iBatis

1. Homepage http://ibatis.apache.org/

2. History
   1) 반복적이고 지루한 JDBC 프로그래밍을 단순화하기 위해 Clinton Begin 이 만든 작은 라이브러리에서 출발
   2) 2004년 ASF 에 기부
   3) Apache iBatis retired at the ASF(2010-06-16)
      a. iBatis project team has moved to mybatis hosted at google code.
         - http://www.mybatis.org
      b. Apache iBATIS moved into the Attic in June 2010.
   4) 지금은 Google Code 를 떠나 Github(github.com/mybatis)로 이사했음.

3. What is iBatis?
   1) iBATIS was a data mapper framework that made it easier to use a relational database with object-oriented applications.
   2) iBATIS is a persistence framework which automates the mapping between SQL databases and objects in Java, .NET, and Ruby on Rails.
   3) iBATIS makes it easier to build better database oriented application more quickly and with less code.
   4) There were both Java and .Net implementations.
   5) The mappings are decoupled from the application logic by packaging the SQL statements in XML configuration files.
   6) iBATIS is a lightweight framework and persistence API good for persisting POJOs( Plain Old Java Objects).
   7) iBATIS is what is known as a data mapper and takes care of mapping the parameters and results between the class properties and the columns of the database table.
   8) A significant difference between iBATIS and other persistence frameworks such as Hibernate is that iBATIS emphasizes use of SQL, while other frameworks typically use a custom query language such has the Hibernate Query Language (HQL) or Enterprise JavaBeans Query Language (EJB QL).
   5) Archived versions of iBATIS may be downloaded from the Apache Archives( http://archive.apache.org/dist/ibatis/).
   6) 개발과 유지보수가 쉽도록 소스 코드에 박혀있는 SQL을 별도의 파일로 분리하는 것이 핵심
   7) DAO.java속에 있는 SQL 문을 자바 코드로 부터 분리하여 xml 파일로 생성함.

4. Download & Configuration
   1) Visit https://archive.apache.org/dist/ibatis/binaries/ibatis.java/
   2) 목록에서 ibatis-2.3.4.726.zip click
   3) Downloads and unzip downloaded file
   4) If Web Application develp, copy ibatis-2.3.4.726.jar to /WEB-INF/lib
   5) Javadoc
      - http://ibatis.apache.org/docs/java/user/

5. Create src\SqlMapConfig.xml
   1) Consider the following:
      - We are going to use JDBC to access the database test.
      - JDBC driver for Oracle XE is "oracle.jdbc.driver.OracleDriver".
      - Connection URL is "jdbc:oracle:thin:@localhost:1521:XE".
      - We would use username and password is "scott" and "tiger".
      - Our sql statement mappings for all the operations would be described in "Employee.xml".
   2) Based on the above assumption we have to create an XML configuration file with name src\SqlMapConfig.xml with the following content.
   3) Copy ibatis-2.3.4.726/simple_example/com/mydomain/data/SqlMapConfig.xml to src\SqlMapConfig.xml. And Rewrite like below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
        "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
    <properties resource="dbinfo.properties" />
    <settings useStatementNamespaces="true"/>
    <transactionManager type="JDBC">
        <dataSource type="SIMPLE">
            <property name="JDBC.Driver" value="${db.driverClass}"/>
            <property name="JDBC.ConnectionURL" value="${db.url}"/>
            <property name="JDBC.Username" value="${db.username}"/>
```

```
            <property name="JDBC.Password" value="${db.password}"/>

            <!--<property name="JDBC.Driver" value="oracle.jdbc.driver.OracleDriver"/>
            <property name="JDBC.ConnectionURL" value="jdbc:oracle:thin:@localhost:1521:XE"/>
            <property name="JDBC.Username" value="scott"/>
            <property name="JDBC.Password" value="tiger"/>-->
        </dataSource>
    </transactionManager>
    <sqlMap resource="com/javasoft/libs/model/Employee.xml"/>
</sqlMapConfig>
```

4) There are other optional properties which you can set using SqlMapConfig.xml file:

```
- <property name="JDBC.AutoCommit" value="true"/>
- <property name="Pool.MaximumActiveConnections" value="10"/>
- <property name="Pool.MaximumIdleConnections" value="5"/>
- <property name="Pool.MaximumCheckoutTime" value="150000"/>
- <property name="Pool.MaximumTimeToWait" value="500"/>
- <property name="Pool.PingQuery" value="select 1 from Employee"/>
- <property name="Pool.PingEnabled" value="false"/>
```

6. iBatis Create Operation

1) To perform any CRUD(Create, Read, Write, Update and Delete) operation using iBATIS, you would need to create a POJOs(Plain Old Java Objects) class corresponding to the table.

2) This class describes the objects that will "model" database table rows.

3) The POJO class would have implementation for all the methods required to perform desired operations.

4) We have following EMPLOYEE table in Oracle XE:

```
CREATE TABLE Employee (
    id NUMBER(4),
    first_name VARCHAR2(20),
    last_name VARCHAR2(20),
    salary NUMBER(8),
    CONSTRAINTS employee_id_pk PRIMARY KEY (id)
);
```

5) Employee POJO Class:

```
package com.javasoft.libs.model;
public class EmployeeDTO {
    private int id;
    private String first_name;
    private String last_name;
    private int salary;
    /* Define constructors for the Employee class. */
    public EmployeeDTO() {}
    public EmployeeDTO(String fname, String lname, int salary) {
        this.first_name = fname;
        this.last_name = lname;
        this.salary = salary;
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = 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 int getSalary() {
        return salary;
```

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

6) com.javasoft.libs.model.Employee.xml File:

- To define SQL mapping statement using iBATIS, we would use <insert> tag and inside this tag definition we would define an "id" which will be used in IbatisInsert.java file for executing SQL INSERT query on database.
- Copy ibatis-2.3.4.726/simple_example/com/mydomain/data/Account.xml to com.javasoft.libs.model.Account.xml
- Rename Account.xml to Employee.xml
- Rewrite like below

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sqlMap
    PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
        "http://ibatis.apache.org/dtd/sql-map-2.dtd">

<sqlMap namespace="Employee">
    <typeAlias alias="employeeDTO" type="com.javasoft.libs.model.EmployeeDTO"/>
    <insert id="insert" parameterClass="employeeDTO">
        <selectKey keyProperty="id" resultClass="int">
            SELECT seq_employee_id.NEXTVAL AS ID FROM DUAL
        </selectKey>
        INSERT INTO EMPLOYEE(id, first_name, last_name, salary)
        VALUES (#id#, #first_name#, #last_name#, #salary#)
    </insert>
</sqlMap>
```

7) src\IbatisInsert.java File:

- This file would have application level logic to insert records in the Employee table:

```
    import com.ibatis.common.resources.Resources;
    import com.ibatis.sqlmap.client.SqlMapClient;
    import com.ibatis.sqlmap.client.SqlMapClientBuilder;
    import com.javasoft.libs.model.EmployeeDTO;
    import java.io.*;
    import java.sql.SQLException;
    import java.util.*;
    public class IbatisInsert{
        public static void main(String[] args) throws IOException,SQLException{
            Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
            SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
            /* This would insert one record in Employee table. */
            System.out.println("Going to insert record.....");
            EmployeeDTO em = new EmployeeDTO("Zara", "Ali", 5000);
            smc.insert("Employee.insert", em);
            System.out.println("Record Inserted Successfully ");
        }
    }
```

- insert.html

```
    <!DOCTYPE html>
    <html>
        <head>
            <meta charset="UTF-8">
            <title>Insert Form</title>
        </head>
        <body>
            <h1>Join membership</h1>
            <form method="post" action="insert.jsp">
                <p><label for="first_name">First Name : </label>
                <input type="text" id="first_name" name="first_name" size="20"/></p>
                <p><label for="last_name">Last Name : </label>
                <input type="text" id="last_name" name="last_name" size="20"/></p>
```

```
            <p><label for="salary">Salary : </label>
            <input type="number" id="salary" name="salary" size="10" /></p>
            <input type="submit" value="join" />
        </form>
    </body>
    </html>
- insert.jsp
    <%@ page language="java" contentType="text/html; charset=utf-8"
    pageEncoding="utf-8"%>
    <%@ page import="com.ibatis.common.resources.Resources" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
    <%@ page import="java.io.Reader" %>
    <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
    <fmt:requestEncoding value="utf-8" />
    <jsp:useBean id="emp" class="com.javasoft.libs.model.EmployeeDTO" />
    <jsp:setProperty name="emp" property="*" />
    <%
        Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
        SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

        smc.insert("Employee.insert", emp);
        out.println("Success");
    %>

7. iBatis Read Operation
  1) com.javasoft.libs.model.Employee.xml File:
    - To define SQL mapping statement using iBATIS, we would add <select> tag in Employee.xml
    file and inside this tag definition we would define an "id" which will be used in IbatisRead.java
    file  for executing SQL SELECT query on database.
  2)
    <select id="getAll" resultClass="employeeDTO">
        SELECT * FROM EMPLOYEE
        ORDER BY id DESC
    </select>
  3) src\IbatisRead.java File:
    - This file would have application level logic to read records from the Employee table:
    import com.ibatis.common.resources.Resources;
    import com.ibatis.sqlmap.client.SqlMapClient;
    import com.ibatis.sqlmap.client.SqlMapClientBuilder;
    import com.javasoft.libs.model.EmployeeDTO;
    import java.io.*;
    import java.sql.SQLException;
    import java.util.*;
    public class IbatisRead{
        public static void main(String[] args) throws IOException,SQLException{
            Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
            SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
            /* This would read all records from the Employee table. */
            System.out.println("Going to read records.....");
            List <EmployeeDTO> ems = (List<EmployeeDTO>)smc.queryForList("Employee.getAll",
            null);

            for (EmployeeDTO e : ems) {
                System.out.print(" " + e.getId());
                System.out.print(" " + e.getFirst_name());
                System.out.print(" " + e.getLast_name());
                System.out.print(" " + e.getSalary());
                System.out.println("");
            }
            System.out.println("Records Read Successfully ");
        }
    }
  4) select.jsp
    <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
    <%@ page import="com.ibatis.common.resources.Resources" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
```

```
<%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
<%@ page import="java.io.Reader" %>
<%@ page import="java.util.List" %>
<%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
<%
   Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

   List<EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);

   out.println("<h2>Account List</h2>");
   out.println("<table border='1' style='table-layout:fixed;height:auto;margin:auto'>");
   out.println("<thead>");
   out.println("<tr>");
   out.println("<th>ID</th><th>First Name</th><th>Last Name</th><th>Salary</th>");
   out.println("</tr>");
   out.println("</thead>");
   out.println("<tbody>");
   for (EmployeeDTO e : list) {
      out.println("<tr>");
      out.print("<td>" + e.getId() + "</td>");
      out.print("<td>" + e.getFirst_name() + "</td>");
      out.print("<td>" + e.getLast_name() + "</td>");
      out.print("<td>" + String.format("%,d", e.getSalary()) + "</td>");
      out.println("</tr>");
   }
   out.println("</tbody>");
   out.println("</table>");
%>
```

8. iBatis Update Operation

1) Employee.xml File:

- To define SQL mapping statement using iBATIS, we would add <update> tag in Employee.xml file and inside this tag definition we would define an "id" which will be used in IbatisUpdate.java file for executing SQL UPDATE query on database.

```
<update id="update" parameterClass="employeeDTO">
   UPDATE EMPLOYEE
   SET    first_name = #first_name#
   WHERE  id = #id#
</update>
```

2) IbatisUpdate.java File:

- This file would have application level logic to update records into the Employee table:

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import com.javasoft.libs.model.EmployeeDTO;
import java.io.*;
import java.sql.SQLException;
import java.util.*;
public class IbatisUpdate{
   public static void main(String[] args) throws IOException,SQLException{
      Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

      System.out.println("Going to update record.....");
      EmployeeDTO emp = new EmployeeDTO();
      emp.setId(1);
      emp.setFirst_name( "Roma");
      smc.update("Employee.update", emp);
      System.out.println("Record updated Successfully ");
      System.out.println("Going to read records.....");
      List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);

      for (EmployeeDTO e : list) {
         System.out.print(" " + e.getId());
         System.out.print(" " + e.getFirst_name());
         System.out.print(" " + e.getLast_name());
```

```
                System.out.print(" " + e.getSalary());
                System.out.println("");
            }
            System.out.println("Records Read Successfully ");
        }
    }
```

3)update.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
    <%@ page import="com.ibatis.common.resources.Resources" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
    <%@ page import="java.io.Reader" %>
    <%@ page import="java.util.List" %>
    <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
    <%
        Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
        SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

        EmployeeDTO emp = new EmployeeDTO();
        emp.setId(3);
        emp.setFirst_name("Roma");
        smc.update("Employee.update", emp);

        List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);

        out.println("<h2>Account List</h2>");
        out.println("<table border='1' style='table-layout:fixed;height:auto;margin:auto'>");
        out.println("<thead>");
        out.println("<tr>");
        out.println("<th>ID</th><th>First Name</th><th>Last Name</th><th>Salary</th>");
        out.println("</tr>");
        out.println("</thead>");
        out.println("<tbody>");
        for (EmployeeDTO e : list) {
            out.println("<tr>");
            out.print("<td>" + e.getId() + "</td>");
            out.print("<td>" + e.getFirst_name() + "</td>");
            out.print("<td>" + e.getLast_name() + "</td>");
            out.print("<td>" + String.format("%,d", e.getSalary()) + "</td>");
            out.println("</tr>");
        }
        out.println("</tbody>");
        out.println("</table>");
    %>
```

9. iBatis Delete Operation

1) Employee.xml File:

- To define SQL mapping statement using iBATIS, we would add <delete> tag in Employee.xml file and inside this tag definition we would define an "id" which will be used in IbatisDelete.java file for executing SQL DELETE query on database.

```
    <delete id="delete" parameterClass="int">
        DELETE FROM EMPLOYEE WHERE  id = #id#
    </delete>
```

2) src\IbatisDelete.java File:

- This file would have application level logic to delete records from the Employee table:

```
        import com.ibatis.common.resources.Resources;
        import com.ibatis.sqlmap.client.SqlMapClient;
        import com.ibatis.sqlmap.client.SqlMapClientBuilder;
        import com.javasoft.libs.model.EmployeeDTO;
        import java.io.IOException;
        import java.io.Reader;
        import java.sql.SQLException;
        import java.util.List;

        public class IbatisDelete{
            public static void main(String[] args) throws IOException,SQLException{
                Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
```

```
            SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

            System.out.println("Going to delete record.....");
            int id = 1;
            smc.delete("Employee.delete", id );
            System.out.println("Record deleted Successfully ");
            System.out.println("Going to read records.....");
            List <EmployeeDTO> ems = (List<EmployeeDTO>)smc.queryForList("Employee.getAll",
            null);

            for (EmployeeDTO e : ems) {
               System.out.print(" " + e.getId());
               System.out.print(" " + e.getFirst_name());
               System.out.print(" " + e.getLast_name());
               System.out.print(" " + e.getSalary());
               System.out.println("");
            }
            System.out.println("Records Read Successfully ");
         }
      }
```

3)delete.jsp

```
   <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
   <%@ page import="com.ibatis.common.resources.Resources" %>
   <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
   <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
   <%@ page import="java.io.Reader" %>
   <%@ page import="java.util.List" %>
   <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>

   <%
      Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

      int id = 3;
      smc.delete("Employee.delete", id );

      List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.getAll", null);

      out.println("<h2>Account List</h2>");
      out.println("<table border='1' style='table-layout:fixed;height:auto;margin:auto'>");
      out.println("<thead>");
      out.println("<tr>");
      out.println("<th>ID</th><th>First Name</th><th>Last Name</th><th>Salary</th>");
      out.println("</tr>");
      out.println("</thead>");
      out.println("<tbody>");
      for (EmployeeDTO e : list) {
         out.println("<tr>");
         out.print("<td>" + e.getId() + "</td>");
         out.print("<td>" + e.getFirst_name() + "</td>");
         out.print("<td>" + e.getLast_name() + "</td>");
         out.print("<td>" + String.format("%,d", e.getSalary()) + "</td>");
         out.println("</tr>");
      }
      out.println("</tbody>");
      out.println("</table>");
   %>
```

10. iBatis Result Maps

1) The resultMap element is the most important and powerful element in iBATIS.

2) You can reduce upto 90% JDBC coding by using iBATIS ResultMap and in some cases allows you to do things that JDBC does not even support.

3) The design of the ResultMaps is such that simple statements don't require explicit result mappings at all, and more complex statements require no more than is absolutely necessary to describe the relationships.

4) Employee.xml File:

- Here we would modify Employee.xml file to introduce <resultMap></resultMap> tag.

- This tag would have an id which is required to run this resultMap in our <select> tag's resultMap attribute.

```
<!-- Using ResultMap -->
<resultMap id="result" class="EmployeeDTO">
   <result property="id" column="id"/>
   <result property="first_name" column="first_name"/>
   <result property="last_name" column="last_name"/>
   <result property="salary" column="salary"/>
</resultMap>
<select id="useResultMap" resultMap="result">
   SELECT * FROM Employee WHERE id=#id#
</select>
```

5) src\IbatisResultMap.java File:

- This file would have application level logic to read records from the Employee table using ResultMap:

```
import com.ibatis.common.resources.Resources;
import com.ibatis.sqlmap.client.SqlMapClient;
import com.ibatis.sqlmap.client.SqlMapClientBuilder;
import com.javasoft.libs.model.EmployeeDTO;
import java.io.Reader;
import java.io.IOException;
import java.sql.SQLException;

public class IbatisResultMap{
   public static void main(String[] args) throws IOException,SQLException{
      Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
      int id = 2;
      System.out.println("Going to read record.....");
      EmployeeDTO e = (EmployeeDTO)smc.queryForObject("Employee.useResultMap", id);
      System.out.println("ID: " + e.getId());
      System.out.println("First Name: " + e.getFirst_name());
      System.out.println("Last Name: " + e.getLast_name());
      System.out.println("Salary: " + e.getSalary());
      System.out.println("Record read Successfully ");
   }
}
```

## 11. iBatis Stored Procedures

1) This is very much possible to call a stored procedure using iBATIS configuration.

2)

```
CREATE OR REPLACE PROCEDURE sp_employee_insert
(
   v_fname     IN       Employee.first_name%TYPE,
   v_lname     IN      Employee.last_name %TYPE,
   v_salary       IN     Employee.salary%TYPE
)
IS
BEGIN
   INSERT INTO Employee(id, first_name, last_name, salary)
   VALUES (SEQ_EMPLOYEE_ID.NEXTVAL, v_fname, v_lname, v_salary);

   COMMIT;
END;
/
```

3) Employee.xml File:

- Here we would modify Employee.xml file to introduce <procedure></procedure> and <parameterMap></parameterMap> tags.
- Here <procedure></procedure> tag would have an id which we would use in our application to call the stored procedure.

```
<parameterMap id="setEmpInfoCall" class="map">
   <parameter property="v_fname" jdbcType="VARCHAR" javaType="java.lang.String"
   mode="IN"/>
   <parameter property="v_lname" jdbcType="VARCHAR" javaType="java.lang.String"
   mode="IN"/>
```

```
        <parameter property="v_salary" jdbcType="INT" javaType="java.lang.Integer" mode="IN"/>
      </parameterMap>
      <procedure id="setEmpInfo" parameterMap="setEmpInfoCall">
        { call sp_employee_insert(?,?,?) }
      </procedure>
```

4)IbatisSP.java File:

- This file would have application level logic to read name of the employee from the Employee table using ResultMap:

```
      import com.ibatis.common.resources.Resources;
      import com.ibatis.sqlmap.client.SqlMapClient;
      import com.ibatis.sqlmap.client.SqlMapClientBuilder;
      import java.io.*;
      import java.sql.SQLException;
      import java.util.*;
      public class IbatisSP{
        public static void main(String[] args) throws IOException,SQLException{
          Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
          SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
          int id = 1;
          System.out.println("Going to read employee name.....");
          Map<String, Object> map = new HashMap<String,Object>();
          map.put("v_fname", "Peter");
          map.put("v_lname", "Bok");
          map.put("v_salary", new Integer(3000));
          smc.insert("Employee.setEmpInfo", map);
          System.out.println("Record Insert Successfully ");
        }
      }
```

5)sp.jsp

```
      <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
      <%@ page import="com.ibatis.common.resources.Resources" %>
      <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
      <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
      <%@ page import="java.io.Reader" %>
      <%@ page import="java.util.List, java.util.Map, java.util.HashMap" %>
      <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>

      <%
        Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
        SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
        System.out.println("Going to read employee name.....");
        Map<String, Object> map = new HashMap<String,Object>();
        map.put("v_fname", "Peter");
        map.put("v_lname", "Bok");
        map.put("v_salary", new Integer(3000));
        smc.insert("Employee.setEmpInfo", map);
        System.out.println("Record name Successfully ");
      %>
```

12. Oracle Cursor with iBatis

1)storeprocedure sp_employee_select

```
      CREATE OR REPLACE PROCEDURE sp_employee_select
      (
        employee_cursor OUT SYS_REFCURSOR
      )
      AS
      BEGIN
         OPEN employee_cursor FOR
         SELECt * FROM employee;
      END;
```

2)Employee.xml

```
      <resultMap class="com.javasoft.libs.model.EmployeeDTO" id="selectResultMap">
        <result property="id" column="id" />
        <result property="first_name" column="first_name" />
        <result property="last_name" column="last_name" />
        <result property="salary" column="salary" />
```

```
    </resultMap>

    <parameterMap id="selectAllMap" class="java.util.Map">
      <parameter property="result" javaType="java.sql.ResultSet" jdbcType="ORACLECURSOR"
      mode="OUT"/>
    </parameterMap>

    <procedure id="selectAll" parameterMap="selectAllMap" resultMap="selectResultMap">
      { call sp_employee_select(?)}
    </procedure>
```

3)src\iBatisCursor.java

```
    import java.io.IOException;
    import java.io.Reader;
    import java.sql.SQLException;
    import java.util.HashMap;
    import java.util.List;
    import java.util.Map;

    import com.ibatis.common.resources.Resources;
    import com.ibatis.sqlmap.client.SqlMapClient;
    import com.ibatis.sqlmap.client.SqlMapClientBuilder;
    import com.javasoft.libs.model.EmployeeDTO;

    public class IbatisReadCursor{
      @SuppressWarnings("unchecked")
      public static void main(String[] args)
            throws IOException,SQLException{
        Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
        SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);

        Map map = new HashMap();
        List<EmployeeDTO> list = smc.queryForList("Employee.selectAll", map);
        for(int i = 0 ; i < list.size(); i++){
          EmployeeDTO employee = list.get(i);
          System.out.println("<ul>");
          System.out.println("<li>ID : " + employee.getId() + "</li>");
          System.out.println("<li>First Name : " + employee.getFirst_name() + "</li>");
          System.out.println("<li>Last Name : " + employee.getLast_name() + "</li>");
          System.out.println("<li>Salary : " + employee.getSalary() + "</li>");
          System.out.println("</ul>");
          System.out.println("-----------------------");
        }
      }
    }
```

4)cursor.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
    <%@ page import="com.ibatis.common.resources.Resources" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
    <%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
    <%@ page import="java.io.Reader" %>
    <%@ page import="java.util.List, java.util.Map, java.util.HashMap" %>
    <%@ page import="com.javasoft.libs.model.EmployeeDTO" %>
    <!DOCTYPE html>
    <html>
    <head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
    </head>
    <body>
      <%
      Reader rd = Resources.getResourceAsReader("SqlMapConfig.xml");
      SqlMapClient smc = SqlMapClientBuilder.buildSqlMapClient(rd);
      Map map = new HashMap();
      List<EmployeeDTO> list = smc.queryForList("Employee.selectAll", map);
      for(int i = 0 ; i < list.size(); i++){
        EmployeeDTO employee = list.get(i);
```

```
            out.println("<ul>");
            out.println("<li>ID : " + employee.getId() + "</li>");
            out.println("<li>First Name : " + employee.getFirst_name() + "</li>");
            out.println("<li>Last Name : " + employee.getLast_name() + "</li>");
            out.println("<li>Salary : " + employee.getSalary() + "</li>");
            out.println("</ul>");
         }
      %>
      </body>
      </html>
```

13. iBatis Dynamic SQL

1) Using dynamic queries is a very powerful feature of iBatis. Sometime you have changing WHERE clause criterion based on your parameter object's state.

2) In such situation iBATIS provides a set of dynamic SQL tags that can be used within apped statements to enhance the reusability and flexibility of the SQL.

3) All the logic is put in .XML file using some additional tags.

4) Following is an example where SELECT statement would work in two ways:
   - If you would pass an ID then it would return all the records corresponding to that ID
   - otherwise it would return all the records where employee ID is set to NULL.

5) Employee.xml File:
   - To define SQL mapping statement using iBATIS, we would add following modified <select> tag in Employee.xml file and inside this tag definition we would define an "id" which will be used in IbatisReadDy.java file for executing Dynamic SQL SELECT query on database.

```
      <select id="findByID" resultClass="employeeDTO">
            SELECT * FROM Employee
         <dynamic prepend="WHERE ">
            <isNotNull property="id">
               id = #id#
            </isNotNull>
         </dynamic>
      </select>
```

6) Above SELECT statement would work in two ways (i) If you would pass an ID then it would return records corresponding to that ID (ii) otherwise it would return all the records.

7) src\IbatisReadDy.java File:
   - This file would have application level logic to read conditional records from the Employee table:

```
      import com.ibatis.common.resources.Resources;
      import com.ibatis.sqlmap.client.SqlMapClient;
      import com.ibatis.sqlmap.client.SqlMapClientBuilder;
      import com.javasoft.libs.model.EmployeeDTO;
      import java.io.*;
      import java.sql.SQLException;
      import java.util.*;
      public class IbatisReadDy{
         public static void main(String[] args) throws IOException,SQLException{
            Reader rd=Resources.getResourceAsReader("SqlMapConfig.xml");
            SqlMapClient smc=SqlMapClientBuilder.buildSqlMapClient(rd);
            /* This would read all records from the Employee table.*/
            System.out.println("Going to read records.....");
            EmployeeDTC rec = new EmployeeDTO();
            rec.setId(1);
            List <EmployeeDTO> ems = (List<EmployeeDTO>)smc.queryForList("Employee.findByID",
            rec);

            for (EmployeeDTO e : ems) {
               System.out.print(" " + e.getId());
               System.out.print(" " + e.getFirstName());
               System.out.print(" " + e.getLastName());
               System.out.print(" " + e.getSalary());
               em = e;
               System.out.println("");
            }
            System.out.println("Records Read Successfully ");
         }
      }
```

8)dynamic.jsp

```
      <%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
<%@ page import="com.ibatis.common.resources.Resources" %>
<%@ page import="com.ibatis.sqlmap.client.SqlMapClient" %>
<%@ page import="com.ibatis.sqlmap.client.SqlMapClientBuilder" %>
<%@ page import="java.io.Reader" %>
<%@ page import="java.util.List" %>
<%@ page import="com.javasoft.libs.model.EmployeeDTO" %>

<%
   Reader rd=Resources.getResourceAsReader("SqlMapConfig.xml");
   SqlMapClient smc=SqlMapClientBuilder.buildSqlMapClient(rd);

   System.out.println("Going to read records.....");
   EmployeeDTO emp = new EmployeeDTO();
   emp.setId(22);
   List <EmployeeDTO> list = (List<EmployeeDTO>)smc.queryForList("Employee.findByID",
   emp);

   for (EmployeeDTO e : list) {
      System.out.print(" " + e.getId());
      System.out.print(" " + e.getFirst_name());
      System.out.print(" " + e.getLast_name());
      System.out.print(" " + e.getSalary());
   }
   System.out.println("Records Read Successfully ");
%>
```

14. iBatis OGNL Expressions

1) iBATIS provides powerful OGNL based expressions to eliminate most of the other elements.
   - if Statement
   - choose, when, otherwise Statement
   - where Statement
   - foreach Statement
2) The if Statement:
   - The most common thing to do in dynamic SQL is conditionally include a part of a where clause.

```
<select id="findActiveBlogWithTitleLike" parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG WHERE state = 'ACTIVE.
   <if test="title != null">
      AND title like #{title}
   </if>
</select>
```

   - This statement would provide an optional text search type of functionality.
   - If you passed in no title, then all active Blogs would be returned.
   - But if you do pass in a title, it will look for a title with the given likecondition.
   - You can include multiple if conditions as follows:
   - The most common thing to do in dynamic SQL is conditionally include a part of a where clause.

```
<select id="findActiveBlogWithTitleLike" parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG WHERE state = 'ACTIVE.
   <if test="title != null">
      AND title like #{title}
   </if>
   <if test="author != null">
      AND author like #{author}
   </if>
</select>
```

3) The choose, when, otherwise Statement:
   - iBATIS offers a choose element which is similar to Java's switch statement.
   - This helps choose only one case among many options.
   - Following example would search only on title if one is provided, then only by author if one is provided.
   - If neither is provided, let's only return featured blogs:

```
<select id="findActiveBlogWithTitleLike" parameterType="Blog" resultType="Blog">
   SELECT * FROM BLOG WHERE state = 'ACTIVE.
   <choose>
      <when test="title != null">
         AND title like #{title}
      </when>
```

```
            <when test="author != null and author.name != null">
              AND author like #{author}
            </when>
            <otherwise>
            AND featured = 1
            </otherwise>
          </choose>
        </select>
```

4) The where Statement:
- If we look previous examples, What happens if none of the conditions are met?
- You would end up with SQL that looked like this:

```
SELECT * FROM BLOG WHERE
```

- This would fail, but iBATIS has a simple solution with one simple change, everything works fine:

```
<select id="findActiveBlogLike" parameterType="Blog" resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
      state = #{state}
    </if>
    <if test="title != null">
      AND title like #{title}
    </if>
    <if test="author != null>
      AND author like #{author}
    </if>
  </where>
</select>
```

- The where element knows to only insert WHERE if there is any content returned by the containing tags.
- Furthermore, if that content begins with AND or OR, it knows to strip it off.

5) The foreach Statement:
- The foreach element is very powerful, and allows you to specify a collection, declare item and index variables that can be used inside the body of the element.
- It also allows you to specify opening and closing strings, and add a separator to place in between iterations.
- You can build an IN condition as follows:

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT * FROM POST P WHERE ID in
  <foreach item="item" index="index" collection="list" open="(" separator="," close=")">
    #{item}
  </foreach>
</select>
```

14. iBatis vs Hibernate

1) There are major differences between iBatis and Hibernate but both the solutions work well, given their specific domain.

2) Personally I would suggest you should use iBATIS if:
- You want to create your own SQL's and are willing to maintain them.
- Your environment is driven by relational data model.
- You have to work existing and complex schema's.

3) And simply use Hibernate if:
- Your environment is driven by object model and wants generates SQL automatically.

4) To count there are few differences:

a. iBATIS is:
- Simpler
- Faster development time
- Flixable
- Much smaller in package size

b. Hibernate:
- Generates SQL for you which means you don't spend time on SQL
- Provides much more advance cache
- Highly scalable

5) Other difference is that iBATIS makes use of SQL which could be database dependent where as Hibernate makes use of HQL which is relatively independent of databases and it is easier to change db in Hibernate.

6) Hibernate maps your Java POJO objects to the Database tables where as iBatis maps the ResultSet from JDBC API to your POJO Objets.

7) If you are using stored procedures, well you can do it in Hibernate but it is little difficult in comparision of iBATIS.
8) As an alternative solution iBATIS maps results sets to objects, so no need to care about table structures.
9) This works very well for stored procedures, works very well for reporting applications, etc .
10) Finally, Hibernate and iBATIS both are open source Object Relational Mapping(ORM) tools available in the industry.
11) Use of each of these tools depends on the context you are using them.
12) Hibernate and iBatis both also have good support from SPRING framework so it should not be a problem to chose one of them.