



ENTITY RELATIONSHIPS

LẬP TRÌNH JAVA #4 (P3.2)

- Tự tạo CSDL PolyOE
- Lập trình làm việc với thực thể kết hợp 1-N
- Lập trình làm việc với thực thể kết hợp N-1



❑ Trong JPA có thể lập trình làm việc với CSDL sẵn có hoặc tự tạo CSDL khi chạy chương trình.

❑ Thêm thẻ sau vào persistence.xml và thay đổi giá trị thuộc tính value để thực hiện:

```
<property name="javax.persistence.schema-generation.database.action"  
value="drop-and-create" />
```

❑ Giá trị thuộc tính value:

- ❖ **Create**: Tạo CSDL nếu chưa tồn tại
- ❖ **Drop**: Xóa CSDL nếu tồn tại
- ❖ **Drop-and-create**: Xóa CSDL cũ và tạo CSDL mới
- ❖ **None**: (mặc định) có sẵn CSDL

```
<persistence-unit name="PolyOE">
    <properties>
        <property name="javax.persistence.jdbc.driver"
            value="com.microsoft.sqlserver.jdbc.SQLServerDriver" />
        <property name="javax.persistence.jdbc.url"
            value="jdbc:sqlserver://localhost;database=PolyOE" />
        <property name="javax.persistence.jdbc.user" value="sa" />
        <property name="javax.persistence.jdbc.password" value="*****" />
        <!-- create|drop|drop-and-create|none -->
        <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />
        <property name="hibernate.show_sql" value="true" />
        <property name="hibernate.format_sql" value="true" />
    </properties>
</persistence-unit>
```

- ❑ @Table(name, uniqueConstraints)
 - ❖ Thuộc tính uniqueConstraints tạo ra ràng buộc unique từ nhiều cột
- ❑ Ví dụ: Đoạn mã sau sẽ tạo bảng Favorites có ràng buộc duy nhất từ 2 cột UserId và Videoid:

```
@Entity
@Table(name = "Favorites", uniqueConstraints = {
    @UniqueConstraint(columnNames = {"UserId", "VideoId"})
})
public class Favorite {
```

❑ @Column(*name*, *length*, *nullable*, *unique*)

- ❖ **Name**: tên cột
- ❖ Dựa vào kiểu của field để sinh kiểu của column
- ❖ **Length**: kích thước (thường dung với chuỗi)
- ❖ **Nullable**: cho phép null hay không
- ❖ **Unique**: duy nhất hay không

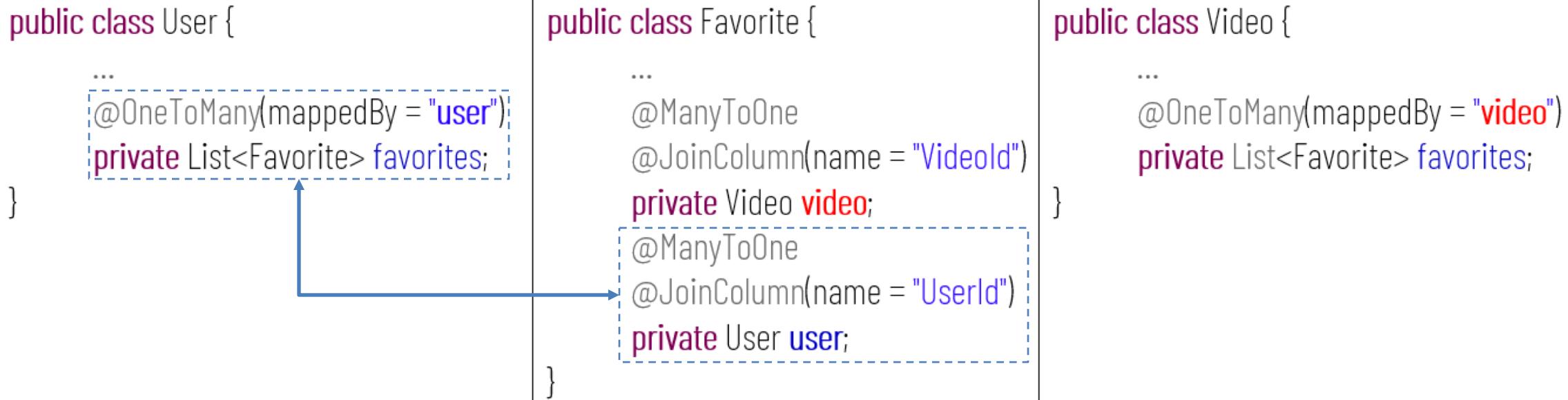
❑ Ví dụ: 2 đoạn mã SQL và Java sau đây là tương đương

- ❖ Email NVARCHAR (50) NOT NULL UNIQUE
- ❖ @Column(name=“Email”, length=50, nullable=false, unique=true)



DEMOSTATION

Tự tạo CSDL thông qua entity class hiện tại
Thêm các ràng buộc và tự tạo CSDL
So sánh sự khác biệt giữa 2 CSDL được tạo ra



❑ Thực thể kết hợp là các thực thể có liên quan đến một thực thể khác thông qua mối quan hệ đã được mapping.

- ❖ @OneToMany: Khi bạn có User thì có thể truy vấn các Favorite kết hợp với User đó bằng cách gọi getFavorites()
- ❖ @ManyToOne: Khi bạn có Favorite thì có thể truy vấn User kết hợp với Favorite đó bằng cách gọi getUser()

TRUY VẤN THỰC THỂ KẾT HỢP

```
public class User {  
    ...  
    @OneToMany(mappedBy = "user")  
    private List<Favorite> favorites;  
}
```

```
public class Favorite {  
    ...  
    @ManyToOne  
    @JoinColumn(name = "VideoId")  
    private Video video;  
    @ManyToOne  
    @JoinColumn(name = "UserId")  
    private User user;
```

```
public class Video {  
    ...  
    @OneToMany(mappedBy = "video")  
    private List<Favorite> favorites;
```

```
EntityManager em = XJPA.getEntityManager();  
User entity = em.find(User.class, "id");  
List<Favorite> favorites = entity.getFavorites();
```

```
EntityManager em = XJPA.getEntityManager();  
Favorite entity = em.find(Favorite.class, 123);  
User user = entity.getUser();
```



DEMOSTATION

Gửi email cho những User đã thích 1 video nào đó

Truy vấn các User có thích ít nhất 1 video.

Trong jpql thì ***o.favorites*** được hiểu là ***o.getFavorites()***

```
String jpql = "SELECT o FROM User o WHERE o.favorites IS NOT EMPTY";
TypedQuery<User> query = em.createQuery(jpql, User.class);
List<User> users = query.getResultList();
```

Truy vấn các Video yêu thích của TeoNV.

Trong jpql thì ***o.video*** được hiểu là ***o.getVideo()***
và ***o.user.id*** được hiểu là ***o.getUser().getId()***

```
String jpql = "SELECT o.video FROM Favorite o WHERE o.user.id='TeoNV'";
TypedQuery<Video> query = em.createQuery(jpql, Video.class);
List<Video> videos = query.getResultList();
```



DEMOSTATION

Truy vấn các User chưa thích bất kỳ một Video nào

- ❑ Thuộc tính fetch có trong @OneToMany(fetch) và @ManyToOne(fetch)
nó có 2 giá trị
 - ❖ FetchType.EAGER: Thực thể kết hợp được nạp cùng lúc với thực thể hiện tại.
 - ❖ FetchType.LAZY: Thực thể kết hợp chỉ được nạp khi gọi phương thức getter của thực thể hiện tại.
- ❑ @OneToMany(fetch): fetch mặc định là LAZY
- ❑ @ManyToOne(fetch): fetch mặc định là EAGER

```
@OneToMany(mappedBy = "user", fetch = FetchType.LAZY)  
private List<Favorite> favorites;
```

*Khi gọi getFavorites()
thì mới tiến hành truy vấn*

```
@ManyToOne(fetch = FetchType.EAGER)  
@JoinColumn(name = "UserId")  
private User user;
```

*User được truy vấn cùng với Favorite.
Khi gọi getUser() thì không truy vấn*



DEMOSTATION

*Khai báo thêm thuộc tính fetch vào @OneToMany và @ManyToOne
Theo chiến lược của GV và minh họa sự khác nhau của LAZY và EAGER*

- Tự tạo CSDL PolyOE
- Lập trình làm việc với thực thể kết hợp 1-N
- Lập trình làm việc với thực thể kết hợp N-1





Cảm ơn