



CS5220 Advanced Topics in Web Programming

Object-Relational Mapping with Hibernate and JPA (II)

Chengyu Sun
California State University, Los Angeles



Schema Generation

- ◆ `SchemaExport` in the Hibernate example
- ◆ Standardized in JPA 2.1
- ◆ Generate DDL from annotated Java classes
 - To file/console/database
 - More in Chapter 9.4 of JPA 2.1 Specification

Basic Object-Relational Mapping

- ◆ Class-level annotations
 - @Entity and @Table
- ◆ Id field
 - @Id and @GeneratedValue
- ◆ Fields of simple types
 - @Basic (can be omitted) and @Column
- ◆ Fields of class types
 - @ManyToOne, @OneToMany, @ManyToMany, and @OneToOne

Advanced ORM

- ◆ Embedded class
- ◆ Collection
- ◆ Inheritance

Embedded Class

```
public class Address {  
    String street;  
    String city;  
    String state;  
    String zip;  
}
```

```
public class User {  
    Integer id;  
    String username;  
    String password;  
    Address address;  
}
```



users

id	...	street	city	state	zip	...
----	-----	--------	------	-------	-----	-----

Mapping Embedded Class

```
@Embeddable
public class Address {
    String street;
    String city;
    String state;
    String zip;
}
```

```
@Entity
public class User {
    @Id
    Integer id;
    String username;
    String password;
    @Embedded
    Address address;
}
```

Collection of Simple Types

```
public class Customer {  
  
    Integer id;  
  
    String name;  
    String address;  
  
    Set<String> phones;  
  
}
```

Mapping Element Collection

```
@ElementCollection  
Set<String> phones;
```



customers

	id

Customer_phones

Customer_id	phones



Customize Collection Table

```
@ElementCollection
@CollectionTable(
    name = "customer_phones",
    joinColumns=@JoinColumn(name = "customer_id")
)
@Column(name="phone")
Set<String> phones;
```

Set vs. List

- ◆ Elements in a list are *ordered*
- ◆ Rows in a relational table are *not ordered*

```
public class Customer {  
  
    Integer id;  
  
    String name;  
    String address;  
  
    List<String> phones;  
  
}
```

Ensure Order in List

Use the natural order (e.g. alphabetic order) of an existing column

customer_id	phone
1	(323) 343-6789
1	(626) 353-4567
1	(323) 343-1234
2	(323) 343-1111
2	(626) 353-2222



Customer 1's list of phones:

(323) 343-1234
(323) 343-6789
(626) 353-4567

Use an additional column to keep the order

customer_id	phone	phone_order
1	(323) 343-6789	2
1	(626) 353-4567	1
1	(323) 343-1234	0
2	(323) 343-1111	0
2	(626) 353-2222	1



Customer 1's list of phones:

(323) 343-1234
(626) 353-4567
(323) 343-6789

List of Simple Types ...

- ◆ If keeping element order is not important:

```
@ElementCollection  
List<String> phones;
```



The order of the elements saved to the database may not be the same as when they are read from the database.

... List of Simple Types

◆ Order by property

- `@OrderBy("<property_name> ASC|DESC")`
- Simple types do not have properties

```
@ElementCollection  
@OrderBy("asc")  
List<String> phones;
```

◆ Order by a separate column

```
@ElementCollection  
@OrderColumn(name = "phone_order")  
List<String> phones;
```

Issues Related to Collections of Class Types

- ◆ Unidirectional vs. Bidirectional Association
- ◆ Set vs. List
- ◆ Lazy Loading and Cascading Behaviors

Association Example

- ◆ A customer may own multiple accounts
- ◆ An account only has one owner

Unidirectional Association #1

```
public class Account {  
  
    Integer id;  
  
    Double balance;  
    Date createdOn;  
  
    @ManyToOne  
    Customer owner;  
  
}
```

```
public class Customer {  
  
    Integer id;  
  
    String name;  
    String address;  
    Set<String> phones;  
  
}
```


Bidirectional Association

```
public class Account {
```

```
    Integer id;
```

```
    Double balance;
```

```
    Date createdOn;
```

```
    @ManyToOne
```

```
    Customer owner;
```

```
}
```

```
public class Customer {
```

```
    Integer id;
```

```
    String name;
```

```
    String address;
```

```
    Set<String> phones;
```

```
    @OneToMany(mappedBy="owner")
```

```
    Set<Account> accounts;
```

```
}
```

↑
property

Unidirectional Association #2

```
public class Account {  
  
    Integer id;  
  
    Double balance;  
    Date createdOn;  
  
}
```

*Do not do this as it
will be handled as a
many-to-many
association which is
less efficient*

```
public class Customer {  
  
    Integer id;  
  
    String name;  
    String address;  
    Set<String> phones;  
  
    @OneToMany  
    Set<Account> accounts;  
  
}
```

Using List ...

```
public class Customer {  
  
    Integer id;  
  
    String name;  
    String address;  
    Set<String> phones;  
  
    @OneToMany(mappedBy="owner")  
    @OrderBy( "createdOn asc" )  
    List<Account> accounts;  
  
}
```

... Using List

- ◆ Avoid using list with @OrderColumn in a bidirectional association
 - See Section 2.4.6.2.1 in [Hibernate Annotation Reference Guide](#)
 - See AnswerSheet and AnswerSection in CSNS2

Many-To-Many Example

- ◆ A customer may own multiple accounts
- ◆ An account may have multiple owners

Mapping Many-To-Many

```
public class Account {  
  
    Integer id;  
  
    Double balance;  
    Date createdOn;  
  
    @ManyToMany  
    Set<Customer> owners;  
  
}
```

```
public class Customer {  
  
    Integer id;  
  
    String name;  
    String address;  
    Set<String> phones;  
  
    @ManyToMany(mappedBy="owners")  
    Set<Account> accounts;  
  
}
```

↑
property

Customize Join Table

```
@ManyToMany
@JoinTable(
    name = "account_owners",
    joinColumns=@JoinColumn(name = "account_id"),
    inverseJoinColumns=@JoinColumn(name="owner_id")
)
Set<Customer> owners;
```

Lazy Loading

```
entityManager = entityManagerFactory  
    .createEntityManager();
```

```
Customer c = entityManager.find( Customer.class, 1 );  
// c is loaded, but c's accounts are not
```

```
Integer accountId = c.getAccounts().get(0).getId();  
// c's accounts are loaded from database when  
// c.getAccounts() is called → Lazy Loading
```

```
entityManager.close();  
// Lazy loading can only happen inside the same entity manager  
// (a.k.a. persistent context)
```


Cascading Behavior

- ◆ Whether an operation on the parent object (e.g. Customer) should be applied to the children objects in a collection (e.g. List<Account>)

```
Customer c = new Customer("cysun");  
Account a1 = new Account();  
Account a2 = new Account();  
c.getAccounts().add( a1 );  
c.getAccounts().add( a2 );
```

```
c = entityManager.merge(c); // will a1 and a2 be saved as well?
```

```
... ..
```

```
entityManager.remove(c); // will a1 and a2 be deleted from db??
```

Cascading Types in JPA

◆ <https://docs.oracle.com/javaee/7/api/java/persistence/CascadeType.html>

CascadeType Examples

```
@OneToMany(mappedBy="owner",  
            cascade=CascadeType.PERSIST)  
List<Account> accounts;
```

```
@OneToMany(mappedBy="owner",  
            cascade={CascadeType.PERSIST, CascadeType.MERGE})  
List<Account> accounts;
```

```
@OneToMany(mappedBy="owner",  
            cascade=CascadeType.ALL)  
List<Account> accounts;
```

Inheritance

```
public class CDAccount extends Account {  
    Integer term;  
}
```

Everything in One Table

accounts

id	account_type	balance	created_on	term
----	--------------	---------	------------	------



Discriminator column

Inheritance Type – SINGLE_TABLE

```
@Entity
@Table(name="accounts")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="account_type")
@DiscriminatorValue("CHECKING")
public class Account { ... }
```

```
@Entity
@DiscrimnatorValue("CD")
public class CDAccount extends Account { ... }
```

Table Per Subclass

accounts

id	balance	created_on
----	---------	------------



foreign key

cd_accounts

account_id	term
------------	------

Inheritance Type – JOINED

```
@Entity
@Table(name="accounts")
@Inheritance(strategy=InheritanceType.JOINED)
public class Account { ... }
```

```
@Entity
@Table(name="cd_accounts")
public class CDAccount { ... }
```


Table Per Concrete Class

accounts

id	balance	created_on
----	---------	------------

cd_accounts

id	balance	created_on	term
----	---------	------------	------

Inheritance Type – TABLE_PER_CLASS

```
@Entity  
@Table(name="accounts")  
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)  
public class Account { ... }
```

```
@Entity  
@Table(name="cd_accounts")  
public class CDAccount { ... }
```

Tips for O/R Mapping

◆ Understand OO design

- Make sure the application design is object-oriented

◆ Understand relational design

- Know what the database schema should look like before doing the mapping

More HQL Examples (I) ...

```
class User {  
  
    Integer id;  
    String username;  
    ...  
}
```

users

id	username
1	cysun
2	vcrespi

```
class Section {  
  
    Integer id;  
    User instructor;  
    ...  
}
```

sections

id	instructor_id
1	1
2	1
3	2

... More HQL Examples (I)

- ◆ Query: find all the sections taught by the user "cysun".
 - *SQL??*
 - *HQL??*

More HQL Examples (II) ...

```
class User {  
  
    Integer id;  
    String username;  
    ...  
}
```

```
class Section {  
  
    Integer id;  
    Set<User> instructors;  
    ...  
}
```

users

id	username
1	cysun
2	vcrespi

sections

id
1
2
3

instructors

section_id	instructor_id
1	1
2	1
2	2

... More HQL Examples (II)

- ◆ Query: find all the sections for which "cysun" is one of the instructors
 - *SQL??*
 - *HQL??*

See `SectionDaoImpl` in `CSNS2` for more HQL join examples

Readings

◆ Hibernate ORM User Guide

- [Chapter 2 Domain Model](#)
- [Chapter 15 HQL and JPQL](#)
- [Chapter 24 Mapping Annotations](#)