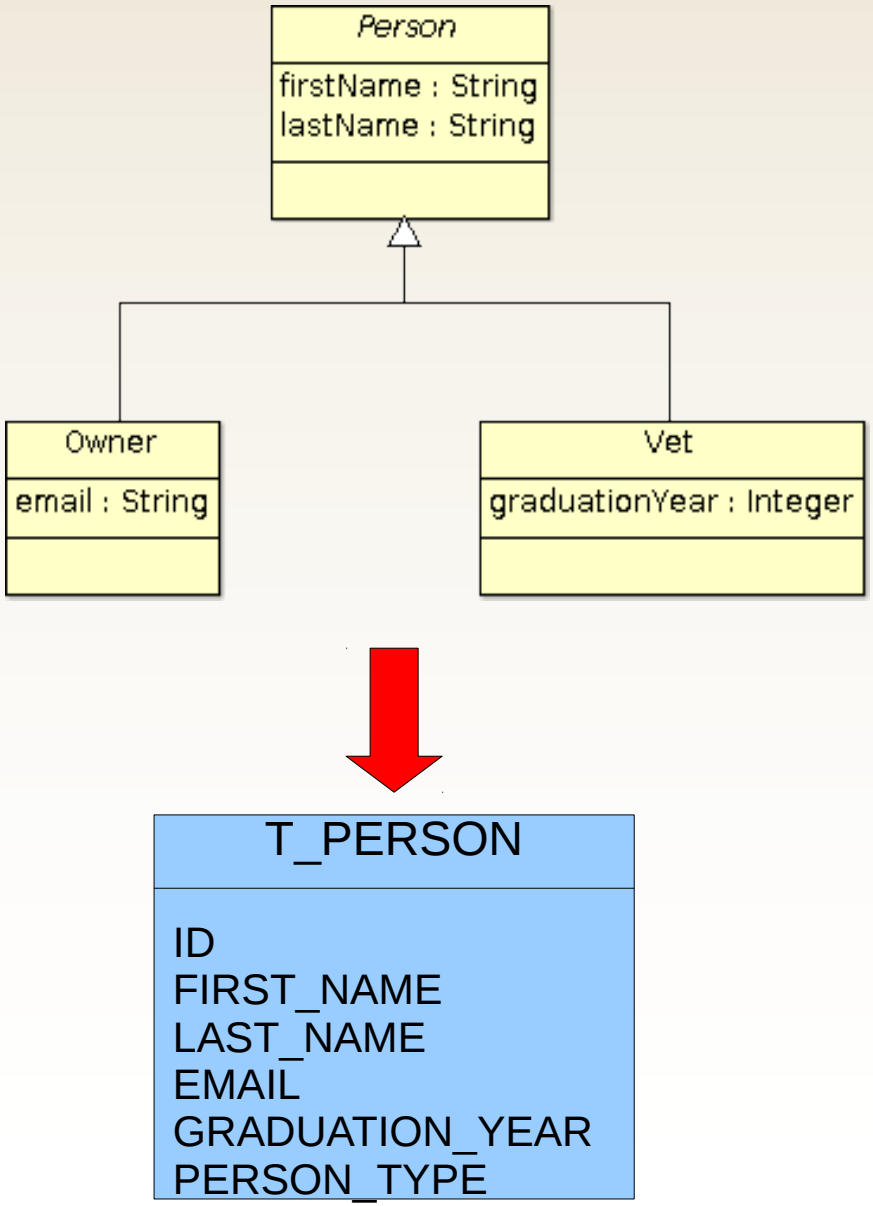
Inheritance Mapping Yöntemleri

harezmi bilişim çözümleri

Inheritance Yöntemleri

- Sınıflar arasındaki **inheritance hiyerarşisi**ni ilişkisel modelde üç farklı biçimde eşlemek mümkündür
  - **Bütün sınıf hiyerarşisi** için tek bir tablo (SINGLE_TABLE)
  - **Her sınıf (abstract ve concrete)** için ayrı bir tablo (JOINED_SUBCLASS)
  - **Her concrete sınıf** için ayrı bir tablo (TABLE_PER_CLASS)

Sınıf Hiyerarşisi İçin Tek Bir Tablo Yöntemi



- **Bütün sınıf hiyerarşisi tek bir tablo** ile eşlenir
- Bu tablo **bütün sınıfların sütunlarını** içerir
- Sınıfların tip bilgisi **discriminator** sütunu ile takip edilir
- Performans ve basitlik açısından avantajlıdır

Sınıf Hiyerarşisi İçin Tek Bir Tablo Yöntemi

```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "PERSON_TYPE",
    discriminatorType = DiscriminatorType.STRING)
public abstract class Person {
    @Id @GeneratedValue
    private Long id = null;
    //…
}
```

```java
@Entity
@DiscriminatorValue("O")
public class Owner extends Person {
    //…
}
```

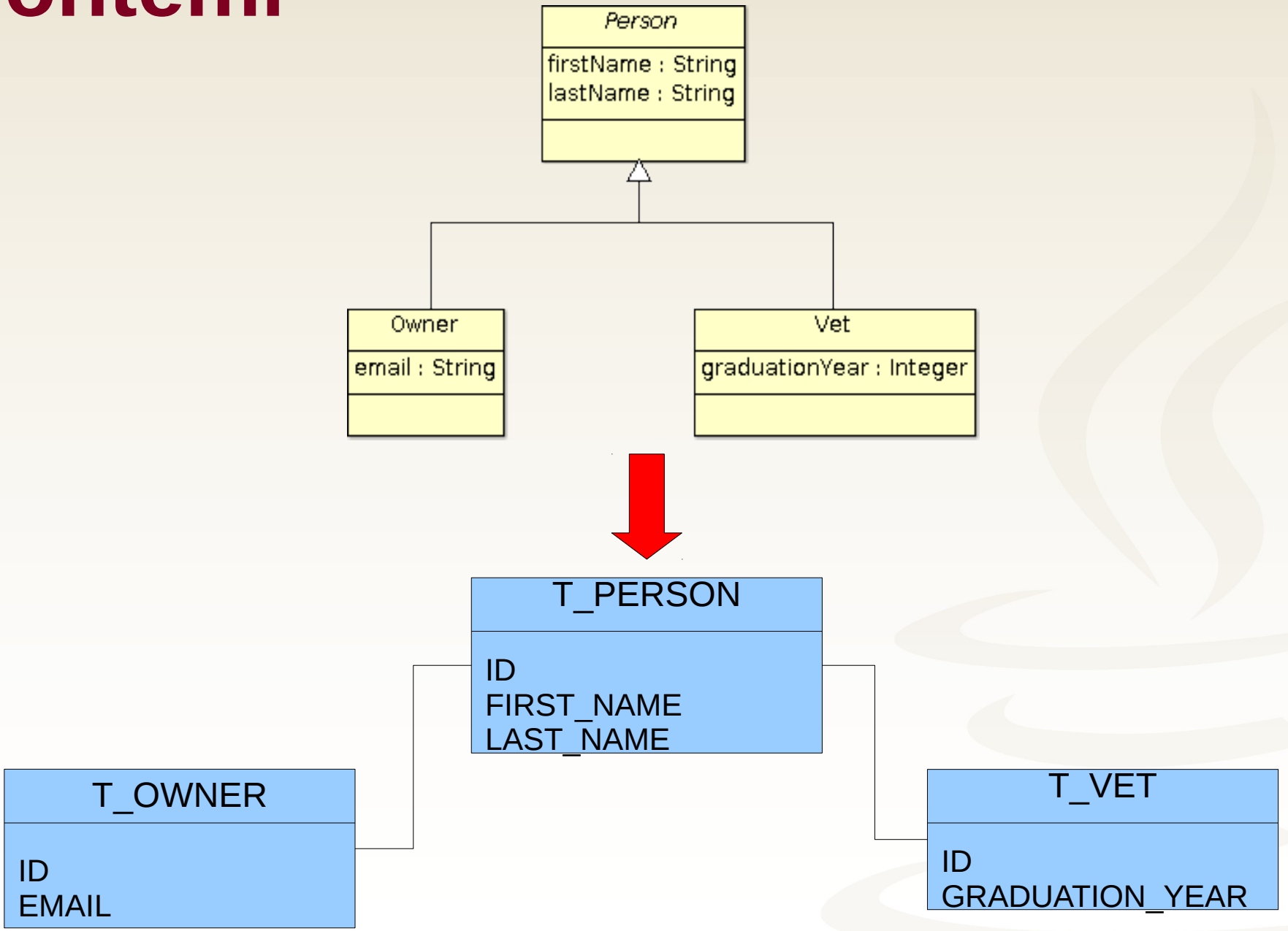
```java
@Entity
@DiscriminatorValue("V")
public class Vet extends Person {
    //…
}
```

Sınıf Hiyerarşisi İçin Tek Bir Tablo Yöntemi

JAVA Eğitimleri

- Hiyerarşideki **bütün concrete sınıflar bir discriminator değerine sahip** olurlar
- Explicit bir discriminator değeri belirtilmez ise:
  - Hibernate XML'de full sınıf ismi
  - Annotasyonlar'da entity ismi kullanılır
- Rapor sorguları **karmaşık join ve union işlemleri gerekmeden** yapılabilir
- Temel problemleri
  - alt sınıflar için tanımlanan sütunların NULLABLE olması ve normalizasyon eksikliği

Her Sınıf İçin Ayrı Bir Tablo Yöntemi



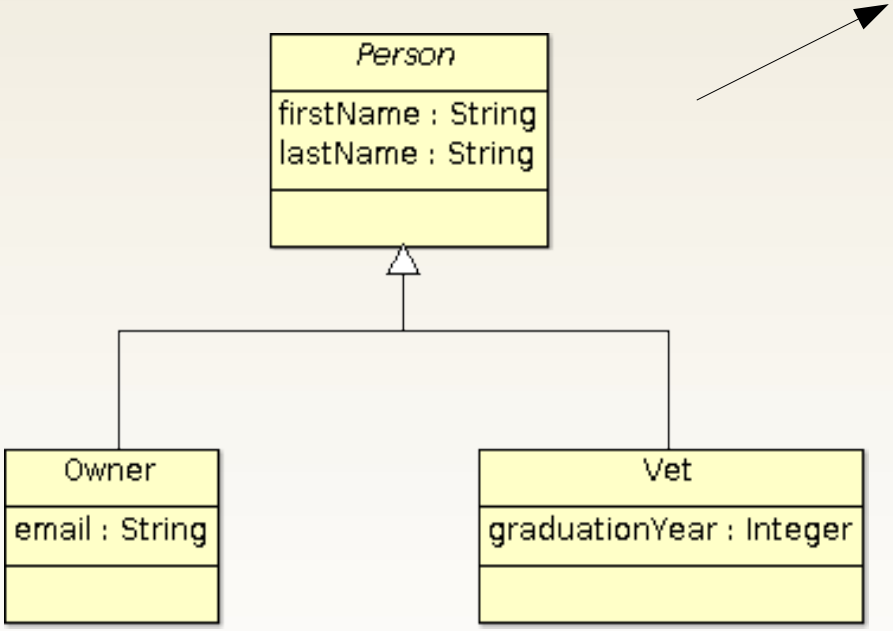
Her Sınıf İçin Ayrı Bir Tablo Yöntemi

- **Abstract sınıflar da dahil her sınıf için ayrı bir tablo** oluşturulur
- Her tablo **sadece o sınıfa özel alanları** içerir
- Alt sınıfın PKsı üst sınıfa FKdır
- Temel faydası şemanın **normalizasyon**udur
- Fakat rapor sorgularında zorluk çıkmaktadır
- Karmaşık sınıf hiyerarşilerinde de joinler **performans problemi**ne yol açabilir

Her Sınıf İçin Ayrı Bir Tablo Yöntemi

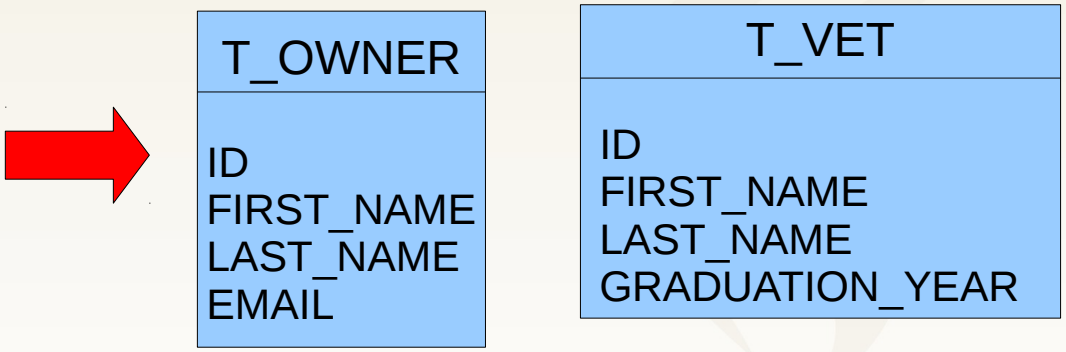
```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Person {
    @Id @GeneratedValue
    private Long id = null;
    //…
}

@Entity
public class Owner extends Person {
    //…
}

@Entity
@PrimaryKeyJoinColumn(name = "VET_ID")
public class Vet extends Person {
    //…
}
```

Her Concrete Sınıf İçin Ayrı Bir Tablo Yöntemi



Sadece concrete sınıflar için ayrı bir tablo yaratılır

Her bir tablo sınıf hiyerarşisindeki **bütün property'ler**i içerir



Her Concrete Sınıf İçin Ayrı Bir Tablo: Implicit Yöntem

JAVA Eğitimleri

```
@MappedSuperclass
public abstract class Person {
     @Id @GeneratedValue
     @Column(name = "ID")
     private Long id = null;
     //…
}
```

Inheritance eşlemesi için **explicit bir metadata kullanımı** söz konusu değildir

Abstract class'da **@MappedSuperclass** tanımlanması yeterlidir

```
@Entity
public class Vet extends Person {
     //…
}
```

Alt sınıflarda inheritance ile ilgili herhangi bir **tanım yapılmaz**. İstenirse üst sınıfta tanımlanmış attribute'ların sütun isimleri vs override edilebilir

```
@Entity
@AttributeOverride(name = "id", column = @Column(name =
"OWNER_ID"))
public class Owner extends Person {
     //…
}
```

Her Concrete Sınıf İçin Ayrı Bir Tablo: Explicit Yöntem

- Bu yaklaşım **TABLE_PER_CLASS** olarak da bilinir
- Entity sınıflarının PK değerleri **inheritance ilişkisinde yer alan tablolar genelinde unique** olmaktadır

Her Concrete Sınıf İçin Ayrı Bir Tablo: Explicit Yöntem

```java
@Entity
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class Person {

    @Id @GeneratedValue
    @Column(name = "ID")
    private Long id = null;
    //…

}

@Entity
@Table(name = "T_VET")
public class Vet extends Person {
    @Column(name = "GRADUATION_YEAR", nullable = false)
    private Integer graduationYear;
    //…
}
```

Her Concrete Sınıf İçin Ayrı Bir Tablo: Explicit Yöntem

- Üretilen SQL statement'ları çok kompleks olduğu için entity polymophism'e ihtiyaç duymayan yerlerde @**MappedSuperclass** ile **implicit yöntem** daha verimli bir tercih olacaktır

İletişim

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