Module 06: Complex Mapping

CS544: Enterprise Architecture

Complex Mappings

- In this module we will cover:
  - Secondary tables – allow a class to be mapped to multiple tables
  - Embedded classes – allow multiple classes to be mapped to a single table
  - Composite keys – can be made using embedded classes
  - Immutable entities – Hibernate can optimize for entities that never change

Complex Mapping

SECONDARY TABLES

Secondary Tables

- Secondary tables can be used anywhere to move properties into separate table(s)
  - To do so, the property has to specify the table
  - Secondary tables can even be used in combination with the Single table inheritance strategy

```
@Entity
@DiscriminatorValue("savings")
@SecondaryTable(
  name="SavingsAccount",
  pkJoinColumns=@PrimaryKeyJoinColumn(name="number")
)
public class SavingsAccount extends Account {
  @Column(table="SavingsAccount")
  private double APY;

  ...
```

Secondary table used in an inheritance hierarchy

Property specifies that it should be on the SavingsAccount table

Secondary Table

@SecondaryTables can specify multiple @SecondaryTable

pkJoinColumns can be used to specify a multi column join

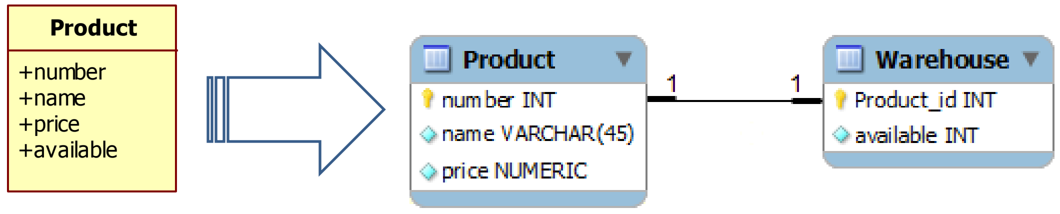
```
@Entity
@SecondaryTables(
  @SecondaryTable(name="warehouse", pkJoinColumns = {
    @PrimaryKeyJoinColumn(name="product_id", referencedColumnName="number")
  }
))
public class Product {
  @Id
  @GeneratedValue
  private int number;
  private String name;
  private BigDecimal price;
  @Column(table="warehouse")
  private boolean available;

  ...
```

JoinColumn name can differ from the referenced column

Properties need to specify the secondary table to be on it

All you really need is @SecondaryTable and a name, the rest is optional

```
@Entity
@SecondaryTable(name="warehouse")
public class Product {
  @Id
  @GeneratedValue
  private int number;
  private String name;
  private BigDecimal price;
  @Column(table = "warehouse")
  private int available;

  ...
```

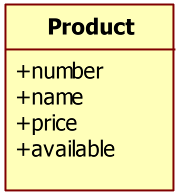


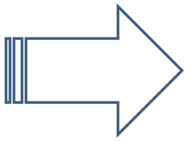
XML

```
<hibernate-mapping package="join_tables">
  <class name="Product">
    <id name="number">
      <generator class="native" />
    </id>
    <property name="name" />
    <property name="price" />

    <join table="warehouse">
      <key column="product_id" />
      <property name="available" />
    </join>
  </class>
</hibernate-mapping>
```

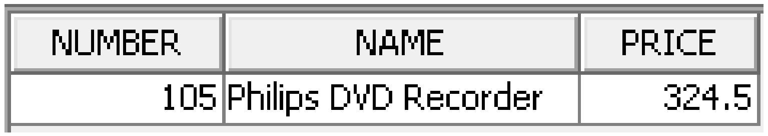
<join> tag to specify the table

Requires <key> to specify the pk join column



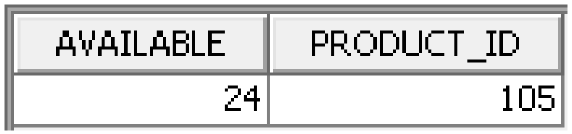


Product Table



NUMBER	NAME	PRICE
105	Philips DVD Recorder	324.5

Warehouse Table



AVAILABLE	PRODUCT_ID
24	105

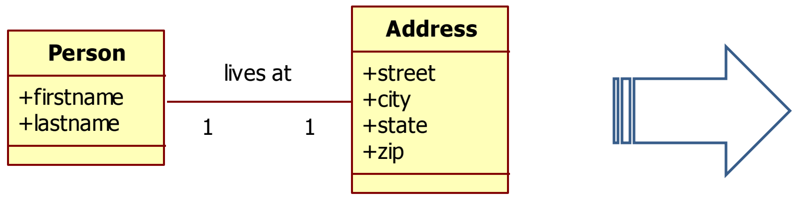
Complex Mapping

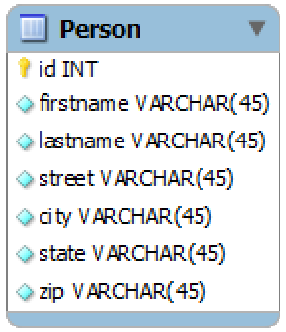
EMBEDDED CLASSES

Embedded Classes

- Combine multiple classes in a single table
- Especially useful for tight associations
- These classes are considered value classes rather than entity classes

Address is embedded inside the Person table





Person
id INT
firstname VARCHAR(45)
lastname VARCHAR(45)
street VARCHAR(45)
city VARCHAR(45)
state VARCHAR(45)
zip VARCHAR(45)

Embeddable

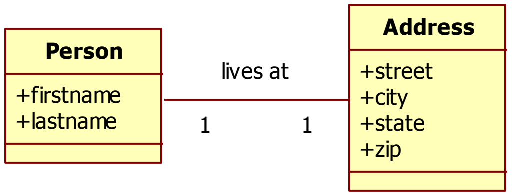
```
@Entity
public class Person {
   @Id
   @GeneratedValue
   private int id;
   private String firstname;
   private String lastname;

   @Embedded
   private Address address;

   ...
```

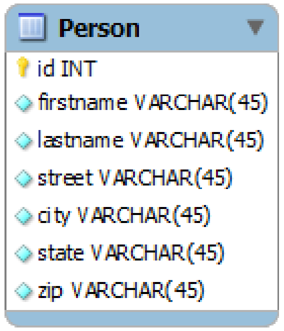
@Embedded annotation is used for embeddable objects

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

   ...
```

@Embeddable instead of @Entity

No @Id in embeddable



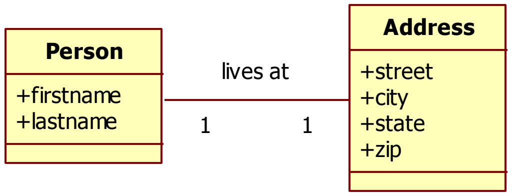


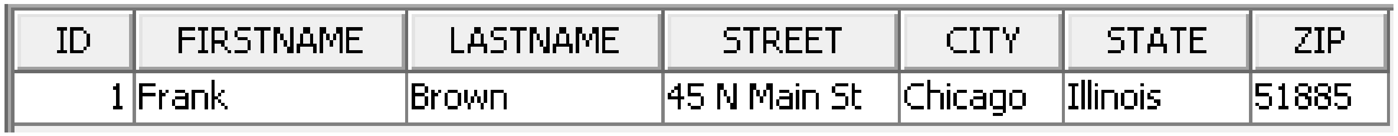
ID	FIRSTNAME	LASTNAME	STREET	CITY	STATE	ZIP
1	Frank	Brown	45 N Main St	Chicago	Illinois	51885

XML

```
<hibernate-mapping package="embedded">
  <class name="Person">
    <id name="id">
      <generator class="native" />
    </id>
    <property name="firstname" />
    <property name="lastname" />

    <component name="address" class="Address">
      <property name="street" />
      <property name="city" />
      <property name="state" />
      <property name="zip" />
    </component>
  </class>
</hibernate-mapping>
```

<component> tag indicates an embedded object



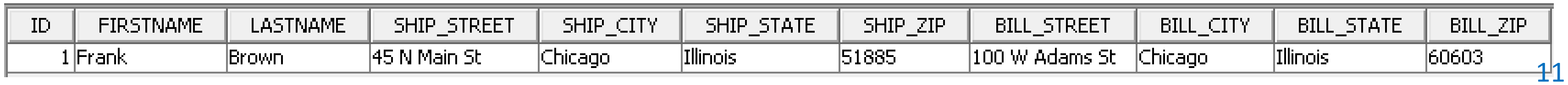


ID	FIRSTNAME	LASTNAME	STREET	CITY	STATE	ZIP
1	Frank	Brown	45 N Main St	Chicago	Illinois	51885

Multiple Embedded Addresses

```
@Entity
public class Customer {
  @Id
  @GeneratedValue
  private int id;
  private String firstname;
  private String lastname;

  @Embedded
  @AttributeOverrides( {
    @AttributeOverride(name="street", column=@Column(name="ship_street")),
    @AttributeOverride(name="city", column=@Column(name="ship_city")),
    @AttributeOverride(name="state", column=@Column(name="ship_state")),
    @AttributeOverride(name="zip", column=@Column(name="ship_zip"))
  })
  private Address shipping;

  @Embedded
  @AttributeOverrides( {
    @AttributeOverride(name="street", column=@Column(name="bill_street")),
    @AttributeOverride(name="city", column=@Column(name="bill_city")),
    @AttributeOverride(name="state", column=@Column(name="bill_state")),
    @AttributeOverride(name="zip", column=@Column(name="bill_zip"))
  })
  private Address billing;

...
```

Rename the column names for the embedded object using @AttributeOverrides

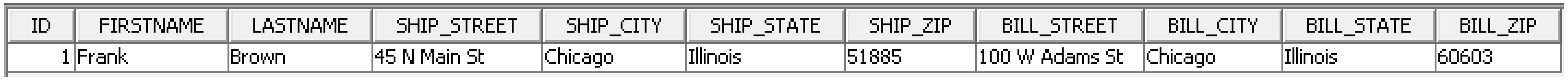


ID	FIRSTNAME	LASTNAME	SHIP_STREET	SHIP_CITY	SHIP_STATE	SHIP_ZIP	BILL_STREET	BILL_CITY	BILL_STATE	BILL_ZIP
1	Frank	Brown	45 N Main St	Chicago	Illinois	51885	100 W Adams St	Chicago	Illinois	60603

Multiple Addresses XML

```
<hibernate-mapping package="embedded">
  <class name="Customer">
    <id name="id">
      <generator class="native" />
    </id>
    <property name="firstname" />
    <property name="lastname" />

    <component name="shipping" class="Address">
      <property name="street" column="ship_street" />
      <property name="city" column="ship_city" />
      <property name="state" column="ship_state" />
      <property name="zip" column="ship_zip" />
    </component>

    <component name="billing" class="Address">
      <property name="street" column="bill_street" />
      <property name="city" column="bill_city" />
      <property name="state" column="bill_state" />
      <property name="zip" column="bill_zip" />
    </component>
  </class>
</hibernate-mapping>
```

You can specify the column name using the column attribute on <property>



ID	FIRSTNAME	LASTNAME	SHIP_STREET	SHIP_CITY	SHIP_STATE	SHIP_ZIP	BILL_STREET	BILL_CITY	BILL_STATE	BILL_ZIP
1	Frank	Brown	45 N Main St	Chicago	Illinois	51885	100 W Adams St	Chicago	Illinois	60603

Complex Mapping

COMPOSITE KEYS

Composite Keys

- Composite Keys are multi-column Primary Keys
  - By definition these are natural keys
  - Have to be set by the application (not generated)
  - Generally found in legacy systems
  - Also create multi-column Foreign Keys

- There are several different mapping strategies:
  - Most common mapping uses an embeddable class as the composite key
  - Other mappings are not supported by both annotations and XML (either one or the other)

Composite Ids

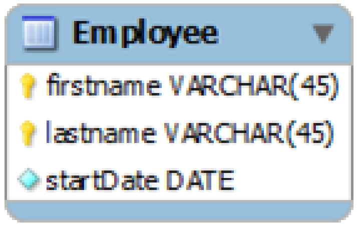
@Embeddable

```
@Embeddable
public class Name {
  private String firstname;
  private String lastname;

  ...
```

Also requires hashCode and equals methods (see next slide)

```
@Entity
public class Employee {
  @EmbeddedId
  private Name name;
  @Temporal(TemporalType.DATE)
  private Date startDate;

  ...
```

Embeddable object as identifier creates composite key



Employee

firstname VARCHAR(45)
lastname VARCHAR(45)
startDate DATE

PK is made of Both firstname and lastname

equals() & hashCode()

```
@Embeddable
public class Name {
  private String firstname;
  private String lastname;

  ...

  public boolean equals(Object obj) {
    if (this == obj)
      return true;
    if ((obj == null) || obj.getClass() != this.getClass())
      return false;
    Name n = (Name) obj;
    if (firstname == n.firstname || (firstname != null && firstname.equals(n.firstname))
      && lastname == n.lastname || (lastname != null && lastname.equals(n.lastname))) {
      return true;
    } else {
      return false;
    }
  }

  public int hashCode() {
    int hash = 1234;
    if (firstname != null)
      hash = hash + firstname.hashCode();
    if (lastname != null)
      hash = hash + lastname.hashCode();
    return hash;
}
```

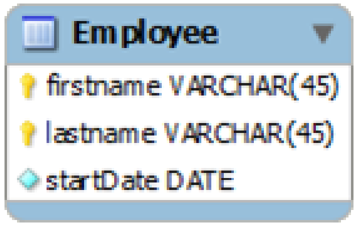
Compares object contents for equality

Generates an int based on the class contents

XML

```
<hibernate-mapping package="composite_key">
  <class name="Employee">
    <composite-id name="name" class="Name">
      <key-property name="firstname" />
      <key-property name="lastname" />
    </composite-id>

    <property name="startDate" type="date"/>
  </class>
</hibernate-mapping>
```

<composite-id> tag is used in XML to specify the property and class name



Employee

firstname VARCHAR(45)
lastname VARCHAR(45)
startDate DATE

PK is made of Both firstname and lastname

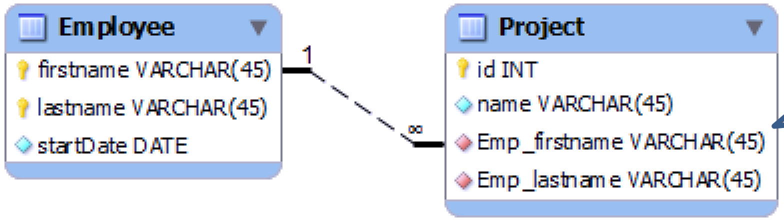
Foreign Keys to Composite Ids

```java
@Entity
public class Employee {
  @EmbeddedId
  private Name name;
  @Temporal(TemporalType.DATE)
  private Date startDate;
  @OneToMany(mappedBy = "owner")
  private List<Project> projects = new ArrayList<Project>();

  ...
```

Same Name embeddable @Id as before

Normal mappedBy on this side

```java
@Entity
public class Project {
  @Id
  @GeneratedValue
  private int id;
  private String name;
  @ManyToOne
  @JoinColumns( {
    @JoinColumn(name = "Emp_firstname", referencedColumnName = "firstname"),
    @JoinColumn(name = "Emp_lastname", referencedColumnName = "lastname")
  })
  private Employee owner;

  ...
```



Two column Foreign Key

Two column FK specification

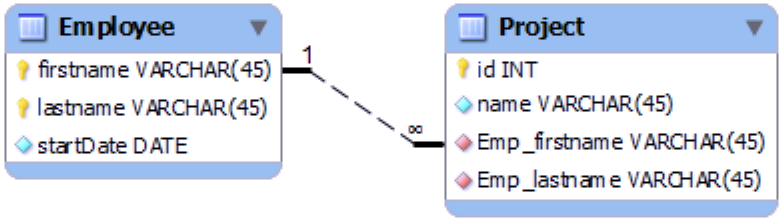
XML Composite FK

```
<hibernate-mapping package="composite_key">
  <class name="Employee">
    <composite-id name="name" class="Name">
      <key-property name="firstname" />
      <key-property name="lastname" />
    </composite-id>
    <property name="startDate" type="date" />

    <bag name="projects" inverse="true">
      <key>
        <column name="Emp_firstname" />
        <column name="Emp_lastname" />
      </key>
      <one-to-many class="Project" />
    </bag>
  </class>
</hibernate-mapping>
```

Even though the collection is inverse we still need to specify both columns

Using <column> tags inside <key> instead of the column attribute on <key>

```
<hibernate-mapping package="composite_key">
  <class name="Project">
    <id name="id">
      <generator class="native" />
    </id>

    <many-to-one name="owner" class="Employee">
      <column name="Emp_firstname" />
      <column name="Emp_lastname" />
    </many-to-one>
  </class>
</hibernate-mapping>
```



Using <column> tags inside <many-to-one> instead of the column attribute on it

Complex Mapping

ELEMENT COLLECTIONS

Element Collections

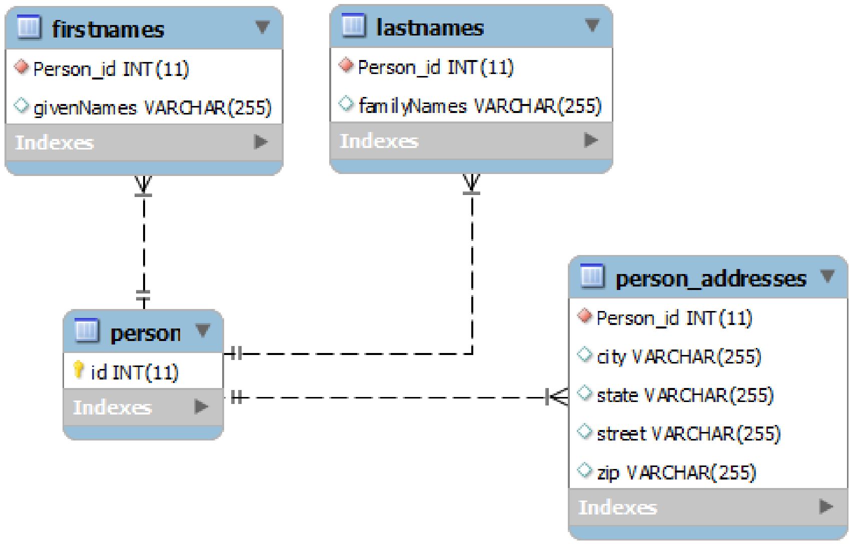
- For collections of primitive values or collections of embeddables
- Does not really make sense from a OO / UML point of view
- Good to know about

@ElementCollection

```
@Entity
public class Person {
  @Id @GeneratedValue
  private int id;
  @ElementCollection
  @CollectionTable(name = "firstNames")
  private List<String> givenNames = new ArrayList<>();
  @ElementCollection
  @CollectionTable(name = "lastNames")
  private List<String> familyNames = new ArrayList<>();
  @ElementCollection
  private List<Address> addresses = new ArrayList<>();
  ...
```

Optionally specify the name for the collection table

Default table name is: Classname_propertyname

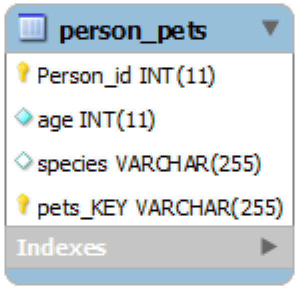


Map

```
@Entity
public class Person {
  @Id @GeneratedValue
  private int id;
  private String name;
  @ElementCollection
  @MapKeyColumn(name = "name")
  private Map<String, Pet> Pets = new HashMap<>();
  ...
```

Optionally specify the name for the additional key column

Default key column name is: propertyname_KEY


person_pets
- Person_id INT(11)
- age INT(11)
- species VARCHAR(255)
- pets_KEY VARCHAR(255)

Complex Mapping

IMMUTABLE ENTITIES

Immutable Entities

- An immutable entity is an entity that
  - Once created, does not change – no updates
  - Hibernate can perform several optimizations

- A Java immutable class:
  - Only has getters methods, no setters
  - Sets all fields in the constructor
  - Gives Hibernate field access

Immutability

```
@Entity
@org.hibernate.annotations.Entity(mutable=false)
public class Payment {
  @Id
  @GeneratedValue
  private final int id;
  private final double amount;
  @Column(name="`to`")
  private final String to;
  @Column(name="`from`")
  private final String from;

  public Payment() {}
  public Payment(double amount, String to, String from) {
    this.amount = amount;
    this.to = to;
    this.from = from;
  }

  public int getId() { return id; }
  public double getAmount() { return amount; }
  public String getTo() { return to; }
  public String getFrom() { return from; }
}
```

Set mutable false using Hibernate Entity extension

Field access through placement of @Id

Data is set in constructor

Getters, but no Setters

XML

Default Field access

```
<hibernate-mapping package="immutable" default-access="field">
  <class name="Payment" mutable="false">
    <id name="id">
      <generator class="native" />
    </id>
    <property name="amount" />
    <property name="to" column="`to`" />
    <property name="from" column="`from`"/>
  </class>
</hibernate-mapping>
```

Mutable = false

(To and From are SQL keywords)

Active Learning

- What is a value class?

- Why do we need to implement hashcode() and equals() when using @EmbeddedId?

Module Summary

- In this module we covered some of the more interesting mappings possible with Hibernate
- Many of these mappings are very useful when mapping to a legacy database
- Embeddable components also have their place in non-legacy systems
  - Allow a fine-grained object model to be mapped to a more coarse and efficient db model
  - Sacrifices some flexibility for greater efficiency