributeNode("name"),
                        @NamedAttributeNode("visits")})
})
@Entity
public class Pet {
    @Id
    private Long id;

    private String name;

    private Date birthDate;

    @OneToMany
    private List<Visit> visits = new ArrayList <Visit>();

    @ManyToOne
    private PetType petType;
}
```

NamedSubGraph

```
@NamedEntityGraph(
name = "ownersWithPetsAndVisits",
attributeNodes=@NamedAttributeNode(value="pets",
                                   subgraph="petsWithVisits")),
subGraphs = @NamedSubGraph(name = "petsWithVisits",
                           attributeNodes=@NamedAttributeNode("visits"))
)
@Entity
public class Owner {
    @Id
    private Long id;
    @OneToMany
    private Set<Pet> pets = new HashSet <Pet>();
}

@Entity
public class Pet {
    @Id
    private Long id;
    @OneToMany
    private List<Visit> visits = new ArrayList <Visit>();
}
```

Entity Graph Kullanımı - Fetch

```
EntityGraph entityGraph = entityManager
        .getEntityGraph("ownersWithPetsAndVisits");

Properties props = new Properties();
props.put("javax.persistence.fetchgraph",entityGraph);

Owner owner = entityManager.find(Owner.class,1L,props);
```

JPA spec'e göre sadece EntityGraph içerisinde tanımlanmış alanları ve ilişkileri getirmeli, diğer ilişkiler lazy gelmelidir

Ancak Hibernate gerçekleştirimi metadata içerisinde EAGER tanımlanmış alanları ve ilişkileri de getirmektedir

Entity Graph Kullanımı - Load

```
EntityGraph entityGraph = entityManager
        .getEntityGraph("ownersWithPetsAndVisits");


Query query = entityManager
                  .createQuery("from Owner");
List<Owner> owners = query
   .setHint("javax.persistence.loadgraph",entityGraph)
   .getResultList();
```

JPA spec'e göre EntityGraph içerisinde yer alan alanların yanı sıra entity içerisindeki attribute ve ilişkilerin metadata içerisindeki tanımları da dikkate alınır

loadgraph'ın fetchgraph'dan farkı, loadgraph'ın Entity içerisinde mapping metadata ile tanımlı fetch plan davranışını değiştirmesidir

Programatik Entity Graph Oluşturma

- EntityManager.**createEntityGraph**() ile çalışma zamanında **dinamik olarak** da entity graph yaratılabilir

```
EntityGraph entityGraph =
            entityManager.createEntityGraph(Pet.class);

entityGraph.addAttributeNodes("name", "visits");
```

İletişim

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