EJB - ENTITY RELATIONSHIPS

EJB 3.0 provides option to define database entity relationships/mappings like one to one, one to many, many to one and many to many relationships. Following are the relevant annotations.

- **OneToOne** - Objects are having one to one relationship. For example, a passenger can travel using a single ticket at time.
- **OneToMany** - Objects are having one to many relationship. For example, a father can have multiple kids.
- **ManyToOne** - Objects are having many to one relationship. For examples, multiple kids having a single mother.
- **ManyToMany** - Objects are having many to many relationship. For examples, a book can have mutiple authors and a author can write multiple books.

We'll demonstrate use of ManyToMany mapping here. To represent ManyToMany relationship, three tables are required.

- **Book** - Book table having records of books
- **Author** - Author table having records of author
- **Book_Author** - Book_Author table having linkage of above mentioned Book and Author table.

Create tables

Create a table **book author**, **book_author** in default database **postgres**.

```
CREATE TABLE book (
   book_id     integer,
   name   varchar(50)
);
```

```
CREATE TABLE author (
   author_id   integer,
   name   varchar(50)
);
```

```
CREATE TABLE book_author (
   book_id     integer,
   author_id   integer
);
```

Create Entity Classes

```
@Entity
@Table(name="author")
public class Author implements Serializable{
   private int id;
   private String name;
   ...
}
```

```
@Entity
@Table(name="book")
public class Book implements Serializable{
   private int id;
   private String title;
   private Set<Author> authors;
   ...
}
```

Use ManyToMany annotation in Book Entity

```
@Entity
public class Book implements Serializable{
   ...
   @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}
      , fetch = FetchType.EAGER)
   @JoinTable(table = @Table(name = "book_author"),
      joinColumns = {@JoinColumn(name = "book_id")},
      inverseJoinColumns = {@JoinColumn(name = "author_id")})
   public Set<Author> getAuthors()
   {
      return authors;
   }
   ...
}
```

Example Application

Let us create a test EJB application to test entity relationships objects in EJB 3.0.

Step	Description
1	Create a project with a name *EjbComponent* under a package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Please use the project created in *EJB - Persistence* chapter as such for this chapter to understand embedded objects in ejb concepts.
2	Create *Author.java* under package *com.tutorialspoint.entity* as explained in the *EJB - Create Application* chapter. Keep rest of the files unchanged.
3	Create *Book.java* under package *com.tutorialspoint.entity*. Use *EJB - Persistence* chapter as reference. Keep rest of the files unchanged.
4	Clean and Build the application to make sure business logic is working as per the requirements.
5	Finally, deploy the application in the form of jar file on JBoss Application Server. JBoss Application server will get started automatically if it is not started yet.
6	Now create the ejb client, a console based application in the same way as explained in the *EJB - Create Application* chapter under topic **Create Client to access EJB**.

EJBComponent *EJBModule*

Author.java

```
package com.tutorialspoint.entity;

import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="author")
public class Author implements Serializable{
   private int id;
   private String name;

   public Author(){}

   public Author(int id, String name){
       this.id = id;
       this.name = name;
   }

   @Id
   @GeneratedValue(strategy= GenerationType.IDENTITY)
   @Column(name="author_id")
   public int getId() {
      return id;
   }

   public void setId(int id) {
      this.id = id;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String toString(){
      return id + "," + name;
   }
}
```

Book.java

```
package com.tutorialspoint.entity;


import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;

@Entity
@Table(name="book")
public class Book implements Serializable{

    private int id;
    private String name;
    private Set<Author> authors;

    public Book(){
    }

    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    @Column(name="book_id")
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```
   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public void setAuthors(Set<Author> authors) {
      this.authors = authors;
   }

   @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}
      , fetch = FetchType.EAGER)
   @JoinTable(table = @Table(name = "book_author"),
      joinColumns = {@JoinColumn(name = "book_id")},
      inverseJoinColumns = {@JoinColumn(name = "author_id")})
   public Set<Author> getAuthors()
   {
      return authors;
   }
}
```

LibraryPersistentBeanRemote.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Remote;

@Remote
public interface LibraryPersistentBeanRemote {

   void addBook(Book bookName);

   List<Book> getBooks();

}
```

LibraryPersistentBean.java

```
package com.tutorialspoint.stateless;

import com.tutorialspoint.entity.Book;
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

@Stateless
public class LibraryPersistentBean implements LibraryPersistentBeanRemote {

   public LibraryPersistentBean(){
   }

   @PersistenceContext(unitName="EjbComponentPU")
   private EntityManager entityManager;

   public void addBook(Book book) {
      entityManager.persist(book);
   }

   public List<Book> getBooks() {
      return entityManager.createQuery("From Book").getResultList();
   }
}
```

- As soon as you deploy the EjbComponent project on JBOSS, notice the jboss log.
- JBoss has automatically created a JNDI entry for our session bean - **LibraryPersistentBean/remote**.
- We'll using this lookup string to get remote business object of type - **com.tutorialspoint.interceptor.LibraryPersistentBeanRemote**

JBoss Application server log output

```
...
16:30:01,401 INFO  [JndiSessionRegistrarBase] Binding the following Entries in Global
JNDI:
   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

LibraryPersistentBean/remote-com.tutorialspoint.interceptor.LibraryPersistentBeanRemote -
EJB3.x Remote Business Interface
16:30:02,723 INFO  [SessionSpecContainer] Starting
jboss.j2ee:jar=EjbComponent.jar,name=LibraryPersistentBean,service=EJB3
16:30:02,723 INFO  [EJBContainer] STARTED EJB:
com.tutorialspoint.interceptor.LibraryPersistentBeanRemote ejbName: LibraryPersistentBean
16:30:02,731 INFO  [JndiSessionRegistrarBase] Binding the following Entries in Global
JNDI:

   LibraryPersistentBean/remote - EJB3.x Default Remote Business Interface

LibraryPersistentBean/remote-com.tutorialspoint.interceptor.LibraryPersistentBeanRemote -
EJB3.x Remote Business Interface
...
```

EJBTester *EJBClient*

jndi.properties

```
java.naming.factory.initial=org.jnp.interfaces.NamingContextFactory
java.naming.factory.url.pkgs=org.jboss.naming:org.jnp.interfaces
java.naming.provider.url=localhost
```

- These properties are used to initialize the InitialContext object of java naming service
- InitialContext object will be used to lookup stateless session bean

EJBTester.java

```
package com.tutorialspoint.test;

import com.tutorialspoint.stateful.LibraryBeanRemote;
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class EJBTester {

   BufferedReader brConsoleReader = null;
   Properties props;
   InitialContext ctx;
   {
      props = new Properties();
      try {
         props.load(new FileInputStream("jndi.properties"));
      } catch (IOException ex) {
         ex.printStackTrace();
      }
      try {
         ctx = new InitialContext(props);
      } catch (NamingException ex) {
         ex.printStackTrace();
      }
      brConsoleReader =
      new BufferedReader(new InputStreamReader(System.in));
   }

   public static void main(String[] args) {

      EJBTester ejbTester = new EJBTester();

      ejbTester.testEmbeddedObjects();
   }

   private void showGUI(){
      System.out.println("**********************");
      System.out.println("Welcome to Book Store");
      System.out.println("**********************");
      System.out.print("Options \n1. Add Book\n2. Exit \nEnter Choice: ");
   }

   private void testEmbeddedObjects(){

      try {
         int choice = 1;

         LibraryPersistentBeanRemote libraryBean =
         (LibraryPersistentBeanRemote)
         ctx.lookup("LibraryPersistentBean/remote");

         while (choice != 2) {
            String bookName;
            String authorName;

            showGUI();
            String strChoice = brConsoleReader.readLine();
            choice = Integer.parseInt(strChoice);
            if (choice == 1) {
               System.out.print("Enter book name: ");
               bookName = brConsoleReader.readLine();
               System.out.print("Enter author name: ");
               authorName = brConsoleReader.readLine();
               Book book = new Book();
               book.setName(bookName);
      Author author = new Author();
      author.setName(authorName);
      Set<Author> authors = new HashSet<Author>();
      authors.add(author);
               book.setAuthors(authors);

               libraryBean.addBook(book);
            } else if (choice == 2) {
               break;
            }
         }

         List<Book> booksList = libraryBean.getBooks();

         System.out.println("Book(s) entered so far: " + booksList.size());
         int i = 0;
         for (Book book:booksList) {
            System.out.println((i+1)+". " + book.getName());
            System.out.print("Author: ");
            Author[] authors = (Author[])books.getAuthors().toArray();
            for(int j=0;j<authors.length;j++){
               System.out.println(authors[j]);
            }
            i++;
         }
      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }finally {
         try {
            if(brConsoleReader !=null){
               brConsoleReader.close();
            }
         } catch (IOException ex) {
            System.out.println(ex.getMessage());
         }
      }
   }
}
```

EJBTester is doing the following tasks.

- Load properties from jndi.properties and initialize the InitialContext object.
- In testInterceptedEjb method, jndi lookup is done with name - "LibraryPersistenceBean/remote" to obtain the remote business object *statelessejb*.
- Then user is shown a library store User Interface and he/she is asked to enter choice.
- If user enters 1, system asks for book name and saves the book using stateless session bean addBook method. Session Bean is storing the book in database.
- If user enters 2, system retrieves books using stateless session bean getBooks method and exits.

Run Client to access EJB

Locate EJBTester.java in project explorer. Right click on EJBTester class and select **run file**.

Verify the following output in Netbeans console.

```
run:
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 1
Enter book name: learn html5
Enter Author name: Robert
**********************
Welcome to Book Store
**********************
Options
1. Add Book
2. Exit
Enter Choice: 2
Book(s) entered so far: 1
1. learn html5
Author: Robert
BUILD SUCCESSFUL (total time: 21 seconds)
```