SECTION I: INTRODUCTION

Writing The First Application

Any project that requires database interaction have started looking at ORM tools than considering the traditional approach i.e. JDBC. This saves a huge amount of time revolving around unnecessary chores.

However, to make this more convincible, let's look at a practical implementation and work on it.

This is the only compelling reason why this chapter has been introduced. This chapter aims at convincing the readers/developers about how easy it is, to begin using an ORM tool via Java Persistence API.

This chapter does not cover the basics or a detailed explanation of ORM tool or the API. Those topics will be covered in the chapters that follow. This chapter leads through the building of a small example called GuestBook that uses JPA and a JPA provider.

To get the first application to work, the following needs to be setup:

- Database
- An ORM tool of choice
Persistence Configuration file
Entity - Plain Old Java Objects [POJOs]

Once all this is in place, the application logic needs to be written which uses JPA to actually do something.

Application Requirement Specifications

The application [example] to be built is called GuestBook. This application should be capable of accepting and displaying visitor's comments.

To achieve this, it should provide a user interface that accepts visitor's name and message/comments.

Diagram 3.1: GuestBook data entry form

After such information is captured and stored, other visitors to the application should be able to view all the available comments as shown in diagram 3.2.

This user interface displays the visitor's name along with the message and the date when the message was keyed in. It should also provide a link to sign the GuestBook which when clicked should display the GuestBook data entry form as shown in diagram 3.1.

Diagram 3.2: View GuestBook
Where Does ORM/JPA Fit

An ORM tool can be invoked from a Java application either directly or via another framework such as Struts, Spring and so on.

Java Persistence API makes it easy for these frameworks to support in one way or another.

Section VI: Enterprise Application Development explains how an ORM tool and JPA can be configured and integrated with such frameworks.

An ORM tool along with the JPA can be used in:

- A Swing application
- A Servlet
- A Portlet
- A JSP
- An Enterprise application
- Any other kind of Java application that has access to a database

Typically, an ORM tool is used to create a Data Access Layer for an application.

The most typical workflow would be:

- Define the configuration details using a Persistence Unit
- Create an EntityManagerFactory object by referencing the Persistence Unit
- Instantiate the EntityManager object through which the application accesses ORM tool's representation of the database

From this application's [GuestBook] point of view, an ORM tool will be used as follows:

- The user invokes the application
- The "Sign the Guest Book" data entry form is served to allow capturing the visitor's name and comments
The user keys in the details and clicks Submit

After such information is captured and the form is submitted, the server-side script in this case a JSP [GuestBookView.jsp] takes charge.

This script invokes the ORM as follows:

- Creates an `EntityManagerFactory` object by referencing the Persistence Unit
- Instantiate the `EntityManager` object
- Uses the `persist()` method of the instantiated `EntityManager` object to save the captured data to the configured database
- Uses the `createQuery()` method of the instantiated `EntityManager` object to query the configured database and fetch all the entries to display them

![View the Guest Book](image)

This user interface displays the visitor's name along with the message and the date when the message was keyed in.

Let's begin!

**Software Requirements**

From the application development perspective, the following software will be required on the development machine:

- Java Development Kit
- NetBeans IDE [The development IDE]
- MySQL Community Server [The database server]
- JDBC Driver for MySQL
- Glassfish Application Server v3
  - EclipseLink  [Default ORM that comes bundled]

OR
Glassfish Application Server v2

- TopLink [Default ORM that comes bundled]

**REMEMBER**

The above mentioned software setups are available in this book's accompanying CDROM.

**Library Files**

The following Java library [JAR] is required:

- **JDBC driver**: This will be specific to a relational database to be used. In this case MySQL is used as the database of choice, hence, the database specific JDBC driver file will be **MySQL Connector/J 5.1.7** [can be downloaded from http://www.mysql.com also available in the Book's accompanying CDROM]

**HINT**

An ORM tool does not include any database specific JDBC drivers. These must be obtained separately. Typically, the database provider offers them, as a separate downloads or bundled with the database installation.

**The Application Development Approach**

This application will be built using **JSP**.

The **data entry form** that captures the data will be called **GuestBookEntry.jsp** and the page that will fetch and display the entries will be called **GuestBookView.jsp**.

The **captured data** will be stored in a **table** called **GuestBook** under the **MySQL** database server.

In the Java application, the **POJO** that will represent the GuestBook database table will be called **myApp.Guestbook.java**.

Just to make this simple, the application development will be carried out/demonstrated using a development IDE called NetBeans IDE 6.5. Ensure that this IDE [available in the Book's accompanying CDROM] is installed on the development machine prior proceeding further.

Refer to **Appendix A: Installing The NetBeans IDE** for the installation steps.
The following are the steps that will help build this application.

1. Create the database schema
2. Create the Web Application
3. Add the Java libraries [JDBC driver] to the application
4. Create a POJO to represent the table in the database schema
5. Create a Persistence unit that points to the database server [MySQL] using JNDI
6. Annotate the POJO to indicate the mapping between the JavaBean properties and the columns in the table
7. Create JSPs with code spec:
   a. To build an `EntityManagerFactory` object by referencing the Persistence Unit
   b. To obtain an `EntityManager` object from the `EntityManagerFactory`
   c. To perform the required database operations

### Creating Database And Tables In MySQL

Since MySQL is the database server of choice, ensure that the MySQL database engine [available in the Book’s accompanying CDROM] is installed on the development machine prior proceeding further. This can also be downloaded from the website [http://www.mysql.com/download](http://www.mysql.com/download).

Login to the MySQL database using a valid username and password. The pre-created super administrator called `root` can also be used.

Create the database named `GuestBook`:

```
CREATE DATABASE GuestBook;
```

Switch to the database `GuestBook`:

```
USE GuestBook;
```

Create the table named `GuestBook`:

```
CREATE TABLE GuestBook(
  VisitorNo Int PRIMARY KEY AUTO_INCREMENT,
  VisitorName varchar(50),
  Message varchar(100),
  MessageDate varchar(40));
```
Creating A Web Application

Since NetBeans is the IDE of choice throughout this book, use it to create a new Web Application Project called GuestBook. Run the NetBeans IDE and create a new Web Application project, as shown in diagram 3.3.1.

Click . Name this Web application as GuestBook as shown in diagram 3.3.2.
This page is not part of the book preview.
Once the NetBeans IDE brings up the GuestBook application, the next step is to add the required library files [JDBC driver] to the GuestBook application.

**Adding The Required Library Files**

*It's a good practice to manually create a dedicated `lib` folder with all the required library files in the project folder and then using NetBeans add libraries from this folder as the source.*

To do so,

Create a directory called `lib` under `<Drive>::\NetBeansProjects\GuestBook`. [NetBeansProjects is the folder where NetBeans places the projects created]

**JDBC Driver For MySQL**

MySQL provides connectivity to client applications developed in the Java EE 5 via a JDBC driver named **MySQL Connector/J**.
MySQL Connector/J is a native Java driver that converts JDBC calls into the network protocol used by the MySQL database. MySQL Connector/J is a Type 4 driver, which means that MySQL Connector is pure Java code spec and communicates directly with the MySQL server using the MySQL protocol.

MySQL Connector/J allows the developers working with Java EE 5, to build applications, which interact with MySQL and connect all corporate data even in a heterogeneous environment.

Visit the site http://www.mysql.com to download the MySQL Connector/J JDBC Driver.

At the time of writing this book the latest version of the MySQL Connector/J was 5.1.7 [available in this Book's accompanying CDROM].

After it is downloaded, using any unzip utility such as Winzip unzip the contents of the zip file. Copy the mysql-connector-java-X.X.X-bin.jar library file to the lib directory created earlier under <Drive>:\NetBeansProjects\GuestBook to store the JDBC library file.

Now add this library file to the project using NetBeans. Expand the Web application project structure in the Project pane, if not already expanded. Right-click on the Libraries folder, select the Add JAR/Folder... menu item as shown in diagram 3.4.1.

![Diagram 3.4.1: Add Jar/Folder](image-url)
This page is not part of the book preview.
Now, let's move to the application development area.

**Creating A JavaBean Class**

To hold the captured data in a structured manner, a bean class is required. This class should expose the following properties:

<table>
<thead>
<tr>
<th>Property Name</th>
<th>To Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>visitorNo</td>
<td>The Primary Key value</td>
</tr>
<tr>
<td>visitorName</td>
<td>Visitor's Name</td>
</tr>
<tr>
<td>message</td>
<td>Message that the visitor enters</td>
</tr>
<tr>
<td>messageDate</td>
<td>The date/time on which the message was entered</td>
</tr>
</tbody>
</table>

The primary purpose of having such a class is to hold individual guestbook entry as and when they are captured.

This class must be annotated with `javax.persistence.Entity`. Entity classes become a table in a relational database. Instances of this class becomes rows in the table.

All entities must have a primary key. Keys can be a single field or a combination of fields. JPA also allows to auto generate the primary key in the database using the `@GeneratedValue` annotation.

The following are the steps to create the entity class using the NetBeans IDE:

1. Right click **Sources Package** directory, select **New → Java Class...** as shown in diagram 3.5.1

![Diagram 3.5.1: Creating entity Class](image-url)
2. Enter **Guestbook** in the Class Name textbox and enter **myApp** in the **Package** drop down list box as shown in diagram 3.5.2

3. Click **Finish**

This creates the bean class named **Guestbook.java** under the package called **myApp**.

### Guestbook.java [Code Spec]

Edit the **Guestbook.java** file with the following contents.

```java
1 package myApp;
2
3 import javax.persistence.Column;
4 import javax.persistence.Entity;
5 import javax.persistence.GeneratedValue;
6 import javax.persistence.GenerationType;
7 import javax.persistence.Id;
8 import javax.persistence.Table;
```
This page is not part of the book preview.
40 Java Persistence API In EJB 3 For Professionals

The identifier property [@Id] visitorNo maps to a column named VisitorNo

- The Primary Key value will be generated by MySQL [@GeneratedValue]

```
CREATE TABLE GuestBook(
    VisitorNo Int PRIMARY KEY AUTO_INCREMENT,
    VisitorName varchar(50),
    Message varchar(100),
    MessageDate varchar(40));
```

- Other properties visitorName, message, messageDate map to columns VisitorName, Message, MessageDate respectively

**HINT**

By default each field is mapped to a column with the name of the field. However, if column names are different than the field names, they can be specified using @Column(name="ColumnName").

All these attributes of the Guestbook class have JavaBean style property accessor methods.

The class also has a constructor with no parameters.

**Creating Persistence Unit [persistence.xml]**

JPA uses the persistence.xml file to create the connection and setup the required environment. This file is used to provide the information which is necessary for making database connections. In this file the JNDI data source that indicates the database driver, the database location, the user and the password is specified.

The NetBeans IDE comes bundled with a plug-in that allows creating a persistence unit using a simple wizard.

To create persistence.xml using this plug-in, click File ➔ New File, select the Persistence category and choose the option Persistence Unit and click Next as shown in diagram 3.6.1.
This page is not part of the book preview.
This page is not part of the book preview.
This page is not part of the book preview.
This page is not part of the book preview.
Click Finish.

This creates the `persistence.xml` file.

This file is populated with the appropriate code spec:

```xml
< ?xml version="1.0" encoding="UTF-8" ? >
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="GuestBookPU" transaction-type="RESOURCE_LOCAL">
        <non-jta-data-source>jdbc/GuestBook</non-jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties/>
    </persistence-unit>
</persistence>
```

This wizard also creates a `server resource` file. This file automatically creates a JNDI data source on the application server when this application is deployed.
This page is not part of the book preview.
Creating JSPs

Before creating the JSP, let's create a directory to hold JSPs.

The following are the steps to create the directory:
1. Right click **Web Pages** directory, select **New → Folder...** as shown in diagram 3.7.1

![Diagram 3.7.1: Creating Folder](image)

2. Enter the name **JSP** in the **Folder Name** textbox as shown in diagram 3.7.2
This page is not part of the book preview.
This page is not part of the book preview.
3. Click Finish

This creates the JSP named GuestBookEntry.jsp under the JSP directory created earlier.

**GuestBookEntry.jsp [Code Spec]**

Edit the `GuestBookEntry.jsp` file with the following contents.

```html
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <title>Guest Book</title>
  </head>
  <body bgcolor="pink">
    <table border="0" cellspacing="0" cellpadding="0" align="center" width="760">
      <tr>
        <td>
          <table border="0" cellspacing="0" cellpadding="0" width="100%">
            <tr>
              <td align="top" align="left" style="padding-right:0px; padding-left:0px;
                  padding-bottom:0px; font:24px/30px Georgia; width:228px; color:#786e4e;
                  padding-top:0px; height:37px;">
                Sign the Guest Book
              </td>
            </tr>
          </table>
        </td>
      </tr>
    </table>
  </body>
</html>
```

```html
<form action="/JSP/GuestBookView.jsp" method="post">
```

Explanation:
This is the data entry form that allows capturing the visitor's name and comments and submits the data to another server-side script called GuestBookView.jsp for further processing.

GuestBookView.jsp [Code Spec]

Using NetBeans create one more JSP called GuestBookView.jsp using the same steps as shown earlier.
Edit the `GuestBookView.jsp` file with the following contents.

```java
<%@page contentType="text/html" pageEncoding="UTF-8" import="java.util.List, java.util.Iterator, myApp.Guestbook, java.persistence.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

```import for JPA```

private EntityManagerFactory emf;
private EntityManager em;
private EntityTransaction tx;
List<Guestbook> guestbook;
```Building an EntityManagerFactory```

```Referencing the Persistence Unit```

```Obtaining an EntityManager object```

```String submit = request.getParameter("btnSubmit");
if(submit != null &amp; &amp; ("Submit").equals(submit)) {
    try {
        String guest = request.getParameter("guest");
        String message = request.getParameter("message");
        String messageDate = new java.util.Date().toString();
    ```Guestbook gb = new Guestbook();
gb.setVisitorName(guest);
gb.setMessage(message);
gb.setMessageDate(messageDate);
```Populating the entity with captured data```

```tx = em.getTransaction();
tx.begin();
em.persist(gb);
tx.commit();
```Beginning a transaction```

```Persisting the entity```

```Committing a transaction```

```catch (RuntimeException e) {
    if(tx != null) tx.rollback();
    throw e;
}
```}
```

```throw e;
```}
```

response.sendRedirect("GuestBookView.jsp");```
This page is not part of the book preview.
This page is not part of the book preview.
This page is not part of the book preview.
This page is not part of the book preview.
This page is not part of the book preview.
This page is not part of the book preview.
Here, an iterator is used to traverse through the List object called guestbook. This list object was populated earlier by:

```java
guestbook = em.createQuery("SELECT g from Guestbook g").getResultList();
```

Using `<TABLE>`, `<TR>` and `<TD>` the elements of the List object are placed.

### Editing The web.xml File

In NetBeans, by default, the web.xml file uses the index.jsp file as the welcome file i.e. whenever the application is run the web.xml file will display the index.jsp file.

This file needs to be edited to invoke the application's data entry form [GuestBookEntry.jsp] every time it's invoked.

Edit the web.xml file with following contents:

```
<web-app version="1.0" encoding="UTF-8">
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>Index.jsp</welcome-file>
        <welcome-file>GuestBookEntry.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```
The GuestBook Application Structure

Running The GuestBook Application

Now that the application is ready, let's run this application [source code available on this Book's accompanying CDROM].

Begin by building the project, using the NetBeans IDE.

To do so, right click the GuestBook project and select the Build menu item as shown in diagram 3.8.1.
Diagram 3.8.1: Building the project

Then run the application by right clicking the **GuestBook** project and selecting the **Run** menu item as shown in diagram 3.8.2.
This page is not part of the book preview.
This page is not part of the book preview.
Diagram 3.8.5: Viewing data

When the GuestBookView.jsp page appears, the data entered by the user is stored in the MySQL database table named GuestBook that was created earlier.

To ensure that this was done successfully, open MySQL command line utility and query the table to view the following output:

<table>
<thead>
<tr>
<th>VisitorNo</th>
<th>VisitorName</th>
<th>Message</th>
<th>MessageDate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sharanam Shah</td>
<td>Welcome Everyone, This is my first Guest Book</td>
<td>Mon Feb 23 15:14:43 ICT 2009</td>
</tr>
</tbody>
</table>

Click [here](http://localhost:8080/GuestBook/JSP/GuestBookView.jsp) link available on the top right corner of the page to go back to the Guest Book data entry form.

This chapter dealt with building a Web application and integrating JPA and Eclipselink into that application that exemplifies the core JPA concepts discussed in the first two chapters.

The Book CDROM holds the complete application source code built using the NetBeans IDE for the following applications:

- GuestBook_Chap03_TopLink [Using GlassFish v2 → TopLink]
- GuestBook_Chap03_EclipseLink [Using GlassFish v3 → EclipseLink]

This can be directly used by making appropriate changes [username/password] to the configuration file [persistence.xml / sun-resource.xml].