Entity İlişkileri

harezmi bilişim çözümleri

Entity İlişki Türleri

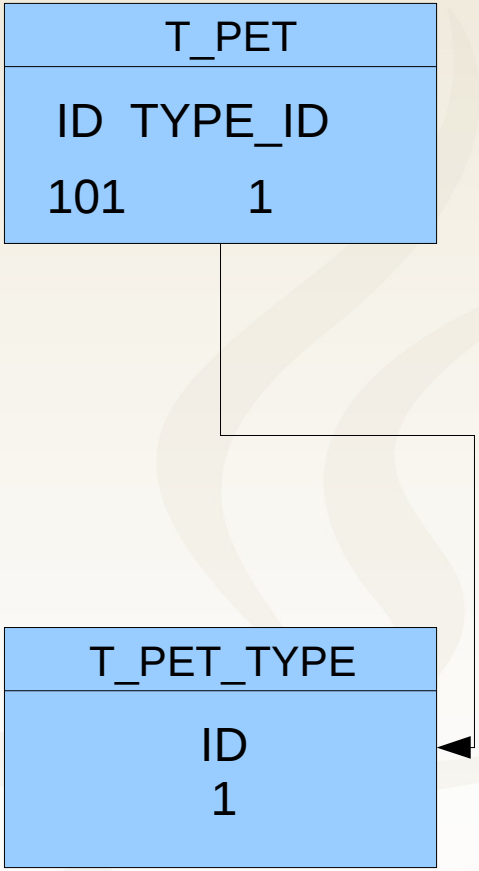
- Entity'ler arasındaki ilişkiler farklı biçimlerde kategorize edilerek incelenebilir
  - Multiplicity durumuna göre: **tekli** (1:1,M:1) veya **çoklu** (1:M,N:M) ilişkiler
  - Çoklu ilişkilerde kullanılan aggregate tipe göre: **list**, **set**, **map**, **bag**
  - İlişkinin yönüne göre: **tek yönlü** (unidirectional), **çift yönlü** (bidirectional)
  - İlişkiyi tutmak için ayrı bir tablo kullanılıp kullanılmamasına göre: **join column**, **join table**

M:1 İlişkiler

```
@Entity
public class Pet {

    @ManyToOne
    @JoinColumn(name = "TYPE_ID")
    private PetType petType;

}
```

```
@Entity
public class PetType {

}
```

@JoinColumn belirtilmez ise
Join-column ismi petType_id
şeklinde olacaktır

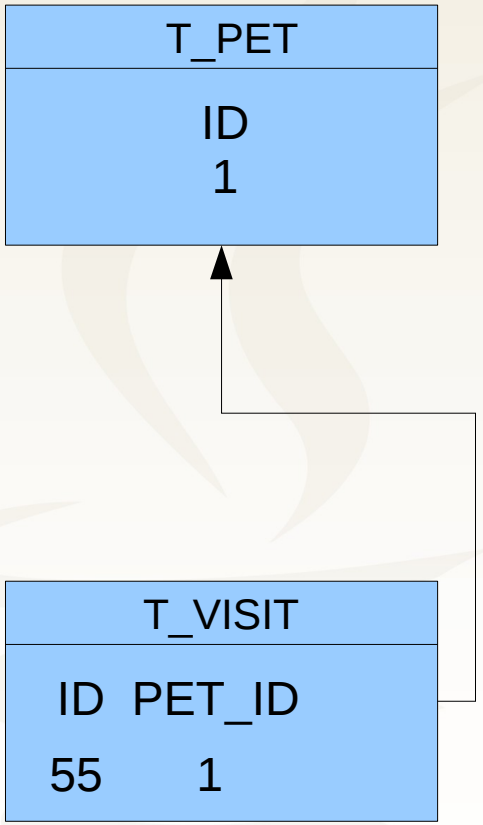


1:M – Tek Yönlü Set

```
@Entity
public class Pet {

   @OneToMany
   @JoinColumn(name = "PET_ID")
   private Set<Visit> visits =
                        new HashSet <Visit>();
}
```

Collection ilişkilerinde hedef entity tipi generic tip bilgisinden tespit edilmektedir

Collection tipi her zaman interface veya abstract type olmalıdır. Hibernate runtime da entity nesneyi load ederken ilişkili entity nesneleri kendi persistent collection tipinde bir nesne içerisinde yönetecektir




```
@Entity
public class Visit {

}
```

@JoinColumn belirtilmez ise ilişki default join-table üzerinden yönetilecektir. Join tablonun ismi pet_visit olacaktır (pet_id,visits_id şeklinde FK'lar olacaktır)

1:M – Tek Yönlü List

```
@Entity
public class Owner {

    @OneToMany
    @JoinColumn(name = "OWNER_ID")

    @OrderColumn(name = "PET_POSITION")

    private List<Pet> pets = new ArrayList<Pet>();

}
```

Pet'lerin kendi aralarındaki sırasını tutacak sütunu tanımlar. Bu sütun sayesinde listedeki Pet nesnelerinin sırası belirlenir

Name belirtilmez ise order-column ismi pets_order olacaktır

```
@Entity
public class Pet {

}
```

PET TABLOSU

ID	OWNER_ID	PET_POSITION	NAME
1	1	0	Foo
2	1	1	Bar
3	2	0	Bat
4	1	2	Baz

1:M – Tek Yönlü Bag (List ile)

@OrderColumn anotasyonu mevcut olmadığı için ilişki bir list değil, bag olarak davranacaktır

```
@Entity
public class Owner {

  @OneToMany
  @JoinColumn(name = "OWNER_ID")
  private List<Pet> pets = new ArrayList <Pet>();

}
```

Java Collections API'de Bag tipi yoktur. Bag olarak List veya Collection tipleri kullanılmaktadır

```
@Entity
public class Pet {

}
```

1:M – Tek Yönlü Bag (Collection ile)

@OrderColumn anotasyonu mevcut olmadığı için ilişki bir list değil, bag olarak davranacaktır

```
@Entity
public class Owner {

  @OneToMany
  @JoinColumn(name = "OWNER_ID")
  private Collection<Pet> pets = new ArrayList <Pet>();

}
```

Java Collections API'de Bag tipi yoktur. Bag olarak List veya Collection tipleri kullanılmaktadır

```
@Entity
public class Pet {

}
```

1:M – Tek Yönlü Map

```
@Entity
public class Owner {

  @OneToMany
  @JoinColumn(name = "OWNER_ID")
  @MapKey(name = "name")
  private Map<String,Pet> pets = new HashMap <String,Pet>();

}
```

Name attribute'una verilen değer Pet sınıfı içerisindeki persistent property'lerden birisinin ismidir. Sütun ismi değildir. Verilmez ise default id property'sidir.

@MapKey annotasyonu **sadece Entity**'ler için kullanılabilir

```
@Entity
public class Pet {

}
```

1:M – Çift Yönlü Set

mappedBy attribute çift yönlü ilişkide, ilişkiyi yöneten tarafın kim olduğunu tanımlar

```
public class Owner {

    @OneToMany(mappedBy = "owner")
    private Set<Pet> pets = new HashSet<Pet>();

}
```

mappedBy attribute olmadığı takdirde aynı FK'yı güncellemeye çalışan **iki farklı SQL ifadesi** söz konusu olurdu

JoinColumn ve **JoinTable** annotasyonları mappedBy ile işaret edilen tarafta tanımlanmalıdır

```
public class Pet {

    @ManyToOne
    @JoinColumn(name = "OWNER_ID")
    private Owner owner;

}
```

1:M – Çift Yönlü Set
(İlişkiyi Yöneten Tarafın Set Olması İstenirse)

```
public class Owner {

    @OneToMany
    @JoinColumn(name = "OWNER_ID", nullable=false)
    private Set<Pet> pets = new HashSet<Pet>();

}
```

Bu durumda iki tarafta da aynı join column tanımı yapılıp M:1 tarafından join column **insertable ve updateable false** yapılır. Hibernate pet nesnesinin owner property'si üzerinde yapılan işlemleri gözardı edecektir

```
public class Pet {

    @ManyToOne
    @JoinColumn(name = "OWNER_ID", updatable=false,
insertable = false, nullable=false)
    private Owner owner;

}
```

1:M – Çift Yönlü List

```
@Entity
public class Owner {

    @OneToMany
    @JoinColumn(name = "OWNER_ID", nullable=false)
    @OrderColumn(name = "PET_POSITION")
    private List<Pet> pets = new ArrayList<Pet>();

}


@Entity
public class Pet {

    @ManyToOne
    @JoinColumn(name = "OWNER_ID", updatable=false,
insertable = false, nullable=false)
    private Owner owner;

}
```

List ilişkileri yöneten tarafın her zaman için tekil taraf olması gerekir, çünkü collection içerisindeki entity nesnelerinin sıralarının belirlenmesi gerekir. Bu nedenle tekil tarafta **mappedBy** kullanılamaz.

M:1 ilişki **insertable** ve **updateable false** yapılarak FK'yı owner tarafındaki ilişkinin yönetmesi sağlanır

1:M – Çift Yönlü Bag

```
@Entity
public class Owner {

    @OneToMany(mappedBy = "owner")
    private Collection<Pet> pets = new ArrayList<Pet>();

}
```

```
@Entity
public class Pet {

    @ManyToOne
    @JoinColumn(name = "OWNER_ID")
    private Owner owner;

}
```

1:M – Çift Yönlü Map

```
@Entity
public class Owner {

   @OneToMany(mappedBy = "owner")
   @MapKey(name = "name")
   private Map<String,Pet> pets = new HashMap <String,Pet>();

}

@Entity
public class Pet {

    @ManyToOne
    @JoinColumn(name = "OWNER_ID")
    private Owner owner;

}
```

Name attribute'una verilen değer Pet sınıfı içerisindeki persistent property'lerden birisinin ismidir. Sütun ismi değildir. Verilmez ise default id property'sidir.

@MapKey annotasyonu **sadece Entity**'ler için kullanılabilir

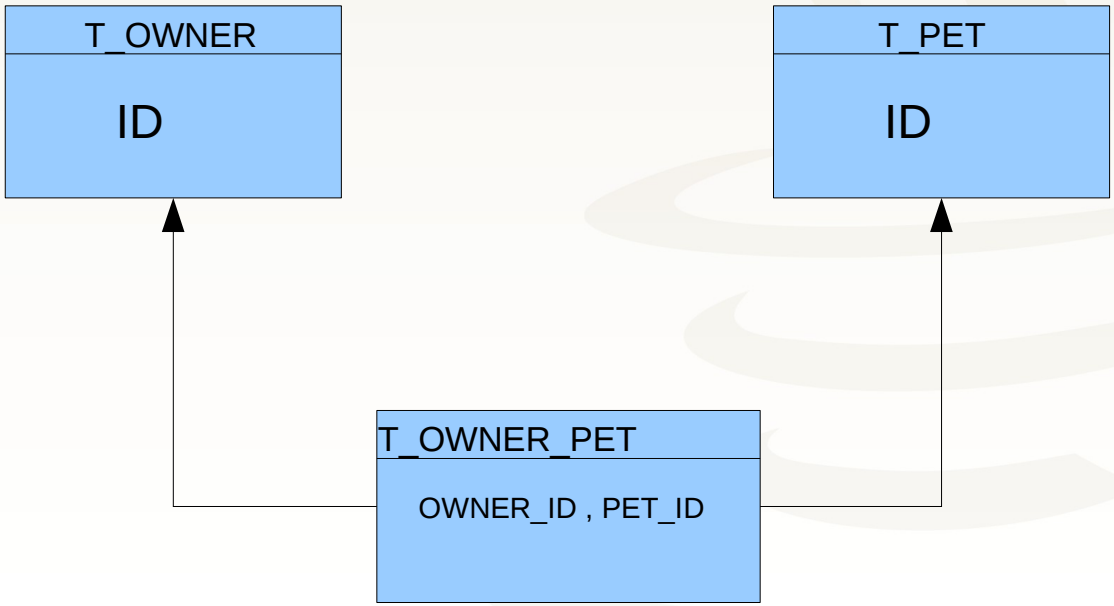
Join Tablo ile 1:M İlişkiler

joinColumns ilişkiyi yöneten tarafın ID'sine FK verir

inverseJoinColumns ise ilişkinin karşı tarafındaki Entity'nin ID'sine FK verir

```
@Entity
public class Owner {
    @OneToMany
    @JoinTable(
      name = "T_OWNER_PET",
      joinColumns = {@JoinColumn(name = "OWNER_ID")},
      inverseJoinColumns = {@JoinColumn(name = "PET_ID")})
    private Set<Pet> pets = new HashSet<Pet>();
}
```

```
@Entity
public class Pet {

}
```



Join Tablo ile 1:M İlişkiler

```
@Entity
public class Owner {
    @OneToMany(mappedBy = "owner")
    private Set<Pet> pets = new HashSet<Pet>();
}
```

@**JoinTable** annotasyonu her zaman ilişkiyi yöneten tarafta tanımlanır

```
@Entity
public class Pet {
     @ManyToOne
     @JoinTable(
      name = "T_OWNER_PET",
      joinColumns = {@JoinColumn(name = "PET_ID")},
      inverseJoinColumns = {@JoinColumn(name = "OWNER_ID")})
     private Owner owner;
}
```

Tek Yönlü N:M İlişkiler

```
@Entity
public class Vet {

   @ManyToMany

   @JoinTable(name = "T_VET_SPECIALTY",
   JoinColumns = {@JoinColumn(name="VET_ID")},
inverseJoinColumns = {@JoinColumn(name="SPECIALTY_ID")})

   private Set<Specialty> specialties=new HashSet<Specialty>();
}

@Entity
public class Specialty {

}
```

M:N ilişkilerde herzaman bir ara tablo olmak zorundadır. **joinColumns** ve **inverseJoinColumns** attribute tanımları daha önce anlatıldığı gibi ilişkiyi yöneten ve karşı tarafın ID'lerine FK vermek için kullanılır

Çift Yönlü M:N İlişkiler

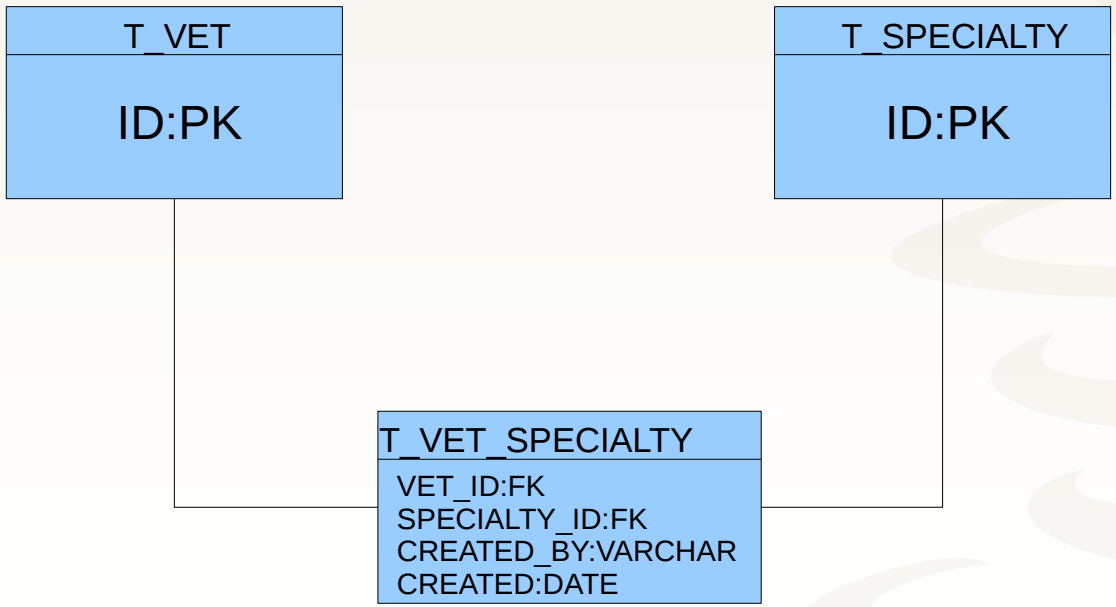
```
@Entity
public class Vet {

   @ManyToMany
   @JoinTable(name = "T_VET_SPECIALTY",
      joinColumns = { @JoinColumn(name = "VET_ID") },
      inverseJoinColumns = {@JoinColumn(name = "SPECIALTY_ID")})
   private Set<Specialty> specialties=new HashSet<Specialty>();

}

@Entity
public class Specialty {
   @ManyToMany(mappedBy = "specialties")
   private Set<Vet> vets =new HashSet<Vet>();

}
```

Çift yönlü M:N ilişki mappedBy ile tanımlanır.

@JoinTable yine ilişkiyi yöneten taraftadır.

M:N İlişkilerde Join Tablolara Sütun Eklenmesi

- M:N ilişkilerin tutulduğu ilişki tablosuna **ilave sütun eklenmesi söz konusu** olduğu vakit ara tabloyu da **ayrı bir entity** olarak ele almak gerekir
- Bu durumda M:N ilişki **iki tane çift yönlü 1:M ilişki**ye döner



Join Tablosunun Ara Entity Sınıf İle İfade Edilmesi

```
@Entity
public class Vet {

   @OneToMany(mappedBy = "vet")
   private Set<VetSpecialty> vetSpecialties = new
HashSet<VetSpecialty>();

}
```

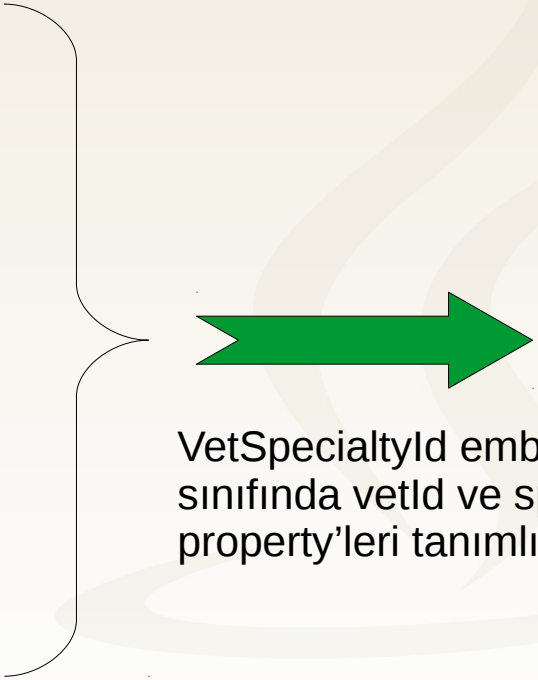
```
@Entity
public class Specialty {

   @OneToMany(mappedBy = "specialty")
   private Set<VetSpecialty> vetSpecialties = new
HashSet<VetSpecialty>();

}
```

M:N ilişkisini oluşturan entity sınıfları **join tabloya karşılık gelen entity sınıfla 1:M ilişki** kurar

Temel **avantajı çift yönlü navigasyon** sağlamasıdır

Join Tablosunun Ara Entity Sınıf İle İfade Edilmesi

```
@Entity
@Table(name = "T_VET_SPECIALTY")
public class VetSpecialty {

    @EmbeddedId
    private VetSpecialtyId id;

    @ManyToOne
    @MapsId(name = "vetId")
    private Vet vet;

    @ManyToOne
    @MapsId(name = "specialtyId")
    private Specialty specialty;

    private String createdBy;

    private Date createdOn;

}
```



VetSpecialtyId embeddable sınıfında vetId ve specialtyId property'leri tanımlı olmalıdır

Join Tablosunun Ara Entity Sınıf İle İfade Edilmesi

İkincil önbellek nedeni ile PK sınıfları Serializable olmak zorundadır

```
@Embeddable
public class VetSpecialtyId implements Serializable {

    @Column(name = "VET_ID")
    private Long vetId;

    @Column(name = "SPECIALTY_ID")
    private Long specialtyId;

    //...
}
```

1:1 İlişkiler

- 1:1 entity ilişkileri **iki farklı biçimde** oluşturulabilir
  - Foreign key üzerinden
  - Primary key üzerinden

Foreign Key Üzerinden 1:1 Unidirectional İlişkiler

```
@Entity
public class Owner {
    @Id
    @GeneratedValue
    private Long id;
}

@Entity
public class Address {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    @JoinColumn(name = "OWNER_ID")
    private Owner addressOwner;
}
```

PK dışında bir sütun üzerinden FK ilişkisi kurulur

Bir tablo diğer tablonun PK alanına işaret eden bir FK sütununa sahiptir

@JoinColumn belirtilmez ise default olarak Join-column addressOwner_id şeklinde olacaktır

Foreign Key Üzerinden 1:1 Bidirectional İlişkiler

```
@Entity
public class Owner {
    @Id @GeneratedValue
    private Long id;

    @OneToOne(mappedBy="owner")
    private Address address;
}

@Entity
public class Address {
    @Id
    @GeneratedValue
    private Long id;

    @OneToOne
    @JoinColumn(name = "OWNER_ID")
    private Owner owner;
}
```

Primary Key Üzerinden 1:1 Unidirectional İlişkiler

```
@Entity
public class Owner {
    @Id @GeneratedValue
    private Long id;
}

@Entity
public class Address {
    @Id
    private Long id;

    @OneToOne
    @MapsId
    private Owner owner;

}
```

İlişkili owner entity'sinin PK değerinin Address entity'sinin de PK değeri olmasını sağlar

JPA 2.0 da gelmiştir

Address entity'sinin PK generation stratejisinin assigned bırakılması gerekir

Primary Key Üzerinden 1:1 Bidirectional İlişkiler

JAVA Eğitimleri

```
@Entity
public class Owner {
    @Id @GeneratedValue
    private Long id;

    @OneToOne(mappedBy ="owner")
    private Address address;
}

@Entity
public class Address {
     @Id
     private Long id;

     @OneToOne
     @MapsId
     private Owner owner;

}
```

İletişim

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