CS 416
Web Programming

Java Persistance API (JPA) Cont.

Dr. Williams
Central Connecticut State University

Review - JPA

- Create an Entity Bean to represent a City. It should store the following information
  - City name
  - State
  - Population
- Complete the AddCityServlet to insert new cities
- Complete the LookUpCityServlet that looks up all cities with the specified state and outputs them

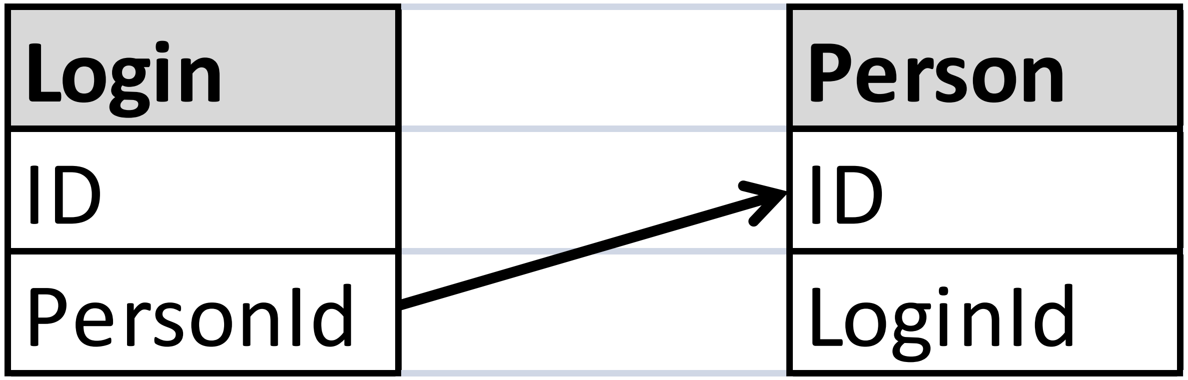
Relationships

- JPA supports loading related classes specified depending on the type of relationship:
  - One-to-one
  - One-to-many
  - Many-to-many
- Depending on the type of relationship and how it is represented in the database affects the JPA relationship syntax

One-to-one

- One to one relationship is created by linking one table to another via foreign key in which case the syntax is:

```
public class Login…
  @OneToOne
  @JoinColumn(name="personId")
  Private Person person;
```



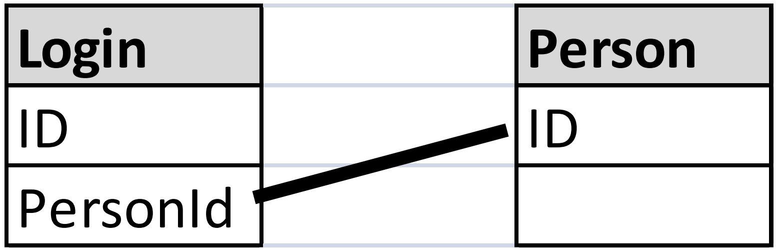
One-to-One with Foreign key

- When Login class is retrieved the associated person class is automatically retrieved and set on the Login bean

```
String queryString = "select l from Login l";
Query query = entityManager.createQuery(queryString);
List<Login> matchingLogins = query.getResultList();
for (Login curLogin : matchingLogins){
  out.println(curLogin.getId()+", "+
  curLogin.getPerson().getLastName()+"<br/>");
}
```

When relationship is not bidirectional

- With most database schemas not all one-to-one relationships are bidirectional, but you can make the relationship so in JPA



- Solution is to use mappedBy

```
public class Person…
@OneToOne(mappedBy = "person")
private Login login;
```

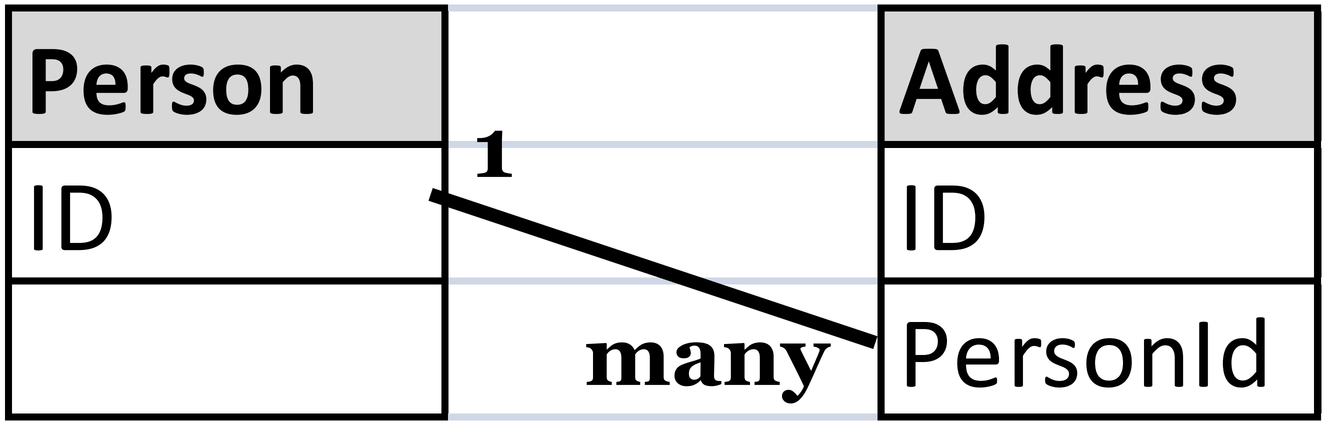
MappedBy value corresponds to what the attribute name is on the Login object that refers back to this object

Persisting data

- While the two beans are loaded together changes to a bean will only be persisted if THAT bean is persisted
- Scenario
    - load Customer bean (login bean is also loaded automatically)
    - Modify customer age
    - Modify login name
    - Persist customer
    - **Only** the age is updated in the DB

One to many relationships

- With JPA one-to-many/many-to-one relationships can be defined bidirection (in database usually only one direction)



Many-to-One

- From the “many” side the syntax is similar the one-to-one relationship as it is the object with the foreign key

```
public class Address{
    …
    @ManyToOne
    @JoinColumn(name=“personId”)
    private Person person;
```

One-to-many

- To add the one to many relationship
  - The “many” entities are stored in a java.util.Set (key is classes primary key)

```
public class Person implements
  Serializable {
@OneToMany(mappedBy="person")
private Set<Address> addresses;
```

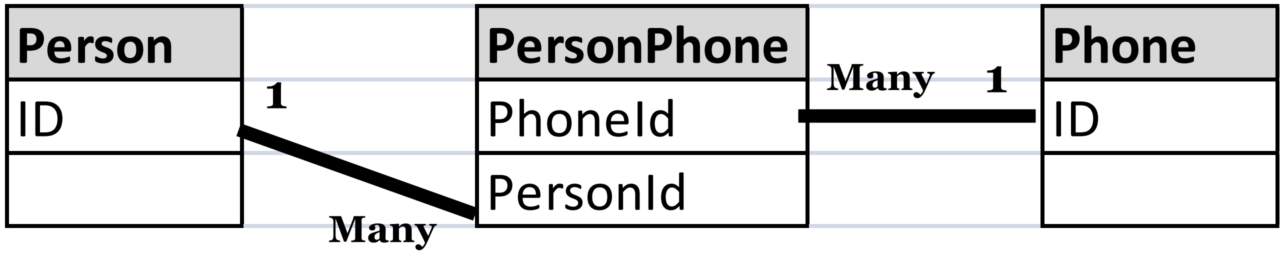
Persisting data

- As with all of the JPA relationships, related data is loaded but to persist related data it must be done explicitly

```
userTransaction.begin();
Person person = entityManager.find(Person.class, 0);
Address newAddress = new Address ();
newAddress.setCity("Peoria");
newAddress.setUsState("IL");
newAddress.setPerson(person);
entityManager.persist(newAddress);
userTransaction.commit();
```

Many-to-Many

- Many-to-many relationships mean that each object maps to 0 to many objects in both directions.
- To create a many to many relationship a **join table** is required. The table structure to represent this type of relationship is:



Many-to-many

- To annotate a many-to-many relationship in the entity beans the join table must be included in the annotation:

On one side: Phone class

```
@ManyToMany
@JoinTable(name="PersonPhone",
  joinColumns=@JoinColumn(name="phoneId",
                referencedColumnName="id"),

  inverseJoinColumns=@JoinColumn(name="person
  Id",
                  referencedColumnName="id"))
private Collection<Person> persons;
```

On the other its simple:

```
@ManyToMany(mappedBy="persons")
private Collection<Phone> phones;
```

Composite primary keys

- A composite primary key is when a table’s primary key is made up of 2 or more fields
- To create an entity object with a composite key a key class must be defined
- Key class must implement standard Bean requirements + overide equals and hashCode()

On class with composite key

```
@Entity
@IdClass(value = EmailPK.class)
public class Email implements
 Serializable{
    @Id
    private String emailType;
    @Id
    private String address;
    public Email(){

    }
```

Lookup by composite key

```
EntityManager entityManager =
  entityManagerFactory.createEntityManager
  ();
EmailPK emailPK = new
  EmailPK("work","cwilliams@ccsu.edu");
Email email =
  entityManager.find(Email.class,emailPK);
```

On your own

Creating the relationship

- Create a one to many relationship between City (one) and Venue (many)
- Complete AddVenueServlet

Retrieval

- Complete the DisplayVenuesServlet

- Create a class Band and add a many-to-many relationship between Band and Venue