Oracle® Database

2 Day + Java Developer's Guide

12*c* Release 2 (12.2)
E85808-01
May 2017

Oracle Database 2 Day + Java Developer's Guide, 12*c* Release 2 (12.2)

E85808-01

Primary Author: Tanmay Choudhury

Contributing Authors: Tulika Das

Contributors: Kuassi Mensah, Nirmala Sundarappa

Contents

Preface

This preface discusses the intended audience and conventions of the *Oracle Database 2 Day + Java Developer's Guide*.. It also includes a list of related Oracle documents that you can refer to for more information.

Audience

This guide is intended for application developers using Java to access and modify data in Oracle Database. This guide illustrates how to perform these tasks using a simple Java Database Connectivity (JDBC) application. This guide uses the Oracle JDeveloper integrated development environment (IDE) to create the application. This guide can be read by anyone with an interest in Java programming, but it assumes at least some prior knowledge of the following:

- Java
- Oracle PL/SQL
- Oracle databases

Related Documents

For more information, see the following documents in the Oracle Database documentation set:

- *Oracle Database JDBC Developer's Guide*
- *Oracle Database Java Developer's Guide*
- *Oracle Universal Connection Pool Developer's Guide*

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Aims and Objectives of This Book

Java is a popular language among developers that is used to build various enterprise solutions.

This guide will help you understand all Java products used to build a Java application. You will learn about Oracle JDBC Thin driver, Universal Connection Pool (UCP), and Java in the Database (OJVM). You will also learn about how to use these in the Web application.

This application will show you how to store details of all employees in a sample organization, add a new employee, delete an employee, and provide a salary increment to all employee in the organization.

The Web application will have two distinct users — 'HR Admin' and 'HR Staff', with each having different roles and privileges.

Over a period of two days, this guide aims to help you build a Web application using all the latest tools and technologies, JDBC driver, UCP, Java in the Database, and Oracle Database 12*c* Release 2 (12.2).

The guide will help you set up all prerequisites to create the Web application, including installing Oracle Database 12*c* Release 2 (12.2).

What you will achieve in

- Part I
- Part II

1.1 Architecture of the Application

Architecture of the Web Application

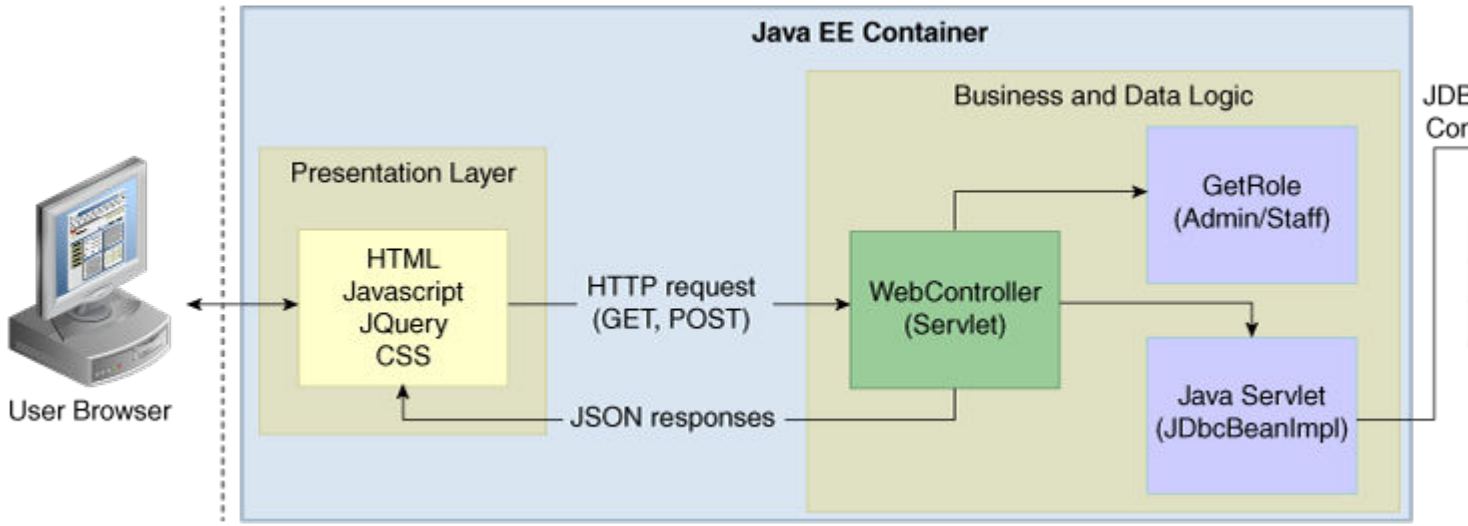
The HR Web application uses the MVC (Model, View, Controller) architecture and the latest tools and technologies.

For the View, or Presentation layer, you will use HTML that internally uses JavaScript, JQuery and CSS to display the results.

The Servlet will act as a controller that connects to Oracle Database through Java Beans. You will also use Maven to compile the entire Web application.

Following is a link to the Web application:
You will use HR schema and the Employees to understand the flows in the Web application.

Figure 1-1



Related Topics:

- https://github.com/oracle/oracle-db-examples/tree/master/java

1.2 Part I

Part I of this covers all tasks that you will cover in Day 1:

1 **Understand JDBC, UCP and Java in the Database:** You will familiarize yourself with the products, associated binaries and packages.

2 **Overview of the HR Web Application:** This chapter will discuss the HR Web application in depth and familiarize you with the flows of the Web application, packages and files that you will use in the Application.

3 **Getting Started with the Application:** You will understand the pre-requisites for building the application and how to get the environment ready. Some of the tools required to run the Application are Oracle Database 12*c* Release 2 (12.2), JDeveloper, Maven, and a Java EE to deploy and run the Application.

4 **List All Employees:** This chapter will help you how to put all components together and build an initial functionality to connect to the Oracle Database, and retrieve employee details from the database.

1.3 Part II

Part II covers all tasks you will complete in Day 2. You will learn how to use Universal Connection Pool (UCP) and Java in the Database (OJVM). You will also learn how to:

1 Search By Employee ID: This chapter provides details on how to implement the 'Search by Employee ID' functionality.

2 Update an Employee Record: In this chapter, you will learn how to update employee records.

3 **Delete an Employee Record:** Here, you will learn a series of steps to delete an employee record.

4 **Increase Salary to All Employees:** You will understand how to provide increment to the salary of the employees listed in the table.

5 **Summary:** This chapter will summarize all that you have learnt over the two days. It will also provide appropriate references and links for enhancing your use of the Application.

2
Using Java with Oracle Database

The Java Database Connectivity (JDBC) standard is used by Java applications to access and manipulate data in relational databases.

JDBC is an industry-standard application programming interface (API) that lets you access a RDBMS using SQL from Java. JDBC complies with the Entry Level of the JDBC escape standard. Each vendor implements the JDBC Specification with its own extensions.

Universal Connection Pool (UCP) is a connection pool used to cache the database connection objects to reuse the connections, thus improving the performance.

Java in the Database (OJVM) helps group SQL operations with Java data logic and load them into the database for in-place processing.

> **See Also:**
>
> `http://www.oracle.com/technetwork/java/overview-141217.html`

This chapter introduces you to the JDBC driver, Universal Connection Pool (UCP) and Java in the Database (OJVM) with Oracle Database 12*c* Release 2 (12.2)

- Java Database Connectivity Driver (JDBC)
- Universal Connection Pool (UCP)
- Java in the Database (OJVM)

2.1 Java Database Connectivity Driver (JDBC)

JDBC is a database access protocol that enables you to connect to a database and run SQL statement and queries on the database. JDBC drivers implement and comply with the latest JDBC specifications. Java application need to have `ojdbc8.jar` compatible with JDK8 in their classpath.

The core Java class libraries provide the JDBC APIs, `java.sql` and `javax.sql`

The following sections describe Oracle support for the JDBC standard:

- Oracle JDBC Thin Driver
- Oracle JDBC Packages

Oracle JDBC Thin Driver

Oracle recommends using the JDBC Thin Driver for most requirements. The JDBC Thin Driver will work on any system with a suitable Java Virtual Machine. (JVM). Some other client drivers that Oracle provides are JDBC thin driver, Oracle Call Interface (OCI) driver, Server side thin driver, and server side internal driver.

The JDBC Thin Driver is a pure Java, Type IV driver. The JDBC driver version)ojdbc8.jar) inludes support for JDK 8.

JDBC Thin Driver communicates with the server using SQL*Net to access the database.

> **See Also:**
>
> *Oracle Database JDBC Developer's Guide*

2.2 Universal Connection Pool

Connection pools help improve performance by reusing connection objects and reducing the number of times that connection objects are created.

Oracle Universal Connection Pool (UCP) is a feature rich Java connection pool that provides connection pool functionalities, along with high availability, scalability and load balancing with the help of tighter integration with Oracle Database configurations.

A Java application or container must have `ucp.jar` in their classpath, along with the `ojdbc8.jar` (JDK8), to be able to use UCP.

> **See Also:**
>
> *Oracle Universal Connection Pool Developer's Guide*

2.3 Java in the Database (OJVM)

Oracle Database has a Java Virtual Machine (JVM) that resides in the server. It helps Java applications running in the Oracle JVM on the server to access data present on the same system and same process.

Java in the Database is recommended for applications that are data-intensive. JVM has the ability to use the underlying Oracle RDBMS libraries directly, without the use of a network connection between the Java code and SQL data. This helps improve performance and execution. For data access, Oracle Database uses server-side internal driver when Java code runs on the server.

3

Overview of the HR Web Application

The HR Web Application is intended to give you access to information related to all employees of AnyCo Corporation.

The two types of users that will be able to access this application are:

- HRStaff
- HRAdmin

The HRStaff and HRAdmin accounts have different privileges.

HRStaff has read only access to the application and does not have privileges to update/delete an employee record. HRStaff can only List the employees and Search by Employee ID.

The HRAdmin, has complete control on the application and has read and write privileges. HRAdmin is the only user who has access to all functionalities of the application such as *update/delete an employee record*, or *provide salary increment for all employees*.

This Chapter has the following sections:

3.1 Functionalities of the HR Web Application

Following is a list of functionalities to access information related to AnyCo Corporation:

- **List All Employees**

  Use the `List All Employees` option to retrieve employee information. This function lists information such as `Employee_ID`, `First_Name`, `Last_Name`, `Email`, `Phone_Number`, `Job_Id`, and `Salary`.

- **Search By Employee ID**

  Use the`primary key` (which is the Employee ID) to search for a particular employee.

- **Update Employee Record**

  You can update employee records, using the `Update Employee Record` function. First, search for employees, based on the name of the employee. You can then update employee details in the record, such as `first_name`, `last_name`, `email`, `phone_number`, `job_id` and `salary` using the `UPDATE` function.

  Use the `DELETE` function to the delete the entire employee record from the database.

- **Increment Salary**

Through the increment salary tab, you can alter (increase or decrease) the percentage of salary for hike.

- **About**

  This page provides an overview of the HR Application and explains the various functionalities it offers.

3.2 Components and Repositories

The following table lists and describes all the components required for the application.

> **See Also:**
>
> You can download the web application from the following github link: https://github.com/oracle/oracle-db-examples/tree/master/java/HRWebApp

Package Name	Desciption
`src`	Contains source files
`target`	Contains class files
`src/main/java/com/oracle/jdbc/samples`	-
`/bean/JdbcBean.java`	Defines the employee details as attributes
`/bean/JdbcBeanImpl.java`	Implementation class of the EmployeeBean
`src/main/java/com/oracle/jdbc/samples`	-
`entity/Employee.java`	Consists of all employee attributes and their defined datatypes
`/web/WebController.java`	Servlet that controls the application flow
`/web/GetRole.java`	Creates HRStaff and HRAdmin roles for the application
`src/main/resources`	-
`SalaryHikeSP.java`	Java class to be invoked from Java in the database to process an increment in salary
`SalaryHikeSP.sql`	SQL file with a procedure to increase the salary of the employees based on their salary range
`src/main/webapp`	-
`about.html`	Contains the details about the HR Web application

`login.html`	Contains the login page for the HR Web application
`login-failed.html`	Page to show when the login is unsuccessful
`index.html`	Landing page of the HR Web application
`listAll.html`	HTML page to display all employee records
`listByName.html`	HTML page to display the result when employees are searched by name
`listById.html`	HTML page to display the result when employees are searched by employee id
`incrementSalary.html`	HTML page to display the result after an increment is made to the salary
`src/main/webapp`	-
`css/app.cs`	Contains all style and font details used in the HR Web application
`src/main/webapp`	-
`WEB-INF/web.xml`	Controller for the HR Web application

4
Getting Started with the Application

To develop a Java application that connects to Oracle Database 12*c* Release 2 (12.2), you must ensure that certain components are installed as required. This chapter covers the following topics:

- What You Need to Install
- Verifying the Oracle Database 12c Release 2 (12.2) Installation
- Installing Oracle JDeveloper or any Java IDE (Eclipse, NetBeans, Intellij)

4.1 What You Need to Install

To be able to develop the sample application, you need to install the following products and components:

- Oracle Database 12c Release 2 (12.2)
- J2SE or JDK
- Integrated Development Environment
- Web Server

The following subsections describe these requirements in detail.

4.1.1 Oracle Database 12*c* Release 2 (12.2)

To develop the Java application, you need a working installation of Oracle Database 12*c* Release 2 (12.2) Server with the `HR` schema, which comes with the database. The installation creates an instance of Oracle Database 12*c* Release 2 (12.2) and provides additional tools for managing this database. For more information, refer to the following Oracle Database 12*c* Release 2 (12.2) installation guides and release notes:

- *Oracle Database Installation Guide for Linux*
- *Oracle Database Installation Guide for Microsoft Windows*

4.1.1.1 Unlocking the HR Schema for the JDBC Application

The `HR` user account, which owns the sample HR schema used for the Java application in this guide, is initially locked. You must log in as a user with administrative privileges (`SYS`) and unlock the account before you can log in as `HR`.

If the database is locally installed, use the **Run SQL Command Line** to unlock the account as follows:

1. To access the **Run SQL Command Line**, from the **Start** menu, select **Programs** (or All Programs), then **Oracle Database 12c Release 2 (12.2)**, and then click **Run SQL Command Line**. Log in as a user with DBA privileges, for example:

   ```
   > CONNECT SYS AS SYSDBA;
   Enter password: password
   ```

2. Run the following command:

```
> ALTER USER HR ACCOUNT UNLOCK;
```

or,

```
> ALTER USER HR IDENTIFIED BY HR;
```

3. Test the connection as follows:

```
> CONNECT HR
Enter password: password
```

You should see a message indicating that you have connected to the database.

> **Note:**
>
> For information about creating and using secure passwords with Oracle Database 12*c* Release 2 (12.2), refer to *Oracle Database Security Guide*.

In addition, some of the constraints and triggers present in the `HR` schema are not in line with the scope of the Java application created in this guide. You must remove these constraints and triggers as follows using the following SQL statements:

```
DROP TRIGGER HR.UPDATE_JOB_HISTORY;
DROP TRIGGER HR.SECURE_EMPLOYEES;
DELETE FROM JOB_HISTORY;
```

4.1.2 J2SE or JDK

To create and compile Java applications, you need the full Java 2 Platform, Standard Edition, Software Development Kit (J2SE SDK), formerly known as the Java Development Kit (JDK).

> **Note:**
>
> Oracle Database 12*c* Release 2 (12.2) supports JDK 8.

> **See Also:**
>
> - `http://www.oracle.com/technetwork/java/javase/downloads/index.html` for information about installing Java
> - `http://www.oracle.com/technetwork/java/overview-141217.html` for information about the JDBC API

4.1.3 Integrated Development Environment

For ease in developing the application, you can choose to develop your application in an integrated development environment (IDE). This guide uses Oracle JDeveloper to create the files for this application.

4.1.4 Web Server

The sample application developed in this guide uses JavaServer Pages (JSP) technology to display information and accept input from users. To deploy these pages, you need a Web server with a servlet and JSP container, such as the Apache Tomcat application server.

This guide uses the embedded server called the Oracle WebLogic Server in JDeveloper for deploying the JSP pages. If you choose not to install Oracle JDeveloper, then any Web server that enables you to deploy JSP pages should suffice.

JDeveloper supports direct deployment to the following production application servers:

- Oracle WebLogic Server
- Oracle Application Server
- Apache Tomcat
- IBM WebSphere
- JBoss

For more information about these servers, please refer to vendor-specific documentation.

4.2 Oracle Database Cloud — Starting with Oracle Database as a Service

Prerequisites for Using Oracle Database Cloud Service

Before getting started with Oracle Database Cloud Service, you should be familiar with the following concepts:

- Oracle Cloud
- Oracle Compute Cloud Service
- Oracle Storage Cloud Service Controllers

  Database Cloud Service uses Oracle Database Backup Cloud service to back up data on cloud storage.

  Database Backup Cloud Service, on the other hand, uses Oracle Storage Cloud Service containers as repositories for backups to the cloud. Hence, before you can create a container, you must have a subscription to Oracle Storage Cloud Service.

What You Need to Create a Database Cloud Service Instance

- A Database Cloud Service Subscription

- A Secure Shell (SSH) public/private key pair to enable secure access to compute nodes supporting the database (optional)
- A container in Oracle Storage Cloud Service to store backups on cloud service

Getting Started with Database Cloud Service Subscriptions

Use the following steps to get started with Oracle Database Cloud Service (trial and paid) subscriptions:

1 Request a trial license, or purchase a license to Oracle Database Public Cloud Services.

2 Set up your Oracle Database Public Cloud Services account.

> **Note:**
>
> Refer the following link to download Oracle Database Cloud Service https://cloud.oracle.com/database

3 Verify if your Database Cloud Service is ready to use.

4 Know more about Oracle Database Cloud Service users and roles.

5 Create accounts for users with required privileges

For more information on how to get started with Oracle Database Cloud Service, refer:

https://docs.oracle.com/en/cloud/paas/database-dbaas-cloud/csdbi/get-started-this-service.html

4.3 Verifying the Oracle Database 12*c* Release 2 (12.2) Installation

Oracle Database 12*c* Release 2 (12.2) installation is platform-specific. You must verify that the installation was successful before you proceed to create the sample application. This section describes the steps for verifying an Oracle Database 12*c* Release 2 (12.2) installation.

Verifying a installation involves the following tasks:

5

Connecting to Oracle Database 12*c* Release 2 (12.2)

This chapter is the first in a series of five chapters, each of which describes how to create parts of a Java application that accesses Oracle Database 12*c* Release 2 (12.2) and displays, modifies, deletes, and updates data on it. To be able to access the database from a Java application, you must connect to the database using a `java.sql.Connection` object.

5.1 Creating an Employee Java Bean

The `Employee` Java bean creates getter and setter methods for columns to be displayed.

```
package com.oracle.jdbc.samples.entity;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;

public class Employee {

private int Employee_Id;
private String First_Name;
private String Last_Name;
private String Email;
private String Phone_Number;
private String Job_Id;
private int Salary;

public Employee (ResultSet resultSet) throws SQL Exception{
this.Employee_Id = resultSet.getInt(1);
this.First_Name = resultSet.getString(2);
this.Last_Name = resultSet.getString(3);
this.Email = resultSet.getString(4);
this.Phone_Number = resultSet.getString(5);
his.Job_Id = resultSet.getString(6);
this.Salary = resultSet.getInt(7);
}


public int getEmployee_Id(){
return Employee_Id;
}

public void setEmployee_Id(int Employee_Id){
this.Employee_Id = Employee_Id;
}

public String getFirst_Name(){
return First_Name;
}
```

```
public void setFirst_Name(String First_Name){
this.First_Name = First_Name;
}

public String getLast_Name(){
return Last_Name;
}

public void setLast_Name(String Last_Name){
this.Last_Name = Last_Name;
}

public String getPhone_Number(){
return Phone_Number;
}

public String getJob_Id(){
return Job_Id;
}

public void setJob_Id(String Job_Id){
this.Job_Id = Job_Id;
}

public int getSalary(){
return Salary;
}

public void setSalary(int Salary){
this.Salary = Salary;
}

}
```

5.2 Creating a Java Bean Interface for a JDBC Connection

```
package com.oracle.jdbc.samples.bean;
import com.oracle.jdbc.samples.entity.Employee;

public interface JdbcBean {
public List <name of employee> getEmployees();
}
```

5.3 Creating a Java Bean Implementation for a JDBC Connection

The following code helps to create Java Bean implementation for a JDBC Connection:

```
package com.oracle.jdbc.samples.bean;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import com.oracle.jdbc.samples.entity.Employee;
import java.sql.PreparedStatement;
importoracle.jdbc.OracleStatement;
```

```
import oracle.jdbc.OracleConnection;
import oracle.jdbc.driver.OracleDriver;
public class JdbcBeanImpl implements JdbcBean {
public static Connection getConnection() throws SQLException {
DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@//slc07qwu.us.oracle.com:5521/
jvma.regress.rdbms.dev.us.oracle.com", "hr", "hr");
return connection;

}
@Override
public List<Employee>getEmployees(){
List<Employee>returnValue = new ArrayList<>();
try (Connection connection = getConnection()) {
try (Statement statement = connection.createStatement()) {
try (ResultSet resultSet = statement.executeQuery("SELECT Employee_Id, First_Name,
Last_Name, Email, Phone_Number, Job_Id, Salary from employees")){
while(resultSet.next()){
returnValue.add(new Employee(resultSet));
}
}
}
}catch (SQLException ex){
logger.log(Level.SEVERE, null, ex);
ex.printStackTrace();
}
return returnValue;
}
```

5.4 Creating a Servlet to Process the Request

The following code describes the steps required to create a Servlet to process a request.

```
package com.oracle.jdbc.samples.web;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
import com.oracle.jdbc.samples.bean.Jdbcbean;
import com.oracle.jdbc.samples.bean.jdbcBeanImpl;
import com.oracle.jdbc.samples.entity.Employee;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;
import javax.servlet.http.*;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
@WebServlet(name="WebController",urlPatterns={"/WebController"})
public class WebController extends HttpServlet {
JdbcBean jdbcBean = new JdbcBeanImpl();
private void reportError(HttpServletResponse response, String message)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
try (PrintWriter out = response.getWriter() {
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<title>Servlet WebController</title>");
```

```
out.println(“</head>”);
out.println(“<body>)”;out.println(“<h1>”+message+”</h1>”);
out.println(“<h1>”+message+”</h1>”);
out.println(:</html>”);
}
}
protected void processRequest(HttpServletRequest request, HttpServletResponse
response)      throws ServletException, IOException {
Gson gson = new Gson();
String value = null;
List<Employee> employeeList = null;
employeeList = jdbcBean.getEmployees();
if(employeeList != null) {
response.setContentType("application/json");
gson.toJson(employeeList,
new TypeToken<ArrayList<Employee>>() { }.getType(),  response.getWriter());
} else {
response.setStatus(HttpServletResponse.SC_NOT_FOUND);
}
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
processRequest(request, response);
}
}
```

5.5 Create an HTML Page to Display Results

Class Name:

`src/main/webapp/listAll.html`

The following code describes how to create a method processResponse() inside the Java script that processes the JSON to show it on the HTML.

Sample Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>List all Employees</title>
<link rel="stylesheet" type="text/css" href="css/app.css" >
</head>
<body>
<div id="id-emp"></div>
<script>
var xmlhttp = new XMLHttpRequest();
var url = "WebController";
xmlhttp.onreadystatechange=function() {
if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
processResponse(xmlhttp.responseText);
}
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
function processResponse(response) {
var arr = JSON.parse(response);
var i;
var out = "<table>";
```

```
keys = Object.keys(arr[0]);
// Print headers
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<th>"+keys[i]+"</th>"
}
out += "</tr>";
// Print values
 for(j = 0; j < arr.length; j++) {
out += "<tr>"
for(i = 0; i < keys.length; ++i) {
out += "<td>"+arr[j][keys[i]]+"</td>"
}
out += "</tr>"
}
out += "</table>";
document.getElementById("id-emp").innerHTML = out;
}
</script>
</body>
</html>
```

5.6 Create a CSS File

The following code creates a method processResponse() inside the Java script that processes the JSON to show it on the HTML.

Sample Code:

```
table {
border-collapse:collapse;
width:100%;
}
th, td{
text-align:left;
padding:8px;
}
tr:nth-child(even){background-color: #f2f2f2}
th {
background-color: #4CAF50;
color: white;
}h {
background-color: #4CAF50;
color: white;
}
body {
font-family: "Lato", sans-serif;
}
.sidenav {
height: 100%;
width: 0;
position: fixed;
z-index: 1;
top: 0;
left: 0;
background-color: #FF0000;
overflow-x: hidden;
transition: 0.5s;
padding-top: 60px;
```

```
}
.sidenav a {
padding: 8px 8px 8px 32px;
text-decoration: none;
font-size: 25px;
color: black;
display: block;
transition: 0.3s
}
.sidenav a:hover, .offcanvas a:focus{
color: #f1f1f1;
}
.closebtn {
position: absolute;
top: 0;
right: 25px;
font-size: 36px !important;
margin-left: 50px;
}
#main {
transition: margin-left .5s;
padding: 16px;
}
```

6
Search by Employee ID

The following section illustrates the steps to search using Employee ID:

6.1 Employee Java Bean

Class Name:.

`/main/java/com/oracle/jdbc/samples/entity/Employee.java`

Use `Employee.java` that you created earlier in this illustration.

6.2 Create a Method in Java Bean for Search by Employee ID

Class Name:

`src/main/java/com/oracle/jdbc/samples/bean/JdbcBean.java`

Use the `getEmployee(int empID)` method to retrieve an employee record by ID.

```
/**
* Get List of employee based on empId. This will always return one row
* but returning a List to make signatures consistent.
* @param empId
* @return
* /
public List<Employee>getEmployee(int empId);
```

6.3 Implement a Method getEmployee() for Search by Employee ID

Class Name:

`src/main/java/com/oracle/jdbc/samples/bean/JdbcBean.java`
Use the `getConnection()` you created in the first step to connect to the same database.

Create a method `getEmployee()` to retrieve the employee based on the employee ID.

```
SELECT Employee_Id, First_Name, Last_Name, Email, Phone_Number, Job_Id, Salary FROM
EMPLOYEES WHERE
Employee_Id = ?
```

Following is a sample code:

```
public List<Employee>getEmployee(int empId) {
List<Employee>returnValue = new ArrayList<>();
try (Connection connection = getConnection()){
try (PreparedStatement preparedStatement = connection.prepareStatement(
“SELECT Employee_Id, First_Name, Last_Name, Email, Phone_Number, Job_Id, Salary FROM
EMPLOYEES WHERE Employee_Id = ?”)){
preparedStatement.setInt(1,empId);
try (ResultSet resultSet = preparedStatement.executeQuery()){
if(resultSet.next()){
returnValue.add(newEmployee(resultSet));
}
}
}
} catch (SQLException ex){
logger.log(Level.SEVERE, null, ex);
ex.printStackTrace();
}
return returnValue;
}
```

6.4 Create a New HTML for Search by Employee Id

Class Name:

`src/main/webapp/listById.html`

This HTML shows a text box to get the employee ID. Then, it generates and submits a request to search by employee ID.

Following is the code sample:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>List Employee by Id</title>
<link rel="stylesheet" type="text/css" href="css/app.css" >
<link rel="stylesheet" href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/
bootstrap.min.css">
</head>
<body>
<div><label>Employee Id: </label>
 <input id="empId" type="textfield"
onkeypress="return waitForEnter(event)"\></div>
<br/>
<br/>
<div id="id-emp"></div>
<script>
function waitForEnter(e) {
if (e.keyCode == 13) {
var tb = document.getElementById("empId");
fetchElementById(tb.value)
return false;
}
}
function fetchElementById(empId) {
var xmlhttp = new XMLHttpRequest();
var url = "WebController?id=" +empId;
xmlhttp.onreadystatechange=function() {
if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
```

```
processResponse(xmlhttp.responseText);

}
}
xmlhttp.open("GET", url, true);
xmlhttp.send();
}
function processResponse(response) {  var arr = JSON.parse(response);  var out =
"";  if (arr == null) {    out = '<div class="alert alert-warning"><strong>Alert!</
strong>'
+' No records found for the given id</div>'  }
else {
var i;    out = "<table>";
keys = Object.keys(arr);
out += "<tr><th>Trash</th><th>Edit</th>"
for(i = 0; i < keys.length; ++i) {
out += "<th>"+keys[i]+"</th>"
}
out += "</tr>"
out += '<tr><td><a href="javascript:confirmDelete()">'
+'<span  class="glyphicon glyphicon-trash"></span>'
+'</a></td>'
+'<td><a href="javascript:allowEditSalary()">'
+'<span class="glyphicon glyphicon-edit"></span>'
+'</a></td>'
for(i = 0; i < keys.length; ++i) {
out += "<td id='" +keys[i]+"'>"+arr[keys[i]]+"</td>"
}
out += "</tr>"
}
document.getElementById("id-emp").innerHTML = out;
}
</script>
</body>
</html>
```

7

Update an Employee Record

Create an Employee Java Bean:

Class Name:

`src/main/java/com/oracle/jdbc/samples/entity/Employee.java`

Use the `Employee.java` file that you created, earlier in the example.

7.1 Create a Method in Java Bean for Search by Employee's First Name

Use the Employee.java that you created in the earlier exercise.

Class Name:

`src/main/java/com/oracle/jdbc/samples/bean/JdbcBean.java`

Create a new method `getEmployeeByFn(String fn);` in the bean

Following is the code to create a method for search by employee's first name:

```
/**
* Get List of employees by First Name pattern
* @param fn
* @return List of employees with given beginning pattern
*/
public List<Employee> get EmployeeByFn(String fn);
```

7.2 Implement a Method `getEmployeebyFn()` for Search by Employee name

Class Name:

`src/main/java/com/oracle/jdbc/samples/bean/JdbcBeanImpl.java`

Use the `getConnection()` method that you created in the first step, to connect to the same database.

Then, create a method getEmployeeByFn() employee data based on employee's first name.

```
SELECT Employee_Id, First_Name, Last Name, Email, Phone_Number, Job_Id, Salary FROM
EMPLOYEES
WHERE First_Name LIKE ?
public List<Employee> getEmployeeByFn(String fn) {
List<Employee> returnValue = new ArrayList<>();
public List<Employee> getEmployeeByFn(String fn) {
List<Employee> returnValue = new ArrayList<>();
```

```
try (Connection connection = getConnection()) {
try (PreparedStatement preparedStatement = connection.prepareStatement(
"SELECT Employee_Id, First_Name, Last_Name, Email, Phone_Number, Job_Id, Salary FROM
EMPLOYEES WHERE First_Name LIKE ?")) {
preparedStatement.setString(1, fn + '%');
try (ResultSet resultSet = preparedStatement.executeQuery()) {
while(resultSet.next()) {
returnValue.add(new Employee(resultSet));
}
}
}
} catch (SQLException ex) {
logger.log(Level.SEVERE, null, ex);
ex.printStackTrace();
}
return returnValue;
}
```

8
Best Practices

This is the start of your topic.

Index