

CS 416

Web Programming

Java Persistence API (JPA)

Dr. Williams
Central Connecticut State University

Review JDBC database connectivity

- Use JDBC in servlet to select and display all people that match the specified first name
 - ReviewJDBCPeopleSearch
 - ReviewJDBCGetNamesServlet

Java Persistence API

- Java Persistence API (JPA)
 - Encapsulates “model” part of MVC
 - Simplifies retrieval and population of objects
 - Simplifies persisting objects to the database
- JPA allows a class to be mapped to a database table
 - Records can be retrieved as the mapped objects
 - Updates to objects can easily be saved off to their respective table
- Result is rather than using record sets can use objects

Java EE Entities

- JPA's mechanism for mapping a table to an object is to create *Entity* objects
- A entity object is a normal java bean that has annotations to indicate it is an Entity and how to map the object to a database table
- Adding @Entity above the class declaration lets the application server know the class maps to the database

Creating the mapping

- Mapping the bean to the database is done through annotation
- At the top of the class an annotation is made to indicate the table the bean is tied to:

```
@Table (name="Person")
```

- With JPA if the bean name matches the table name this annotation isn't required unless it maps to a different table than the name of the class

Creating the mapping cont.

- Within, the bean annotation is used to map each class attribute to the appropriate field in the table
- Attributes to columns is mapped using the annotation:

```
@Column(name = "fullName")  
private String personName;
```

Like with the table, if the attribute name matches the column name the annotation isn't needed

Primary keys

- An annotation is required to identify the primary key of the table

@Id

```
Private Integer id;
```

Annotations example

```
@Entity
@Table(name="Person")
public class Person implements Serializable {
    @Id
    @Column(name="ID")
    private Integer ID = null;

    @Column(name="firstname")
    private String firstName = null;

    @Column(name="lastname")
    private String lastName = null;

    @Column(name="age")
    private Integer age = null;
```


Persisting beans

- All interaction between the Entity classes and the database is controlled by the EntityManager
- The EntityManager provides functionality to lookup, add, update, and delete Entities
- To function an Entity manager must be tied to a database connection which is specified as a persistence unit

Persistence unit

- A persistence unit is a mapping of a persistence unit name to a connection pool
- This configuration is specified in the persistence.xml file

```
<persistence version="2.0" ...>
  <persistence-unit name="Lec17DemosPU" transaction-
type="JTA">
    <jta-data-source>jdbc/Lect16DB</jta-data-source>
    <exclude-unlisted-classes>>false</exclude-unlisted-
classes>
    <properties>
    </properties>
  </persistence-unit>
</persistence>
```

Creating Persistence Unit in NetBeans

- On the project right-click select New
- Under category select Persistence
- Under file types select Persistence Unit
- For the data source either select the JNDI name you gave the data source or you may need to select new data source and add it (jdbc/Lect16DB)
- Click finish

EntityManager and UserTransaction

- Once the persistence unit has been defined it can be used to create an EntityManagerFactory that can then create a EntityManager

```
@PersistenceUnit(unitName="Lec17DemosPU")  
private EntityManagerFactory  
    entityManagerFactory;
```

- Your application also needs an instance of a UserTransaction to control commits and rollbacks
 - If you are tying multiple things together you can get a generic transaction from the app server

```
@Resource  
private UserTransaction userTransaction;
```

- Alternatively if just the EntityManager just use it's transaction

```
entityManager.getTransaction()
```

Record creation

- To create a record with JPA all that is involved is creating the object to be stored and telling the Entity Manager to persist it.

```
// Create a new Person bean
Person person = new Person();
person.setID(1);
person.setFirstName("John");
person.setLastName("Doe");
person.setAge(24);
```

```
// Persist it to the database to create
the record
entityManager.persist(person);
```

Refer to JPAAAddPerson

Letting JPA do the work

- Mapping to an existing table is great if you want complete control of table creation, however you can also let JPA do the work for you
- If you persist an object that has no table, JPA will automatically create the table with the attribute names as the column names
- Rather than handling primary keys on our own. Let the system handle them for us by adding `@GeneratedValue`

Annotations example

`@Entity`

```
public class Pet implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue
```

```
    private Integer ID = null;
```

```
    private String name = null;
```

```
    private String type = null;
```

```
    private Integer age = null;
```

When class is deployed Pet table and sequence for generating value will be created

JPA queries

There are two ways to retrieve records (or in this case objects) from the database

- By primary key

```
Person person = entityManager.find(Person.class, 1);
```

- Or by querying

```
List matchingPeople = query.getResultList();
```

Rather than retrieving record sets for you to process, it brings back prepopulated objects

JPA query

- Creating a JPA query is similar to a prepared statement with some minor differences

The syntax is:

```
String queryString = "Select p from Person p  
    where p.firstName = :firstName";
```

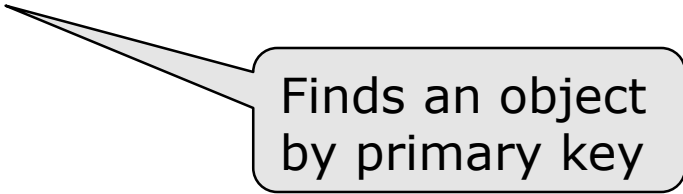
```
Query query =  
    entityManager.createQuery(queryString);  
query.setParameter("firstName", "Bob");  
List matchingPeople = query.getResultList();
```

Refer to NamesLikeServlet

Record update

- To update a record a object instance that represents a record is retrieved from the database. The field is updated and the object is persisted

```
// Retrieve person from database as an object  
Person person = entityManager.find(Person.class, 1);
```



Finds an object
by primary key

```
//update the object and write it back to the database  
person.setFirstName("Jon");  
entityManager.persist(person);
```

Refer to JPASimpleUpdatePerson

Record Delete

- Like update a instance, to delete a record needs to be retrieved then EntityManager is called to remove the object

```
// Retrieve person from database as an object  
Person person = entityManager.find(Person.class, 1);
```

```
// Delete a object from the database  
entityManager.remove(person);
```

Refer to JPADeletePerson

More examples

1. JPAAddPet to insert the passed pet information into the database
2. JPAUpdatePerson to search for ALL people with the specified first and last name and update them all to have the new last name
3. JPAPetSearch to retrieve all pets of the specified type – it should then enhance the request and forward to the PetDisplay.jsp to display the pets. Modify that JSP to loop through and display their names and ages