

Introduction to Modern Application Development
Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Indian Institute of Technology – Madras

Lecture – 28
Week 09 – Session 2

Hello. Welcome to the Session 2 of Week9.

Section 1: Review of the lecture

We have looked at:

- how MySQL is set up
- Set up of tables in a database using MySQL
- Querying in MySQL.



Aim of the lecture:

1. Use Java to extract data from MySQL. (We have done this once before, but we explore this in more detail.)
2. Implementing the same, but by using servlets.
3. Get data from a database and show it on a webpage using jsp.

Upon doing these, we will have brought together jsp and databases.

Upcoming lectures: Use cookies for implementing the logic of the application. This will complete the implementation of the entire web application.

(Refer Slide Time: 01:28)


Week 09


Review: accessing mysql from java

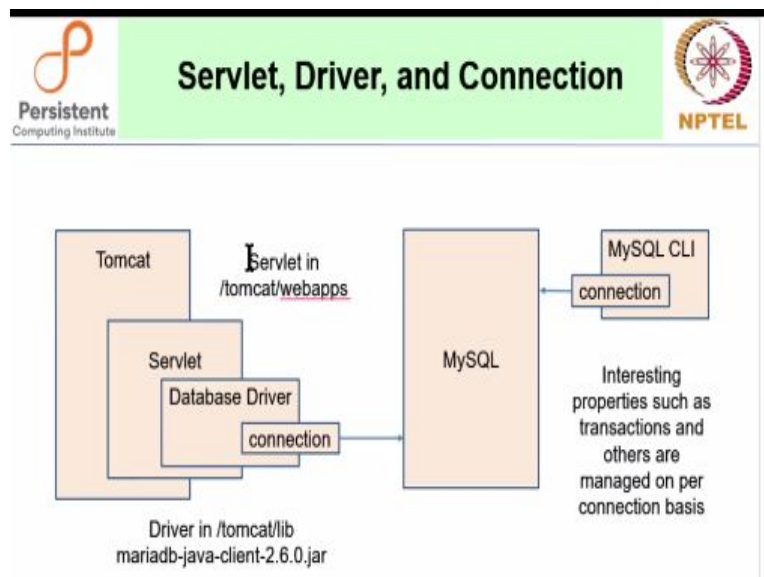
- Database drivers
- The database connection
- ResultSet

Review of this lecture:

- Access MySQL from java.
- Learn about database drivers and database connections.
- ResultSet object which contains information about the result of the statement from the database, how to access it and convert it into html.

SECTION 2: OVERALL PICTURE OF TOMCAT, SERVLETS AND DATABASE

(Refer Slide Time: 01:52)



1. Tomcat is the server.
2. There are servlets (classes which can be loaded when you configure them in the manager).

3. Servlet will communicate with the database using a database driver which, in turn, creates a database connection.
4. Literally, the database connection represents the TCP connection between the Tomcat process and the MySQL process. But figuratively, it also represents other data properties.

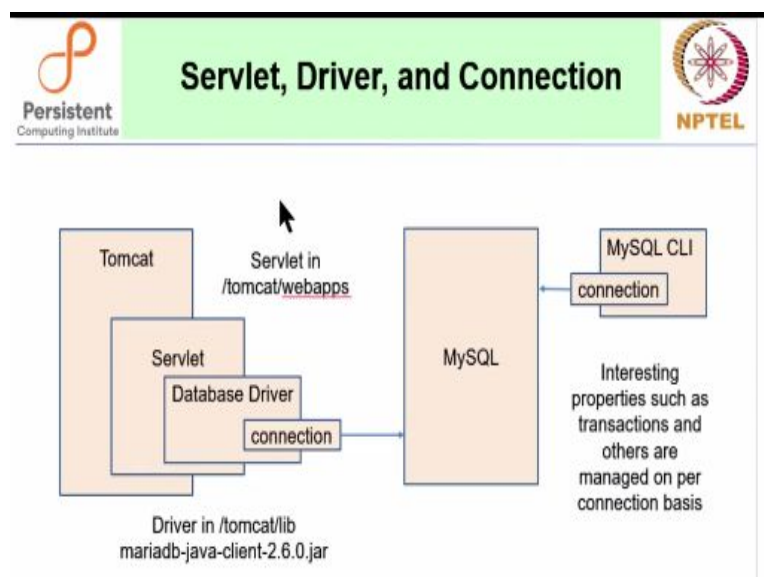
The process of how MySQL CLI interacts with the database is also similar. The CLI also opens a connection and it is through the connection that we communicate with MySQL.

However, unlike CLI, the arrangement of all this software is that the servlet is in tomcat's webapps. Tomcat layout has xampp which has tomcat. Tomcat directory has webapps which contain all the web applications that we are going to use. This is where the webapps are stored. There is a lib directory in tomcat directory which contains all the libraries. These are loaded by the tomcat server on the fly whenever it needs some facility.

Example of libraries in lib directory :

- tomcat-api.jar: We will see how to use it in embedded Tomcat.
- servlet-api.jar: It is used to compile the servlets and then used when we are loading them. This library is not Tomcat specific, it is a java standard and so there are other non-Tomcat java server sets that also implement this. **(Video Ends: 04:43)**

(Refer Slide Time: 04:44)



MySQL forked into MariaDB which is the open source version. So the servlet is in tomcat webapps, driver is in /tomcat/lib/mariadb-java-client-2.6.0.jar. Interesting properties of transactions are managed on a per connection basis and not having too many connections to the database is also important.

SECTION 3: How to establish a connection between driver and servlet?

To demonstrate this, consider the simple example where we retrieve a bunch of information from the database and show it on the webpage.

(Refer Slide Time: 05:39)



The slide features a green header with the title "MySQL Programming Strategy" in bold black text. On the left is the logo for "Persistent Computing Institute" (an orange infinity symbol) and on the right is the "NPTEL" logo (a circular emblem with a star). Below the header, the slide content consists of two bullet points:

- Have a simple file for basic testing
- Embed into servlet

- Test using a simple file: Code in this simple file is in the first step after some configuration that a servlet needs for different purposes. So next step will be servlet configuration.

The same code that we use in our testing app can be used in a servlet, but there are limitations to that code. For that reason we use JNDI resources. JNDI resource is basically a way of talking about databases without forcing ourselves to specialize to a specific database driver.

Implementation in three stages:

1. Write a simple program.
2. Put it in a servlet

3. Configure the servlet into a Tomcat friendly form.

Implement a simple class: readdatabase.java

REMEMBER:

1. The nptel database has 2 tables(`expenses` and `users`) which we created in the last lecture.
2. Users table has 3 attributes (`ctime`, `usrID`, `uname`).
3. If we do `select * from users`, we see the 3 users we added in the last lecture.
4. Today's goal is just to retrieve resulting rows of `select * from users` and display it on a webpage. We will also see how to get updates and whatever we need for our actual application.

STEP1:Simple code for retrieving information from this database using a programming language.

```
import java.sql.SQLException;
import java.sql.Statement;

public class readdb
{
    static String url      = "jdbc:mysql://localhost:3306/nptel";
    static String user     = "nptel_user";
    static String password = "npuserpwd";
    static String DBDriverClass = "org.mariadb.jdbc.Driver";

    static private Connection conn = null;
    static private Statement stmt = null;
    static private ResultSet rs = null;

    public static void main(String[] args) {
        try {
            Class.forName(DBDriverClass);

            conn = DriverManager.getConnection(url, user, password);
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from users");

            while (rs.next()) {
                long usrid = rs.getLong("usrid");
                long ctime = rs.getLong("ctime");
                String uname = rs.getString("uname");

                System.out.println("usrid: " + usrid +
                    ", ctime: " + ctime +
                    ", uname: " + uname);
            }
        }
    }
}
```

Create a test class called readdb.

- url field in readdb: "jdbc:mysql://localhost:3306/nptel" is used by another standard called *jdbc* which is a way of connecting to any database (here, MySQL).

- Url is written the following way: Protocol jdbc, followed by MySQL(database software), followed by varied runs and the last thing is the name of the database that you want to connect.
- Second thing this connection needs is the user and password.
- Finally the driver class itself (`DBDriverClass` field) which we have already put in `lib` as we have seen before.
- `connection` is the actual database connection class, `statement` is the class the statement you want to send to the database should be of, and the value of the statement comes back as type `resultSet`.
- In the main method: We ask to identify the class which loads `DBDriver`. `DriverManager` is defined in one of the imported packages (i.e. it is a part of `java.sql.*`). Load the `DriverManager`, establish a connection `conn`, create a statement using `stmt=conn.createStatement()`, to execute a specific query on the database, write the statement in `rs=stmt.executeQuery("select * from users")`. The result set is an object with an attribute `next`. `ResultSet` has a list of rows, with one row accessible at a time. The types of `ctime` and `usrID` is `bigint` which are 64 bit types. So in Java, they are represented as `Long` and therefore we get that information using `getLong()`. We can name each field in `resultSet` as well.
- The result set, the statement, and the connection need to be closed using `rs.close()` , `stmt.close()`, `conn.close()`; when we are done with the actual job. In the test program of course, it does not matter much if something goes wrong, we will print the stack trace and we will print a message.

compile this program with `javac readdb.java`, start MySQL, `java readdb`. This outputs the rows of the users table as expected.

How to do the same from the servlet and what does the output look like?

Copy the earlier created `dbforms` to `apps/nptel/`. This time, however, we will communicate with the `nptel` database.

So let us take a look at `src/nptel/dbforms`.

```

import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DbForms extends HttpServlet {

    private static final long serialVersionUID = 1L;

    static String url = "jdbc:mysql://localhost:3306/nptel";
    static String user = "nptel_user";
    static String password = "npuserpwd";
    static String DBDriverClass = "org.mariadb.jdbc.Driver";

    static Connection conn = null;
    static Statement stmt = null;
    static ResultSet rs = null;

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        request.setAttribute("title", "info via db");

        try {
            Class.forName(DBDriverClass);

            conn = DriverManager.getConnection(url, user, password);
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from users");

            while (rs.next()) {
                long usrid = rs.getLong("usrid");
                long ctime = rs.getLong("ctime");
                String uname = rs.getString("uname");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class readdb
{
    static String url = "jdbc:mysql://localhost:3306/nptel";
    static String user = "nptel_user";
    static String password = "npuserpwd";
    static String DBDriverClass = "org.mariadb.jdbc.Driver";

    static private Connection conn = null;
    static private Statement stmt = null;
    static private ResultSet rs = null;

    public static void main(String[] args) {
        try {
            Class.forName(DBDriverClass);

            conn = DriverManager.getConnection(url, user, password);
            stmt = conn.createStatement();
            rs = stmt.executeQuery("select * from users");

            while (rs.next()) {
                long usrid = rs.getLong("usrid");
                long ctime = rs.getLong("ctime");
                String uname = rs.getString("uname");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

DbForms.java vs readdb.java

- Import the same packages.
- Instead of just having readdb class, we use an extension of HttpServlet.
- Member variables of the classes are the same.
- DbForms has request.setAttribute(), i.e. initialisation to the html page to be printed by the web application.

```

throws IOException, ServletException
{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    request.setAttribute("title", "info via db");

    try {
        Class.forName(DBDriverClass);

        conn = DriverManager.getConnection(url, user, password);
        stmt = conn.createStatement();
        rs = stmt.executeQuery("select * from users");

        String start =
            "<html>\r\n" +
            "<head> <title> DB Forms </title> </head>\r\n" +
            "<body>\r\n";

        out.println(start);

        String tstart =
            "<table border='1' cellpadding='5'>\r\n" +
            "<caption><h2>Users Table</h2></caption>\r\n" +
            "<tr><th>usrid</th> <th>ctime</th> <th>uname</th></tr>";

        out.println(tstart);

        while (rs.next()) {
            long usrid = rs.getLong("usrid");
            long ctime = rs.getLong("ctime");
            String uname = rs.getString("uname");

            System.out.println("usrid: " + usrid +
                ", ctime: " + ctime +
                ", uname: " + uname);
        }

        rs.close();
        stmt.close();
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

static String password = "npuserpwd";
static String DBDriverClass = "org.mariadb.jdbc.Driver";

static private Connection conn = null;
static private Statement stmt = null;
static private ResultSet rs = null;

public static void main(String[] args) {
    try {
        Class.forName(DBDriverClass);

        conn = DriverManager.getConnection(url, user, password);
        stmt = conn.createStatement();
        rs = stmt.executeQuery("select * from users");

        while (rs.next()) {
            long usrid = rs.getLong("usrid");
            long ctime = rs.getLong("ctime");
            String uname = rs.getString("uname");

            System.out.println("usrid: " + usrid +
                ", ctime: " + ctime +
                ", uname: " + uname);
        }

        rs.close();
        stmt.close();
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

- There is a driver manager and connection, createStatement, etc. that we have seen in readdb.java. The only difference is now we are going to decorate the output with

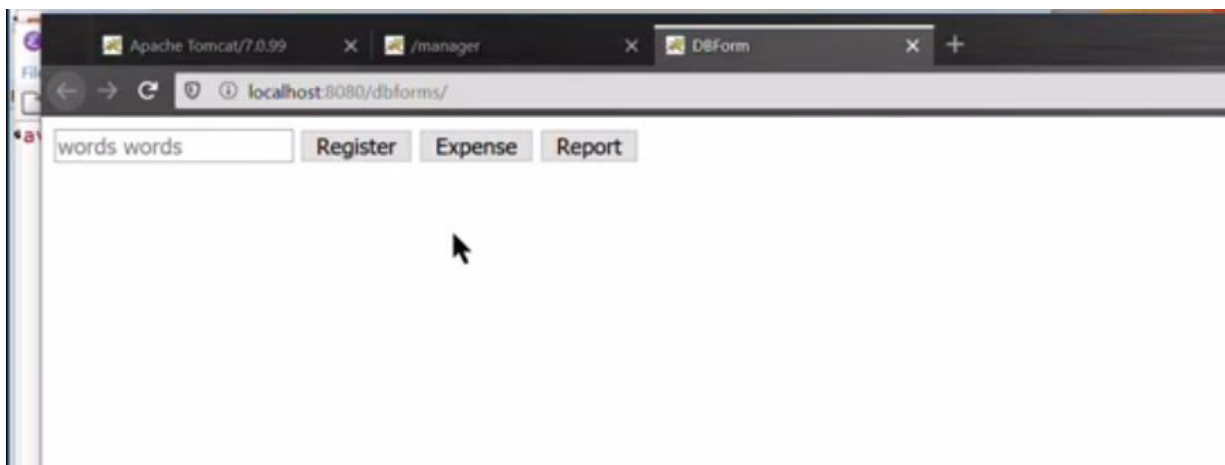
html.

Execute `jar cvf dbforms.war jsp WEB-INF`.

```
D:\apps\nptel\dbforms>jar cvf dbforms.war jsp WEB-INF
added manifest
adding: jsp/(in = 0) (out= 0)(stored 0%)
adding: jsp/dbform.jsp(in = 670) (out= 331)(deflated 50%)
adding: jsp/dbform.jsp~(in = 671) (out= 333)(deflated 50%)
adding: jsp/jspenv.jsp(in = 1458) (out= 487)(deflated 66%)
adding: jsp/jspenv.jsp~(in = 320) (out= 196)(deflated 38%)
adding: jsp/jspform.jsp(in = 671) (out= 333)(deflated 50%)
adding: jsp/jspform.jsp~(in = 583) (out= 320)(deflated 45%)
adding: jsp/jsptagenv.jsp(in = 1439) (out= 508)(deflated 64%)
adding: jsp/jsptagenv.jsp~(in = 1524) (out= 528)(deflated 65%)
adding: WEB-INF/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/nptel/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/classes/nptel/DbForms.class(in = 3227) (out= 1766)(deflated 45%)
adding: WEB-INF/classes/nptel/JspForms.class(in = 1478) (out= 778)(deflated 47%)
adding: WEB-INF/lib/(in = 0) (out= 0)(stored 0%)
adding: WEB-INF/lib/taglibs-standard-impl-1.2.5.jar(in = 206430) (out= 180631)(deflated 87%)
adding: WEB-INF/lib/taglibs-standard-spec-1.2.5.jar(in = 40153) (out= 34410)(deflated 86%)
adding: WEB-INF/web.xml(in = 859) (out= 323)(deflated 62%)
adding: WEB-INF/web.xml~(in = 876) (out= 331)(deflated 62%)
```

So, it adds `jspenv`, `WEB-INF` which have `classes/nptel/DbForms.class`. Open `web.xml<dbforms>`. When you visit `dbform` itself, we will just go to `/` and then when it is clicked again you open `dbforms` just like we did with `jspforms` the last time.

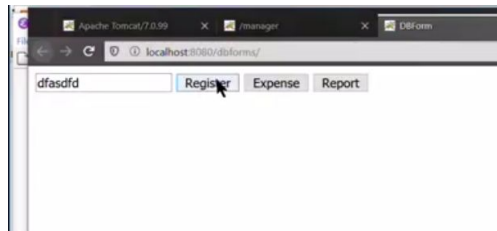
Deploy it as usual. Go to the `manager` app in Tomcat and we see `dbForms`.



The code associated with the layout.


```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" %>
<!DOCTYPE html>
<html>
  <head>
    <title> DBForm </title>
  </head>
  <body>
    <form action="/dbforms/dbshow" method="post"
      id="formid" name="formname" method="post">
      <div class="group">
        <input id="inp1" name="myfield" placeholder="words word
*s">
        <button id="btn1" name="Register" type="submit" value="R
*Register">Register</button>
        <button id="btn2" name="Expense" type="submit" value="E
*Expense">Expense</button>
        <button id="btn3" name="Report" type="submit" value="Re
*Report">Report</button>
      </div>
    </form>
  </body>
</html>
```

Type the following in the textbox:



Output:



usrid	ctime	uname
1	123	f1
2	124	f2
3	3334	f3

User table showed up here, it ignored the information in the textbox and executed the query “select * from users” as that was the only stmt in the connection, so the information in the textbox, which is on the servlet side of the database connection never reaches the database.

had this code which looked like this, where what you have to do is see get something called a request dispatcher from the request itself and then tell it to get access to the jsp which then you end up calling the request dispatcher with the same request response that we had before after putting these kinds of things into the request object and this user table again. Similar to jsp from last time, it has get attribute.

We were getting all these tables of attributes and parameters and so on and the general idea is that we had this kind of for loop. So, the same type of for loop exists here in user table and it is the user data that we are going to populate in this statement. Once populated, you extract all these tidbits using these types of names which are specific to the way jsp tag libraries work okay. So, you get the request dispatcher and then this is what we do, we include the title much like we included the title in this jsp form.

So, we had to remember the `request.setAttribute(title)`, and the next attribute we set it to users data. Users data is populated from the function which sets `resultSet` to a list of maps. So, we are getting the set of rows. We model the row with a map, key-value pairs and we stick all the maps into a list, i.e. the list of rows that we got.

Forward it to the request dispatcher which will do the job of running `,user table .jsp`. Both the users data are the same thing at this point. Value of `$users` data is list of maps. We took `dbforms` and instead of `dbforms` direct driver, we get it through the context object and then instead of just pumping out html, we do what we did for our `jspforms` and send it to a particular jsp, either jsp with `env`, with ordinary jsp or using jsp tags.

The only real novelty we have not seen so far is two things: one is how did all this context and all this thing get set up and what is this function doing?

The function is very simple. From the `resultSet`, get metadata object which tells us how many columns are there in the result set and we will be using that information below. Create a list which is an `ArrayList` loop over `rs.next`. For every row, create a new `HashMap` which has the same number of columns as we obtained and for each of these columns, start putting `getColumnName(i)` and `rs.getObject(i)`, etc.

So at the end of this, we have a database object corresponding to every row, we have the name of the column and the actual object and we add that to the list, return the value, put this value in to setAttribute, then extract it in the user table library.

So the forEach is done in jsp rather than doing it in code. Notice that unlike the usual Java loop which goes over 0 to i<columns, in order to keep interacting properly with this metadata object, we have to look from 1 to the end of the column.

The context gets set up in a new directory called META-INF which has context.xml file. It is describing to Tomcat what the particular data connection looks like. It says that it has all things like max active, max idle, etc.

It is mostly around managing the number of connections and you have username and password, driver class name, etc., etc. All of this because it is in the context.xml library is accessible to our class and therefore we are able to run this thing, initial context and data source. So, this is the general idea in code like this. You get information from the database. You convert that into a data structure, you send that data structure back into a JSP library. The database itself becomes known through code like this which talks about resources.

But as far as the servlet is concerned, it does not need to know too much about the details of MySQL versus Postgres or whatever other database oracle, etc that you are going to use and you write the loop. For every field in the database, you can have an output that looks like in the above case. As we saw last time, there is a work directory, which converts all jsp files into Java code at runtime and loads it. Here we have, for example, usertable_jsp .java.

```
File Edit Options Buffers Tools Java Help
-|- useretable.jsp.java 22% L66 (Java//1 Abbrev) <V>

}
return _jsp_instancemanager;
}

public void _jspInit() {
    _005fjspx_005ftagPool_005fc_005fforEach_0026_005fvar_005fitems = org.apache.jasper.runtime.TagHandlerPool.getTagHand
lerPool(getServletConfig());
}

public void _jspDestroy() {
    _005fjspx_005ftagPool_005fc_005fforEach_0026_005fvar_005fitems.release();
}

public void _jspService(final javax.servlet.http.HttpServletRequest request, final javax.servlet.http.HttpServletResponse
ponse response)
    throws java.io.IOException, javax.servlet.ServletException {

    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;

    try {
        response.setContentType("text/html; charset=ISO-8859-1");
        pageContext = _jspxFactory.getPageContext(this, request, response,
            null, true, 8192, true);
        _jspx_page_context = pageContext;

        null, true, 8192, true);
        _jspx_page_context = pageContext;
        application = pageContext.getServletContext();
        config = pageContext.getServletConfig();
        session = pageContext.getSession();
        out = pageContext.getOut();
        _jspx_out = out;

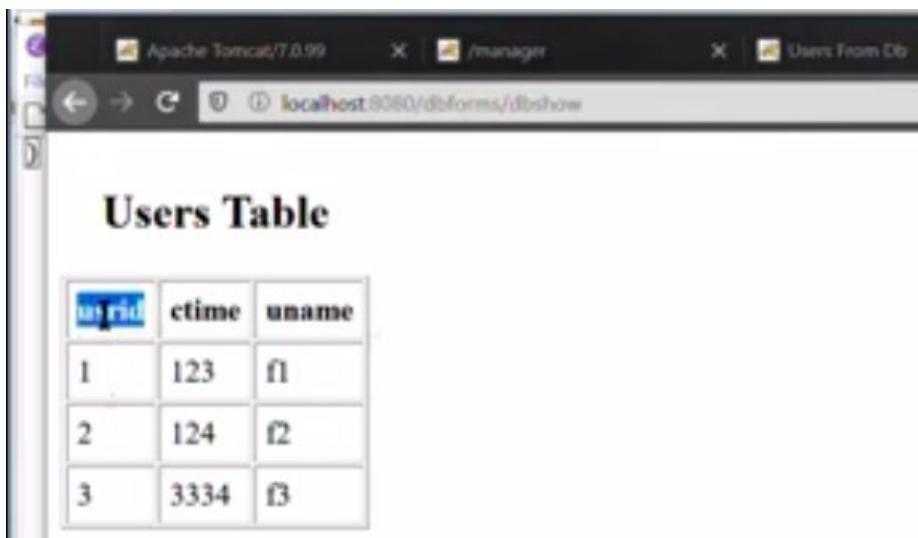
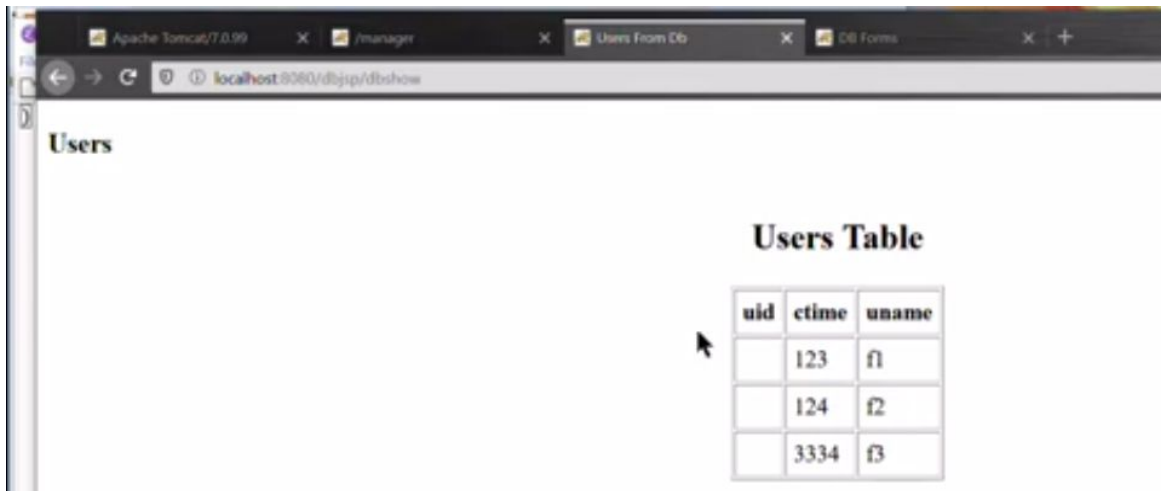
        out.write("\r\n");
        out.write("\r\n");
        out.write("\r\n");
        out.write("<DOCTYPE html>\r\n");
        out.write("<html>\r\n");
        out.write("  <head>\r\n");
        out.write("    <title> ");
        out.print(request.getAttribute("title"));
        out.write("  </title>\r\n");
        out.write("  </head>\r\n");
        out.write("  <h3> Users </h3>\r\n");
        out.write("\r\n");
        out.write("    <div align='center'>\r\n");
        out.write("\t<table border='1' cellpadding='5'>\r\n");
        out.write("\t  <caption>Users Table</caption>\r\n");
        out.write("\t  <tr>\r\n");
        out.write("\t\t<th>uid</th>\r\n");
        out.write("\t\t<th>ctime</th>\r\n");
        out.write("\t\t<th>uname</th>\r\n");
        out.write("\t  </tr>\r\n");
        out.write("\t  ");
        if (_jspx_meth_c_005fforEach_005f0(_jspx_page_context))
            return;
    }
}
```

It is kind of painfully difficult to read, but you can get some of the basic ideas of what is going on.

```
File Edit Options Buffers Tools Java Help
-|- useretable.jsp.java 35% L109 (Java//1 Abbrev) <V>
Mark set

    out.write("\r\n");
    out.write("\r\n");
    out.write("\r\n");
    out.write("<DOCTYPE html>\r\n");
    out.write("<html>\r\n");
    out.write("  <head>\r\n");
    out.write("    <title> ");
    out.print(request.getAttribute("title"));
    out.write("  </title>\r\n");
    out.write("  </head>\r\n");
    out.write("  <h3> Users </h3>\r\n");
    out.write("\r\n");
    out.write("    <div align='center'>\r\n");
    out.write("\t<table border='1' cellpadding='5'>\r\n");
    out.write("\t  <caption>Users Table</caption>\r\n");
    out.write("\t  <tr>\r\n");
    out.write("\t\t<th>uid</th>\r\n");
    out.write("\t\t<th>ctime</th>\r\n");
    out.write("\t\t<th>uname</th>\r\n");
    out.write("\t  </tr>\r\n");
    out.write("\t  ");
    if (_jspx_meth_c_005fforEach_005f0(_jspx_page_context))
        return;
}
```

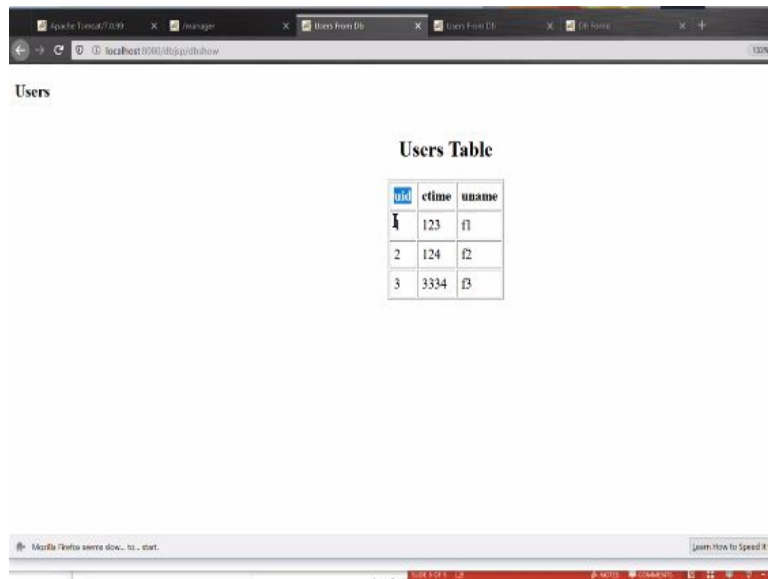
Eventually you are going to get all these out.write statements from the code above. Load it into the manager app.



During the conversion from database to Java can only be detected at runtime which is one of the more unfortunate aspects of this type of programming and to fix that what we will have to do is to see that this cannot be uid but it has to be user ID and then we go back and go to dbjsp, use the jar file which mentions META-INF in JSP. We only changed the config file.

There is no simple way to verify these kinds of things, although it does mean that somewhat tedious testing needs to be done every time. But using embedded tomcat testing can be done easily. We will see this in some upcoming sessions.

(Refer Slide Time: 38:23)



We learnt to successfully communicate to the database to show the users table. We will use this technique to execute complex statements in subsequent sessions.