JPA Mapping I

Sang Shin
www.JPassion.com
"Learn with JPassion!"

Agenda

- Entity Relationships
- Directionality
- Cardinality
- Inheritance (will be covered in “JPA Mapping II”)

Entity Relationships

Aspects of Entity Relationships

- Directionality
  - Uni-directional
  - Bi-directional
- Cardinality relationships
  - One to one
  - One to many
  - Many to many
- Inheritance relationship
  - Single-table
  - Joined-table

Entity Relationships: Java vs. Table

- Relationship between entities in Java code is normal object relationship
- Relationship among tables can be represented in two forms
  - Join table
  - Foreign key
- You can control how the object relationship between Java objects can be mapped into tables through JPA annotations

Directionality

Directionality & Navigation

- Directionality affects the navigation
- Uni-directional
  - *customer.getAddress()* is allowed but *address.getCustomer()* is not supported for one-to-one relationship
  - *customer.getOrders()* is allowed but *order.getCustomer()* is not supported for one-to-many relationship
  - *speaker.getEvents()* is allowed but event.*getSpeakers()* is not supported for many-to-many relationship
- Bi-directional
  - Navigation on both direction are supported

Directionality & Ownership

- Relationship has an ownership
  - Who owns the relationship affects the how tables are created
- Uni-directional
  - Ownership is implied
- Bi-directional
  - Ownership needs to be explicitly specified
  - Owner of the relationship, inverse-owner of the relationship

Cardinality Relationships

Cardinality Entity Relationships

- Cardinality relationships
  - @OneToOne: Customer – Address
  - @OneToMany, @ManyToOne: Customer - Orders
  - @ManyToMany: Speakers - Events
- One-to-Many and Many-to-Many relationships are represented in Java code through
  - Collection, Set, List and Map

Cardinality & Ownership

- @OneToOne bidirectional relationship
  - The owner is the side with the foreign key
- @OneToMany, @ManyToOne bidirectional relationshgip
  - The owner is always the “Many” side
- @ManyToMany bidirectional relationship

Cardinality Relationships: One to One

One to One Relationships

- Directionality
  - One-to-One Unidirectional
  - One-to-One Bidirectional
- Table model
  - Always Foreign key based (No join table)

Demo:

jpa1_0x_OneToOne_*

4321_jpa_mapping.zip

Cardinality Relationships: One to Many

One to Many Relationships

- Directionality
  - One-to-Many Unidirectional
  - One-to-Many Bidirectional
- Table model
  - Join Table (default) or
  - Foreign key column

Relationship Mappings – OneToMany

```
@Entity
public class Customer {
  @Id
  int id;
  ...

  @OneToMany(mappedBy="cust")
  List<Order> orders;
}
```

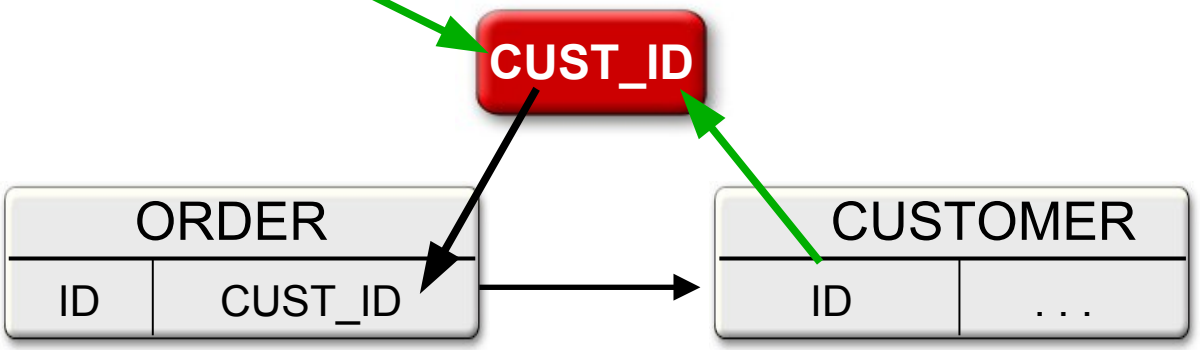
```
@Entity
public class Order {
  @Id
  int id;
  ...
  @ManyToOne
  Customer cust;
}
```

Relationship Mappings – ManyToOne

```
@Entity
public class Order {

    @Id
     int id;

    @ManyToOne
     Customer cust;
}
```



Automatically creates a CUST_ID field for mapping. Can be overridden via @JoinColumn (or @JoinColumns for composite foreign keys).

Examples: OneToMany & ManyToOne

Example 1

Customer Entity

```
@OneToMany(mappedBy="cust")
public Set<Order> orders;
```

Order Entity

```
@ManyToOne public Customer cust;
```

Example 2

Customer Entity

```
@OneToMany(mappedBy="cust", cascade=ALL)
public Set<Order> orders;
```

Order Entity

```
@ManyToOne
@JoinColumn(name="my_cust_id")
public Customer cust;
```

Relationship Mappings – ManyToMany

```
@Entity
public class Customer {
  @Id
  int id;
  ...
  @ManyToMany
  Collection<Phone> phones;
}
```

```
@Entity
public class Phone {
  @Id
  int id;
  ...
@ManyToMany(mappedBy="phones")
Collection<Customer> custs;
}
```

Join table name is made up of the 2 entities. Field name is the name of the entity plus the name of the PK field.

Demo:

jpa1_0x_OneToMany_*

4321_jpa_mapping.zip

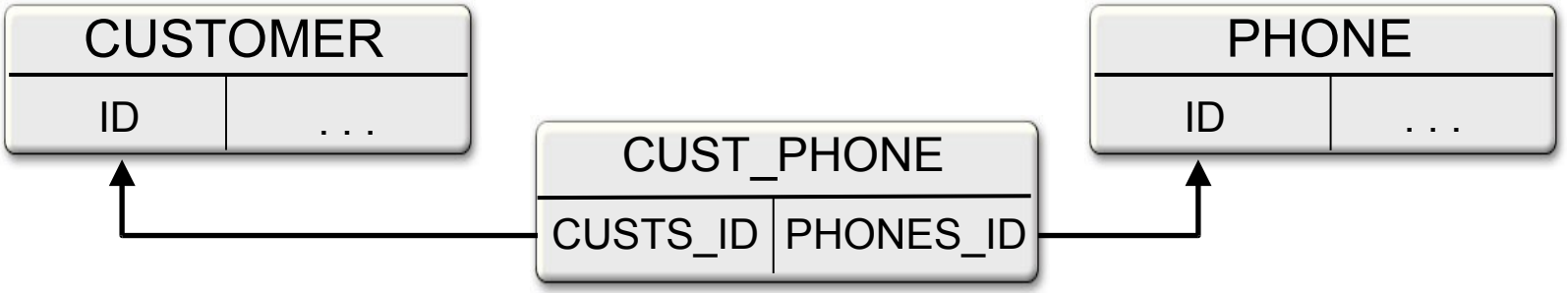
Cardinality Relationships: Many to Many

Many to Many Relationships

- Directionality
  - Many-to-Many Unidirectional
  - Many-to-Many Bidirectional
- Table model
  - Always Join Table

Example – ManyToMany

```
@Entity(access=FIELD)
public class Customer {
  ...
  @ManyToMany
  @JoinTable(name="CUST_PHONE",
    joinColumns=@JoinColumn(name="CUSTS_ID"),
    inverseJoinColumns=@JoinColumn(name="PHONES_ID"))
  Collection<Phone> phones;
}
```



We are overriding the default OR mapping here.

Demo:

jpa1_0x_ManyToMany_*

4321_jpa_mapping.zip

Cascading

Cascading Behavior

- Cascading is used to propagate the effect of an operation to associated entities
- Cascading operations will work only when entities are associated to the persistence context
  - If a cascaded operation takes place on detached entity, IllegalArgumentException is thrown
- Cascade=PERSIST
- Cascade=REMOVE
- Cascade=MERGE
- Cascade=REFRESH
- Cascade=ALL

Thank you!

Check JavaPassion.com Codecamps!
http://www.javapassion.com/codecamps