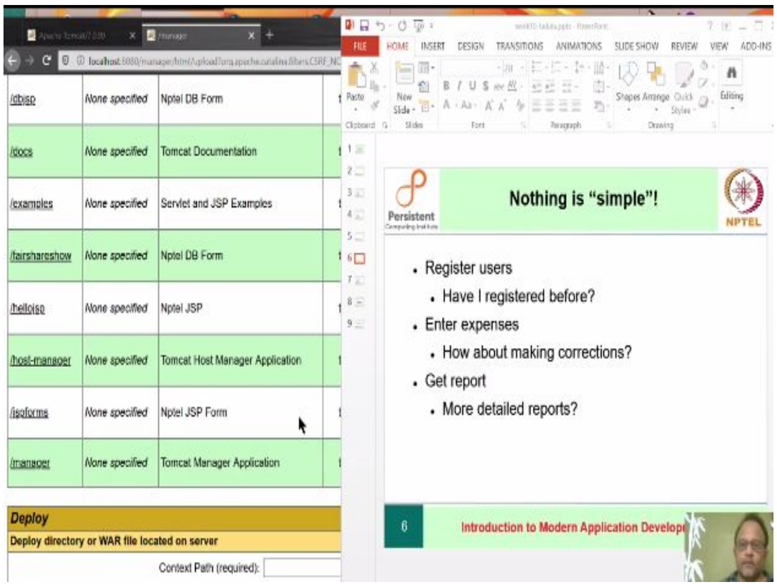
Introduction to Modern Application Development

Prof. Aamod Sane
FLAME University and Persistent Computing Institute
Indian Institute of Technology – Madras

Lecture – 31
Week 10 – Part 3

(Refer Slide Time: 00:11)



/dbjsp	None specified	Nptel DB Form
/docs	None specified	Tomcat Documentation
/examples	None specified	Servlet and JSP Examples
/fairshareshow	None specified	Nptel DB Form
/hellojsp	None specified	Nptel JSP
/host-manager	None specified	Tomcat Host Manager Application
/jspforms	None specified	Nptel JSP Form
/manager	None specified	Tomcat Manager Application

Deploy

Deploy directory or WAR file located on server

Context Path (required):

Nothing is "simple"!

- Register users
  - Have I registered before?
- Enter expenses
  - How about making corrections?
- Get report
  - More detailed reports?

6 Introduction to Modern Application Develop

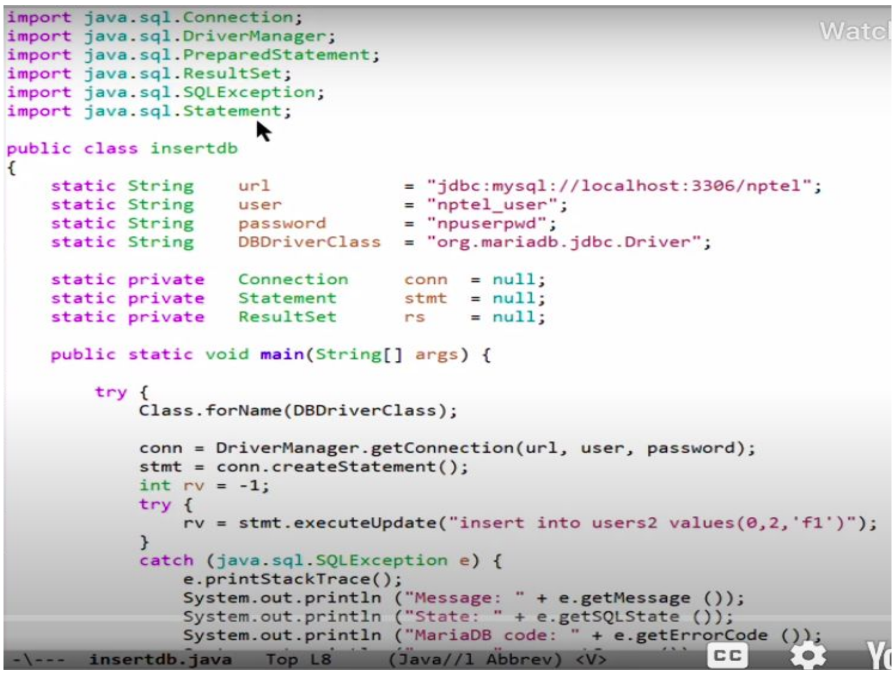
So far, we have created a screen with an input field and 3 buttons for registering, expenses and report. Register shows the registered users in the database. However these users were registered using MySQL from the command prompt.

The next step is to try to register a user from the browser instead which is a reasonable expectation for a user of the web application. Updating the database and dealing with user input brings up several new issues, such as;

1. Every time a new user is requested to be registered, check if the username already exists and they should choose a new username.
2. Enter expenses: If, by mistake, an extra or incorrect expense gets added to the expense table, an extra functionality of being able to edit or delete a previously entered row must be designed.

To achieve this, we are going to use a combination of database facilities and make certain changes to the existing app.

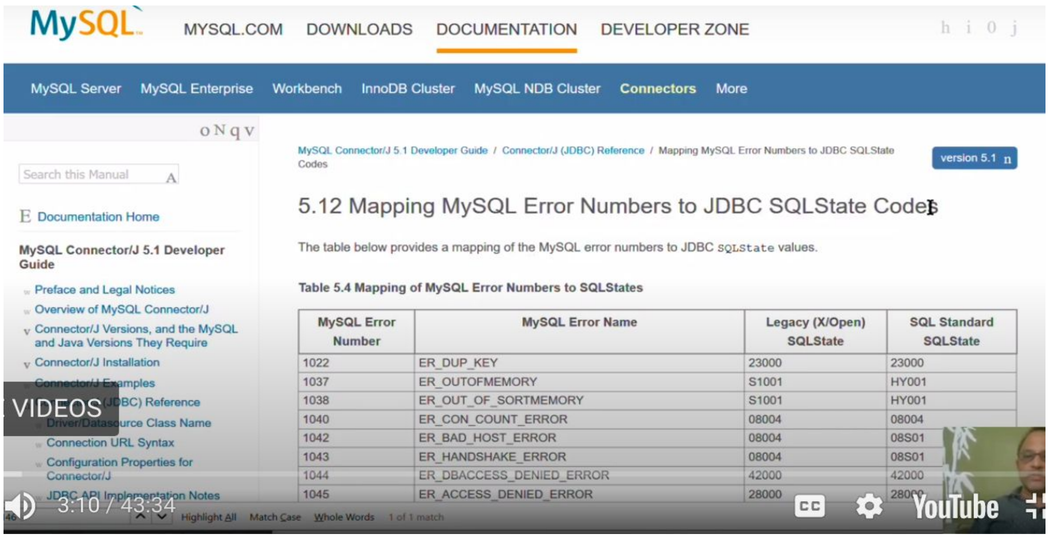
Consider the following simple test program: insertdb.java, where we try to insert records into the database.



```
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 insertdb
{
    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();
            int rv = -1;
            try {
                rv = stmt.executeUpdate("insert into users2 values(0,2,'f1')");
            }
            catch (java.sql.SQLException e) {
                e.printStackTrace();
                System.out.println ("Message: " + e.getMessage ());
                System.out.println ("State: " + e.getSQLState ());
                System.out.println ("MariaDB code: " + e.getErrorCode ());
-\---  insertdb.java    Top L8     (Java//l Abbrev) <V>
```

Most of the statements are familiar.

- We create a database connection using the database driver and statement has the query to be executed on the database nptel specified in the url using jdbc standard.
- The try block has `rv=stmt.executeUpdate("insert into users2 values(0,2,'f1')");`
- Though this is a simple update query, the important difference from earlier scenarios is errors might arise, since we are updating the database (unlike querying where we only read from the database). We need to catch exceptions using the standard SQLException object which has a lot of information.
- SQLException object has a message, SQL state, error code and a cause field. SQL state is a SQL language standard.

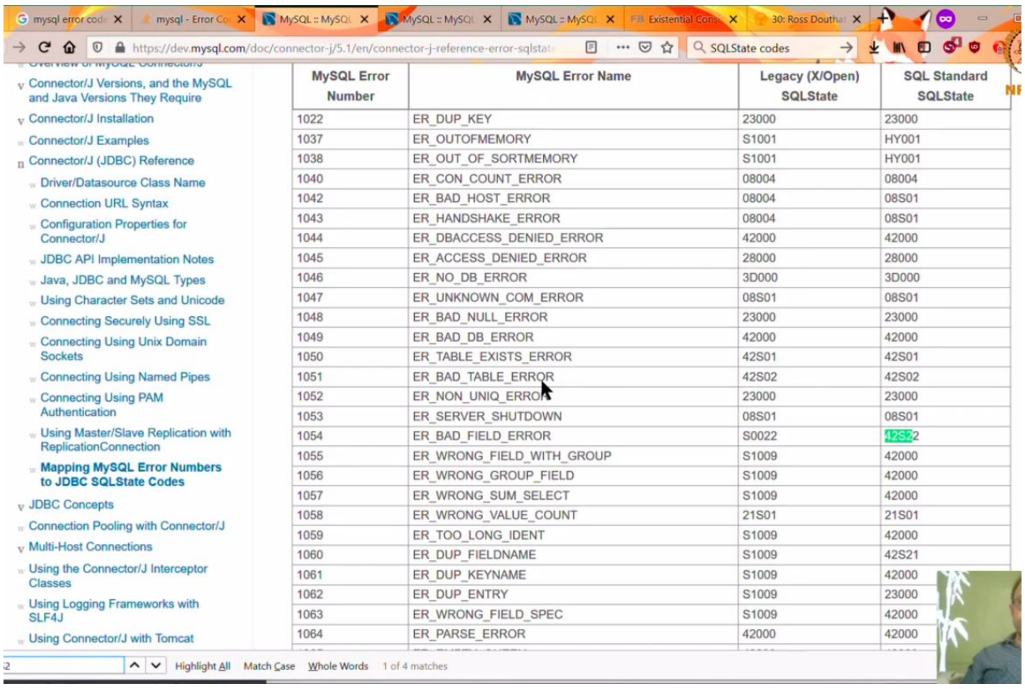


5.12 Mapping MySQL Error Numbers to JDBC SQLState Codes

The table below provides a mapping of the MySQL error numbers to JDBC SQLState values.

Table 5.4 Mapping of MySQL Error Numbers to SQLStates

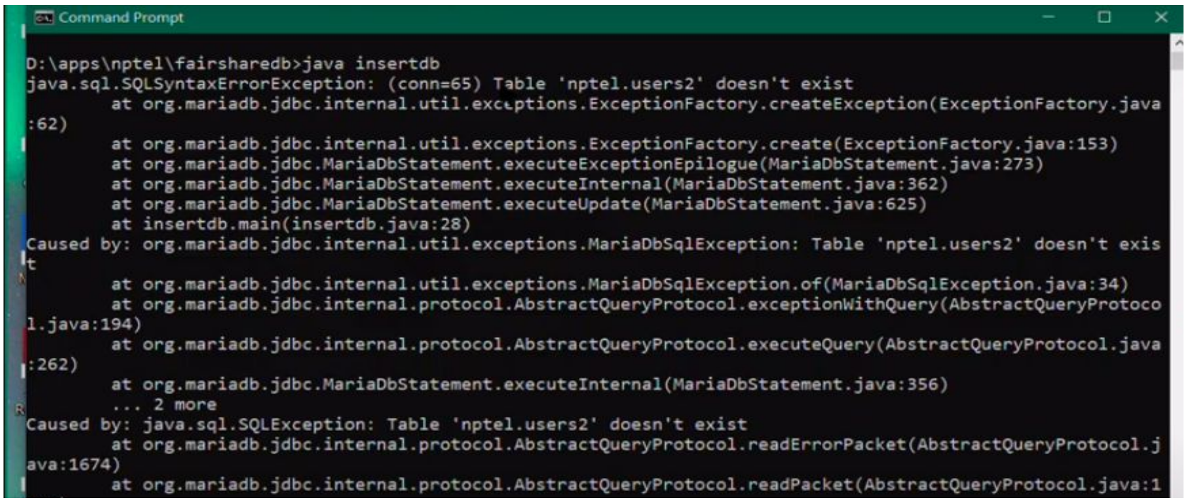
MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1022	ER_DUP_KEY	23000	23000
1037	ER_OUTOFMEMORY	S1001	HY001
1038	ER_OUT_OF_SORTMEMORY	S1001	HY001
1040	ER_CON_COUNT_ERROR	08004	08004
1042	ER_BAD_HOST_ERROR	08004	08S01
1043	ER_HANDSHAKE_ERROR	08004	08S01
1044	ER_DBACCESS_DENIED_ERROR	42000	42000
1045	ER_ACCESS_DENIED_ERROR	28000	28000



MySQL Error Number	MySQL Error Name	Legacy (X/Open) SQLState	SQL Standard SQLState
1022	ER_DUP_KEY	23000	23000
1037	ER_OUTOFMEMORY	S1001	HY001
1038	ER_OUT_OF_SORTMEMORY	S1001	HY001
1040	ER_CON_COUNT_ERROR	08004	08004
1042	ER_BAD_HOST_ERROR	08004	08S01
1043	ER_HANDSHAKE_ERROR	08004	08S01
1044	ER_DBACCESS_DENIED_ERROR	42000	42000
1045	ER_ACCESS_DENIED_ERROR	28000	28000
1046	ER_NO_DB_ERROR	3D000	3D000
1047	ER_UNKNOWN_COM_ERROR	08S01	08S01
1048	ER_BAD_NULL_ERROR	23000	23000
1049	ER_BAD_DB_ERROR	42000	42000
1050	ER_TABLE_EXISTS_ERROR	42S01	42S01
1051	ER_BAD_TABLE_ERROR	42S02	42S02
1052	ER_NON_UNIQ_ERROR	23000	23000
1053	ER_SERVER_SHUTDOWN	08S01	08S01
1054	ER_BAD_FIELD_ERROR	S0022	42S22
1055	ER_WRONG_FIELD_WITH_GROUP	S1009	42000
1056	ER_WRONG_GROUP_FIELD	S1009	42000
1057	ER_WRONG_SUM_SELECT	S1009	42000
1058	ER_WRONG_VALUE_COUNT	21S01	21S01
1059	ER_TOO_LONG_IDENT	S1009	42000
1060	ER_DUP_FIELDNAME	S1009	42S21
1061	ER_DUP_KEYNAME	S1009	42000
1062	ER_DUP_ENTRY	S1009	23000
1063	ER_WRONG_FIELD_SPEC	S1009	42000
1064	ER_PARSE_ERROR	42000	42000

- In the MySQL web page, we can find mapping of MySQL error numbers to SQL state code.
- We will encounter 42S02 in the webapp created till now, which ER_BAD_TABLE_ERROR.
- We print out the message and close the database.

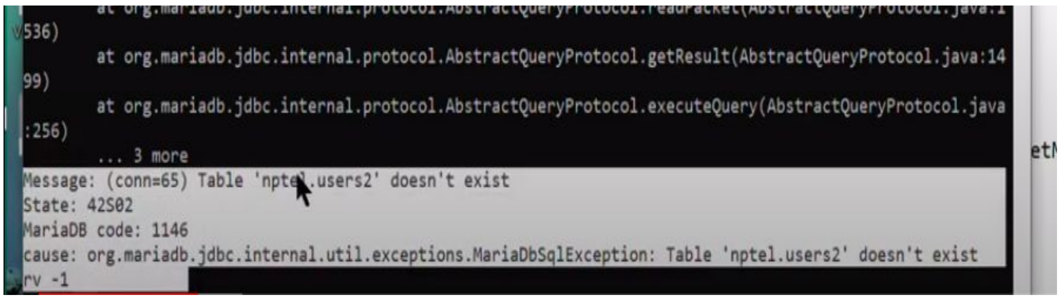
This is just a test program. Compile the insertdb.java file using `javac insertdb.java` and execute using `java insertdb`. We get this error message:



```
D:\apps\nptel\fairsharedb>java insertdb
java.sql.SQLSyntaxErrorException: (conn=65) Table 'nptel.users2' doesn't exist
        at org.mariadb.jdbc.internal.util.exceptions.ExceptionFactory.createException(ExceptionFactory.java
:62)
        at org.mariadb.jdbc.internal.util.exceptions.ExceptionFactory.create(ExceptionFactory.java:153)
        at org.mariadb.jdbc.MariaDbStatement.executeExceptionEpilogue(MariaDbStatement.java:273)
        at org.mariadb.jdbc.MariaDbStatement.executeInternal(MariaDbStatement.java:362)
        at org.mariadb.jdbc.MariaDbStatement.executeUpdate(MariaDbStatement.java:625)
        at insertdb.main(insertdb.java:28)
Caused by: org.mariadb.jdbc.internal.util.exceptions.MariaDbSqlException: Table 'nptel.users2' doesn't exis
t
        at org.mariadb.jdbc.internal.util.exceptions.MariaDbSqlException.of(MariaDbSqlException.java:34)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.exceptionWithQuery(AbstractQueryProtoco
l.java:194)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.executeQuery(AbstractQueryProtocol.java
:262)
        at org.mariadb.jdbc.MariaDbStatement.executeInternal(MariaDbStatement.java:356)
        ... 2 more
Caused by: java.sql.SQLException: Table 'nptel.users2' doesn't exist
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.readErrorPacket(AbstractQueryProtocol.j
ava:1674)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.readPacket(AbstractQueryProtocol.java:1
```

The error message: `Table 'nptel.users2' doesn't exist.`

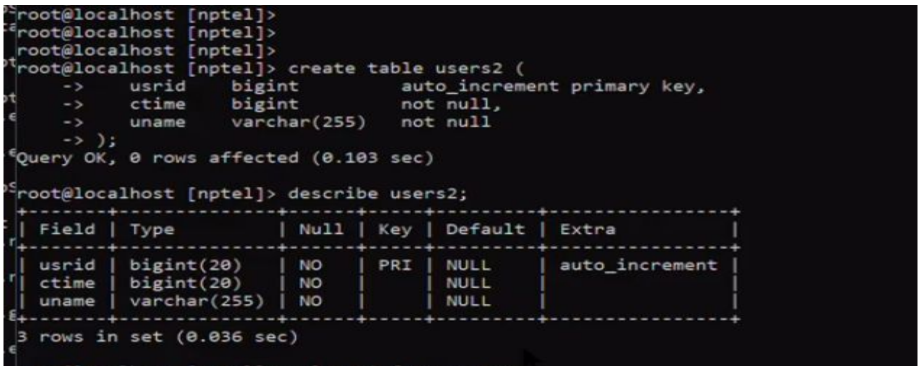
This is because we have not created it yet. This reason to run it before creating was to demonstrate the description of the error.



```
536)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.getResult(AbstractQueryProtocol.java:14
99)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.executeQuery(AbstractQueryProtocol.java
:256)
        ... 3 more
Message: (conn=65) Table 'nptel.users2' doesn't exist
State: 42S02
MariaDB code: 1146
cause: org.mariadb.jdbc.internal.util.exceptions.MariaDbSqlException: Table 'nptel.users2' doesn't exist
rv -1
```

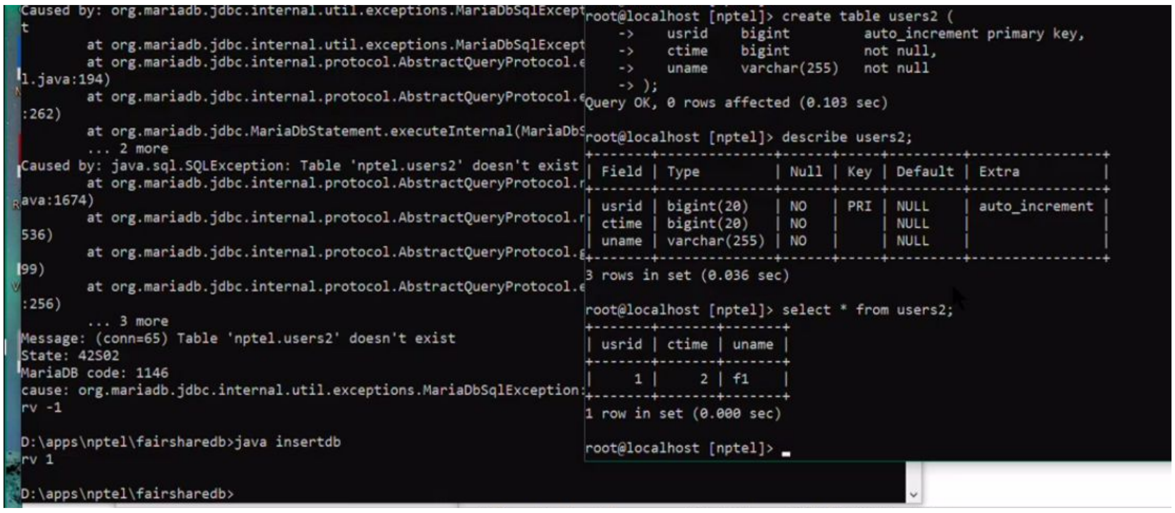
The highlighted text has a message field of SQLException object. The state is a standard from SQL, value is 42S02 as predicted. MariaDB's MySQL also has status codes that explain the type of error, in this case its value is 1146.

To fix the error, create a table users2.



```
root@localhost [nptel]>
root@localhost [nptel]>
root@localhost [nptel]>
root@localhost [nptel]> create table users2 (
    ->     usrid    bigint          auto_increment primary key,
    ->     ctime    bigint          not null,
    ->     uname    varchar(255)    not null
    -> );
Query OK, 0 rows affected (0.103 sec)

root@localhost [nptel]> describe users2;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| usrid | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| ctime | bigint(20)   | NO   |     | NULL    |                |
| uname | varchar(255) | NO   |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
3 rows in set (0.036 sec)
```

Now run insertdb using `java insertdb`, it outputs rv=1. This is the number of rows in users2. This means insertion query in the insertdb.java was successful.

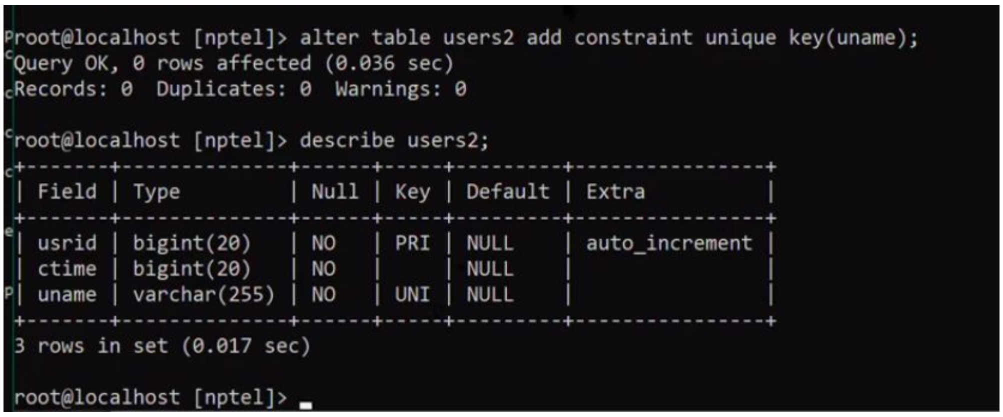


```
Caused by: org.mariadb.jdbc.internal.util.exceptions.MariaDbSqlExcept
t
        at org.mariadb.jdbc.internal.util.exceptions.MariaDbSqlExcept
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.e
1.java:194)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.e
:262)
        at org.mariadb.jdbc.MariaDbStatement.executeInternal(MariaDbS
        ... 2 more
Caused by: java.sql.SQLException: Table 'nptel.users2' doesn't exist
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.r
ava:1674)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.r
536)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.g
99)
        at org.mariadb.jdbc.internal.protocol.AbstractQueryProtocol.e
:256)
        ... 3 more
Message: (conn=65) Table 'nptel.users2' doesn't exist
State: 42S02
MariaDB code: 1146
cause: org.mariadb.jdbc.internal.util.exceptions.MariaDbSqlException:
rv -1

D:\apps\nptel\fairsharedb>java insertdb
rv 1

D:\apps\nptel\fairsharedb>
```

```
root@localhost [nptel]> create table users2 (
    ->    usrid    bigint          auto_increment primary key,
    ->    ctime    bigint          not null,
    ->    uname    varchar(255)    not null
    -> );
Query OK, 0 rows affected (0.103 sec)

root@localhost [nptel]> describe users2;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| usrid | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| ctime | bigint(20)   | NO   |     | NULL    |                |
| uname | varchar(255) | NO   |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
3 rows in set (0.036 sec)

root@localhost [nptel]> select * from users2;
+-------+-------+-------+
| usrid | ctime | uname |
+-------+-------+-------+
|     1 |     2 | f1    |
+-------+-------+-------+
1 row in set (0.000 sec)

root@localhost [nptel]>
```

The left command prompt executes insertdb after table users2 is created, the updated table is displayed on the right command prompt as output for the query `select * from users2`.

To ensure that one cannot enter the same username is using `unique constraint`, just like primary keys are required to be unique, we can also ensure that the user name is unique.



```
root@localhost [nptel]> alter table users2 add constraint unique key(uname);
Query OK, 0 rows affected (0.036 sec)
Records: 0  Duplicates: 0  Warnings: 0

root@localhost [nptel]> describe users2;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| usrid | bigint(20)   | NO   | PRI | NULL    | auto_increment |
| ctime | bigint(20)   | NO   |     | NULL    |                |
| uname | varchar(255) | NO   | UNI | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
3 rows in set (0.017 sec)

root@localhost [nptel]>
```

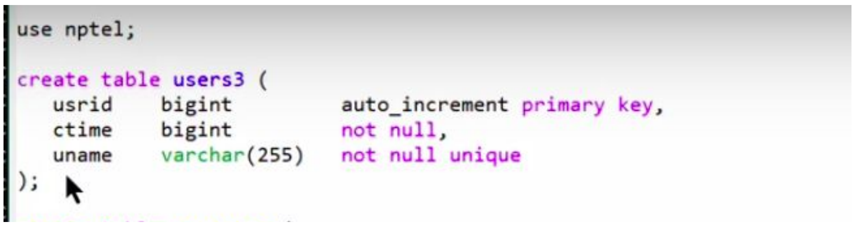
Now if we attempt to insert the same row, there is an exception which is that the duplicate key f1 is entered for uname and this time the status code changes to 23,000 and the mariadb code is 1062. 23000 is the standard error code for ER_DUP_ENTRY.

The usual way to fix errors is to look up the status code (here, ER_DUP_ENTRY which means that a duplicate entry has been made).

TIP:

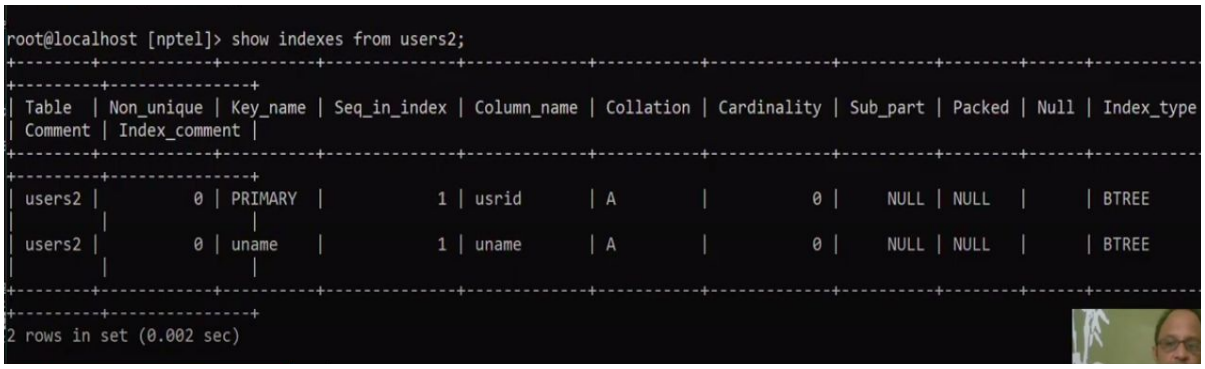
When we use websites, almost all of them require users to register and identify themselves since name, city, etc., is not necessarily unique or it can also be the case that somebody might easily impersonate you knowing your details. So instead, we create a new identity or you use an identity which is already known to be unique such as Yahoo email address where some third party like Yahoo or Google have taken care that the email ids are unique to finally reach a stable notion of user identity for your app.

In the previous code, we created a table without a unique constraint and then added it by altering it. The alternative is to use this unique declaration right in the schema of the table as shown below.



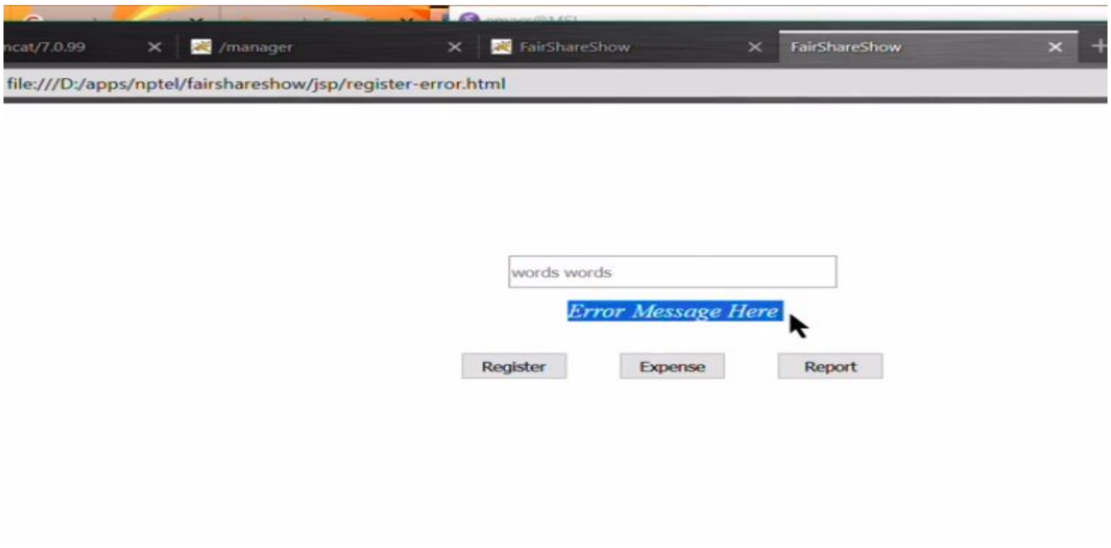
```
use nptel;

create table users3 (
   usrid    bigint         auto_increment primary key,
   ctime    bigint         not null,
   uname    varchar(255)   not null unique
);
```

So for example if you want to see how this uniqueness is achieved use `show indexes from users2;`



```
root@localhost [nptel]> show indexes from users2;
+--------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+-----------
+---------+---------------+
| Table  | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type
| Comment | Index_comment |
+--------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+-----------
+---------+---------------+
| users2 |          0 | PRIMARY  |            1 | usrid       | A         |           0 |     NULL | NULL   |      | BTREE
|         |               |
| users2 |          0 | uname    |            1 | uname       | A         |           0 |     NULL | NULL   |      | BTREE
|         |               |
+--------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+------+-----------
+---------+---------------+
2 rows in set (0.002 sec)
```

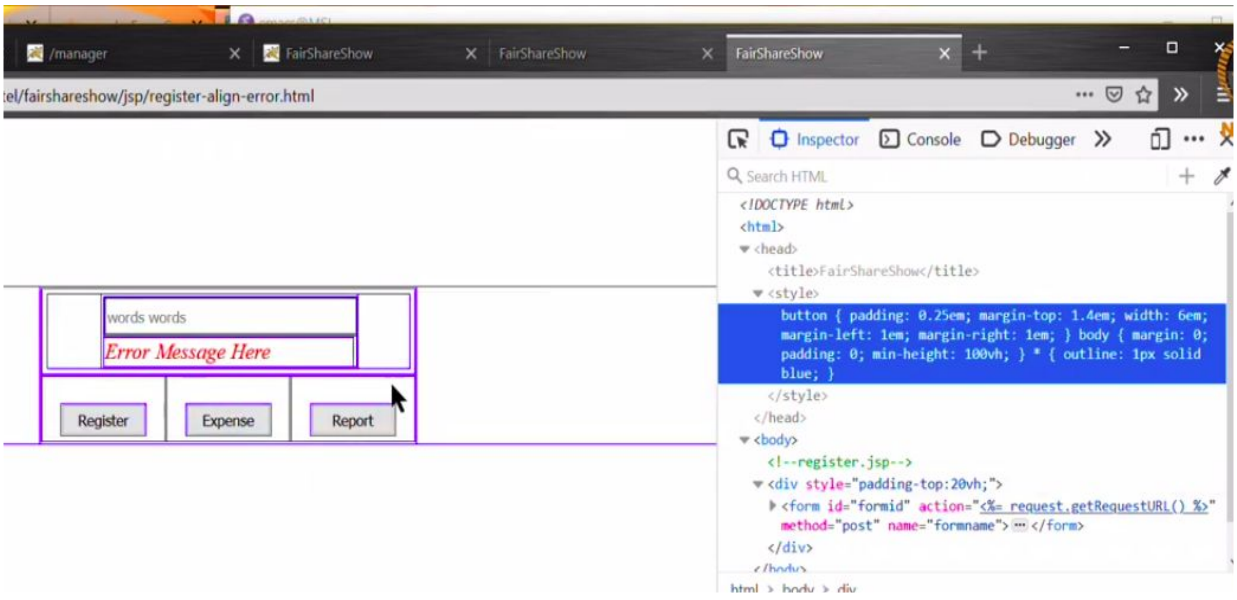
Register page of the website needs to have the functionality of displaying errors. We instead create a new screen with a different url for displaying queries which lead to an error.


file:///D:/apps/nptel/fairshareshow/jsp/register-error.html

The right place to return an error like this is a matter of UI design.

In the register page itself, we have the input field which corresponds to the input field shown here, but in the register error page we add another row(highlighted in the above figure) besides the input field.

A designer might prefer the error message to be centered or aligned to the left hand side of the input field. All these effects can be achieved, but it is ugly to achieve just with tables. CSS is better but it needs a lot more knowledge of CSS than what we have learnt.


el/fairshareshow/jsp/register-align-error.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>FairShareShow</title>
    <style>
      button { padding: 0.25em; margin-top: 1.4em; width: 6em;
      margin-left: 1em; margin-right: 1em; } body { margin: 0;
      padding: 0; min-height: 100vh; } * { outline: 1px solid
      blue; }
    </style>
  </head>
  <body>
    <!--register.jsp-->
    <div style="padding-top:20vh;">
      <form id="formid" action="<%= request.getRequestURL() %>"
      method="post" name="formname"> ··· </form>
    </div>
  </body>
html > body > div
```

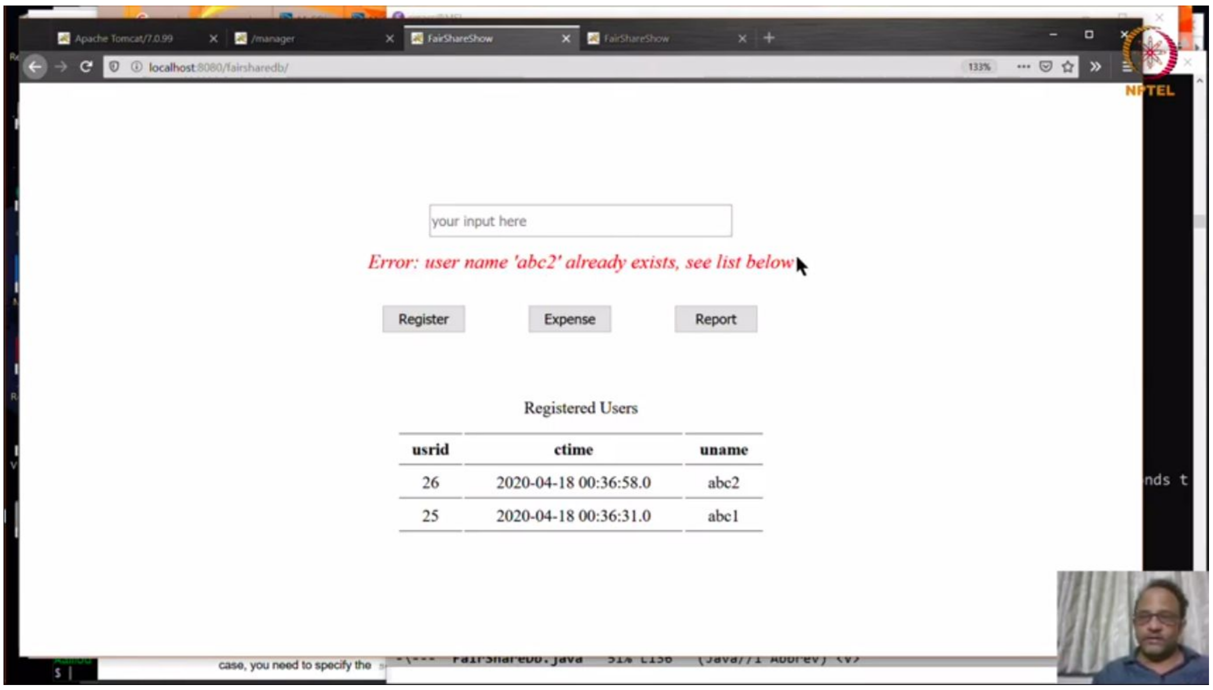
The error page with an extra style called `outline:1px solid blue`. This is done to show

how the first row has a table with 2 rows, so there is a table inside a table.

As we know the primary part of the structure is a row of buttons, a row with the input field, but then this input field in turn actually consists of a table which itself contains the input field and the second row has the error message text. Doing this and sizing them can be tedious.

CSS does make it a lot easier to create applications which work for mobiles, tablets as well as normal websites.

User interface gets affected based on the input to the input field. The user interface becomes more dynamic as the app has to detect the presence or absence of errors and output different html files.
We want to see the following output when an already created user abc2 is registered again:

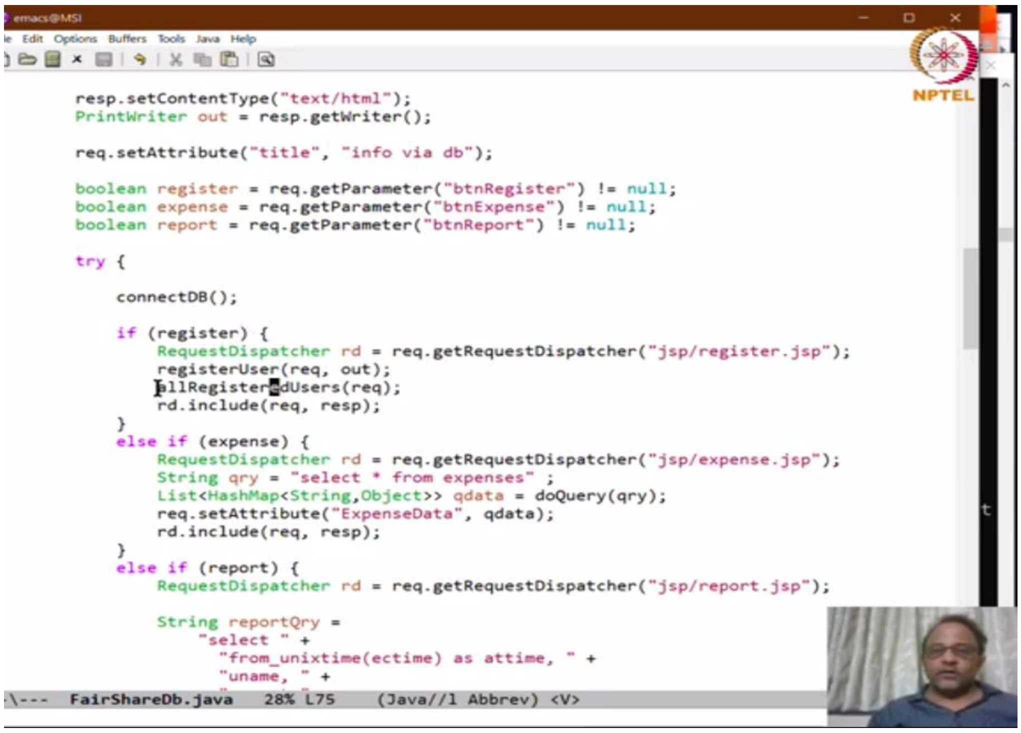


Error: user name 'abc2' already exists, see list below

Registered Users

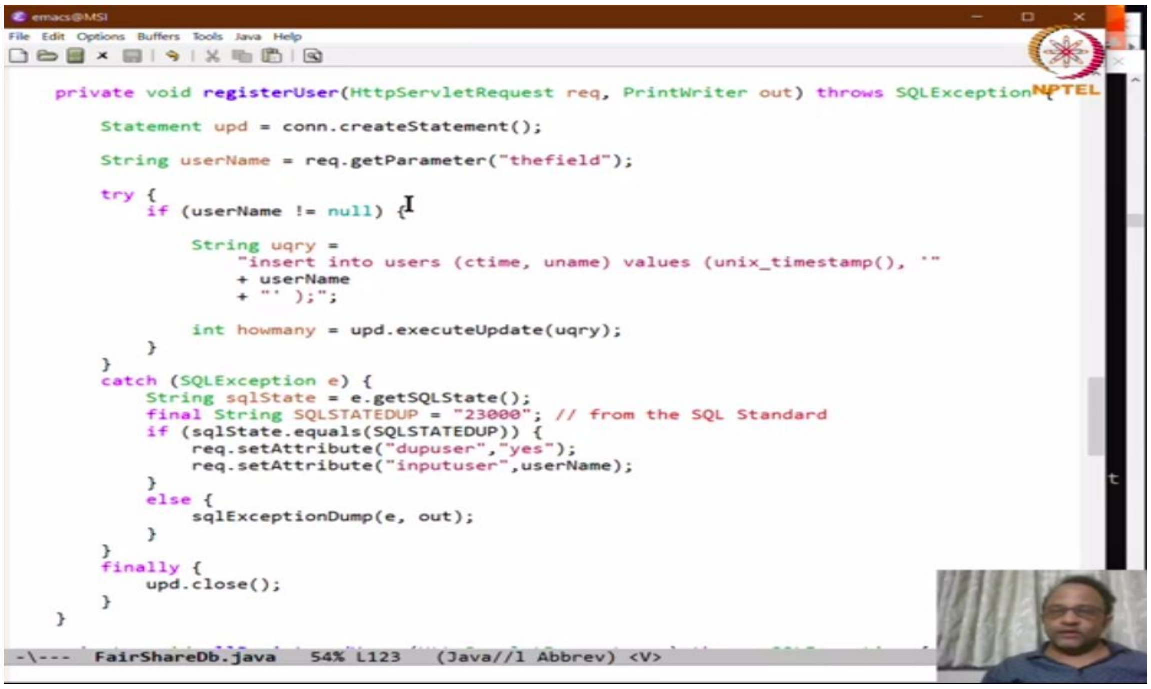
usrid	ctime	uname
26	2020-04-18 00:36:58.0	abc2
25	2020-04-18 00:36:31.0	abc1

We are going to add two things to the FairShareDb.java file:



```
    resp.setContentType("text/html");
    PrintWriter out = resp.getWriter();

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

    boolean register = req.getParameter("btnRegister") != null;
    boolean expense = req.getParameter("btnExpense") != null;
    boolean report = req.getParameter("btnReport") != null;

    try {

        connectDB();

        if (register) {
            RequestDispatcher rd = req.getRequestDispatcher("jsp/register.jsp");
            registerUser(req, out);
            allRegisteredUsers(req);
            rd.include(req, resp);
        }
        else if (expense) {
            RequestDispatcher rd = req.getRequestDispatcher("jsp/expense.jsp");
            String qry = "select * from expenses" ;
            List<HashMap<String,Object>> qdata = doQuery(qry);
            req.setAttribute("ExpenseData", qdata);
            rd.include(req, resp);
        }
        else if (report) {
            RequestDispatcher rd = req.getRequestDispatcher("jsp/report.jsp");

            String reportQry =
                "select " +
                 "from_unixtime(ectime) as attime, " +
                 "uname, " +
-\---  FairShareDb.java    28% L75     (Java//l Abbrev) <V>
```

- The update part is done in a function called `registerUser(req,out)` which takes the request and the output. This output is the PrintWriter which we get from the response object.
- Get all the registered users in the query part and this is included in the template as we have seen before.



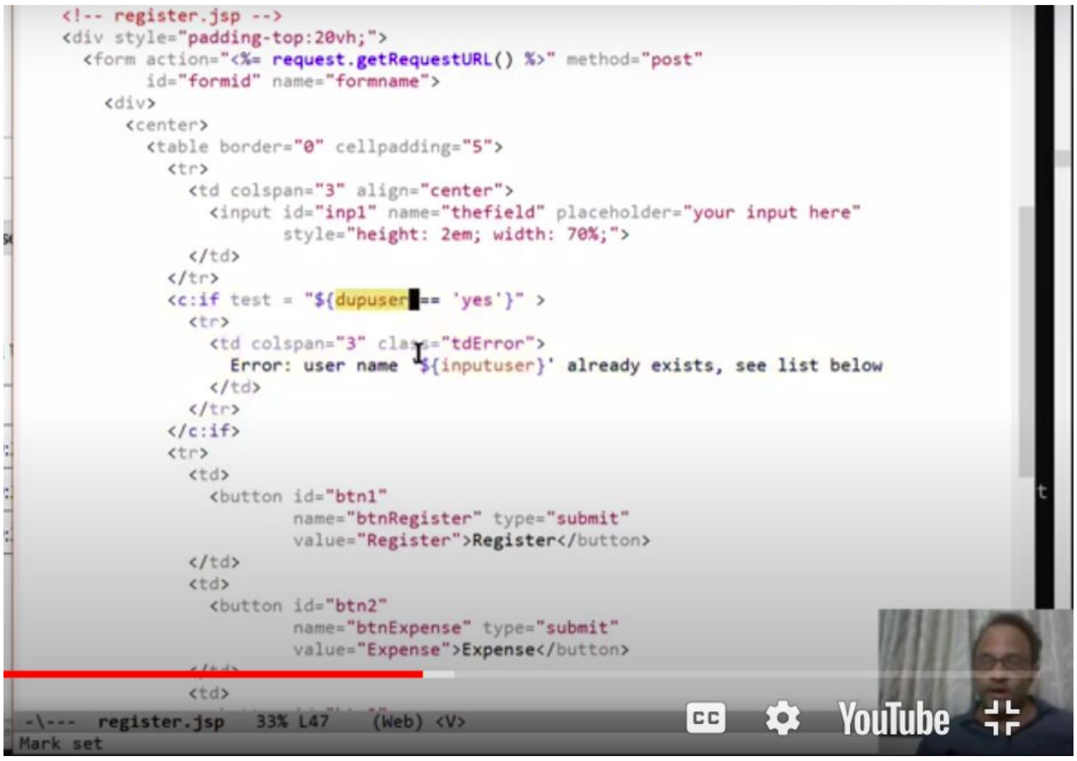
```
    private void registerUser(HttpServletRequest req, PrintWriter out) throws SQLException {

        Statement upd = conn.createStatement();

        String userName = req.getParameter("thefield");

        try {
            if (userName != null) {

                String uqry =
                    "insert into users (ctime, uname) values (unix_timestamp(), '"
                    + userName
                    + "' );";

                int howmany = upd.executeUpdate(uqry);
            }
        }
        catch (SQLException e) {
            String sqlState = e.getSQLState();
            final String SQLSTATEDUP = "23000"; // from the SQL Standard
            if (sqlState.equals(SQLSTATEDUP)) {
                req.setAttribute("dupuser","yes");
                req.setAttribute("inputuser",userName);
            }
            else {
                sqlExceptionDump(e, out);
            }
        }
        finally {
            upd.close();
        }
    }
-\---  FairShareDb.java    54% L123    (Java//l Abbrev) <V>
```

The unix timestamp gets evaluated on the server side, this is a SQL function: unix_timestamp(). `howmany` field stores the number of affected rows from the previously executed register query.

When there is an exception, SQL state is checked for duplicate entry using `sqlState.equals(SQLSTATEDUP)`. We are going to set the attribute in the request that says "yes", there is a duplicate user and we are also going to send the username that is attempted to be duplicated, so that the error message can contain what user exactly was duplicated.

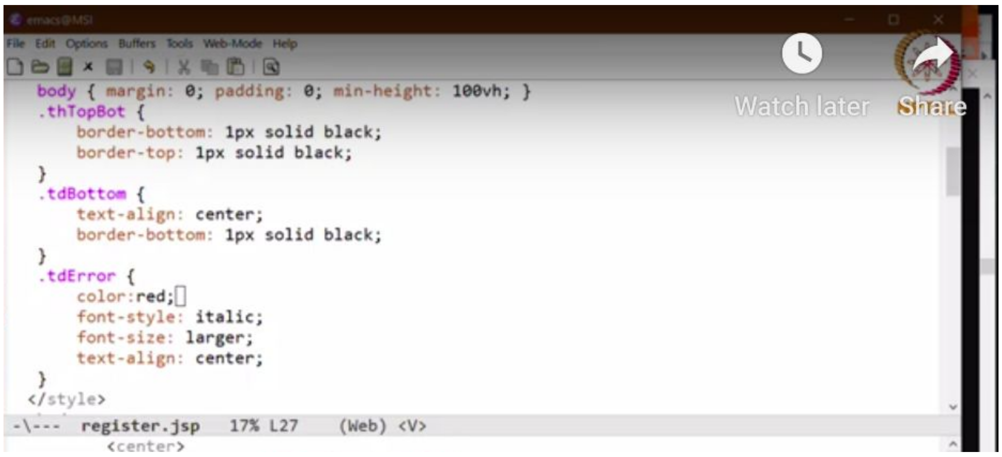
If the SQL state code is something else, we are simply going to dump the SQL exception and then close the statement. We are going to take action with duplicate entry error types.

Execute the query and result is a list of maps, set the user data attribute and go to the register.jsp function. One novelty in here is that this template does 2 different things based on whether there was a duplicate user or not. As we saw, the goal is to add a row containing the error message only if there is a duplicate user and if there is such a user, then you should mention what this is.



```
<!-- register.jsp -->
<div style="padding-top:20vh;">
  <form action="<%= request.getRequestURL() %>" method="post"
        id="formid" name="formname">
    <div>
      <center>
        <table border="0" cellpadding="5">
          <tr>
            <td colspan="3" align="center">
              <input id="inp1" name="thefield" placeholder="your input here"
                     style="height: 2em; width: 70%;">
            </td>
          </tr>
          <c:if test = "${dupuser == 'yes'}" >
            <tr>
              <td colspan="3" class="tdError">
                Error: user name '${inputuser}' already exists, see list below
              </td>
            </tr>
          </c:if>
          <tr>
            <td>
              <button id="btn1"
                      name="btnRegister" type="submit"
                      value="Register">Register</button>
            </td>
            <td>
              <button id="btn2"
                      name="btnExpense" type="submit"
                      value="Expense">Expense</button>
            </td>
            <td>
-\---  register.jsp    33% L47    (Web) <V>
Mark set
```

You might have noticed that the error was printed in red colored text using CSS class: tdError.



```
body { margin: 0; padding: 0; min-height: 100vh; }
.thTopBot {
    border-bottom: 1px solid black;
    border-top: 1px solid black;
}
.tdBottom {
    text-align: center;
    border-bottom: 1px solid black;
}
.tdError {
    color:red;
    font-style: italic;
    font-size: larger;
    text-align: center;
}
</style>
-\---  register.jsp    17% L27    (Web) <V>
        <center>
```

tdError class: Color should be red, font style is italic, font size is larger i.e. just larger than default font, align to the center. The above two CSS classes were used to get the table layout from last time.

Validation starts becoming very important when updations are made. To demonstrate, click register user with no input entered in the input field. Since username !=null is present in the code, if there is no username, there is a new row in the table with a blank name though the expected behavior as an end user of the website might be to not add any row to the table.

There is much more to user validation than the simple example. To demonstrate this, we first make a change to reject the empty string if the username is empty.

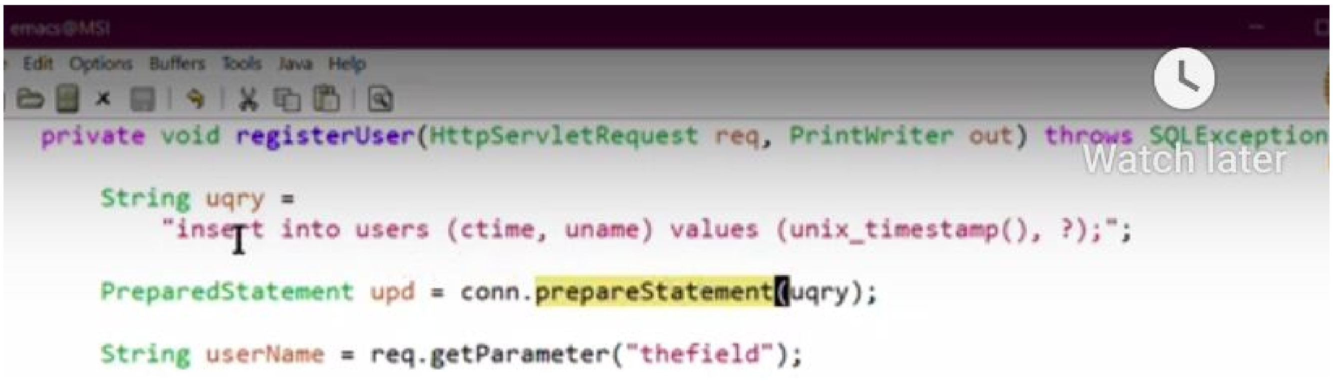
We need to ensure nothing fails when the table is empty, but our website displays an error.

The testing of a database should be automated but it is a characteristic feature of modern web development where most of them are tested in some version.

In our case, we could use embedded tomcat but there are many ways to automate. Once changing, entering and checking the details starts becoming the feature of your application, life does get considerably harder. Even adding a single quote(‘) to the input throws an exception which our app cannot handle, it just dumps the stack trace. So just a bad input is hard to handle.

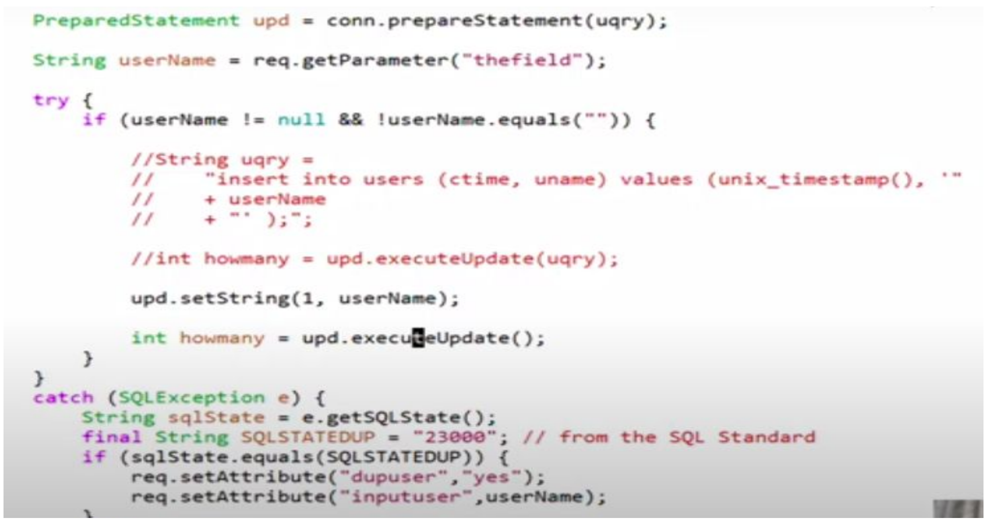
We use the concept of prepared statements to achieve better testing. Under certain circumstances they can be more efficient as well, especially when used in loops. It specifies the database which part of the statement is specified by developer and which part is coming as input from the users, so that the database will treat it not as something that whose syntax the database has to understand, it will clean the isolate user input and then store it separately without letting it affect the rest of the statement.

In our example, nothing is irretrievably broken, only the request gets affected . So we can reload the page to erase the previous execution.



```
private void registerUser(HttpServletRequest req, PrintWriter out) throws SQLException

    String uqry =
        "insert into users (ctime, uname) values (unix_timestamp(), ?);";

    PreparedStatement upd = conn.prepareStatement(uqry);

    String userName = req.getParameter("thefield");
```

Instead of creating just a statement, we are going to create a prepared statement where at the same place the query is also supplied. The prepared statement will contact the database, compile the prepared statement, this is another reason because of pre-compilation that prepared statements can be more efficient and in the prepared statement we are going to leave a hole, a parameter where the user value will be plugged in. The usual way to do that is to have a ? to say that this is where we are going to plug in a value. The rest remains the same as before.



```
PreparedStatement upd = conn.prepareStatement(uqry);

String userName = req.getParameter("thefield");

try {
    if (userName != null && !userName.equals("")) {

        //String uqry =
        //    "insert into users (ctime, uname) values (unix_timestamp(), '"
        //    + userName
        //    + "' );";

        //int howmany = upd.executeUpdate(uqry);

        upd.setString(1, userName);

        int howmany = upd.executeUpdate();
    }
}
catch (SQLException e) {
    String sqlState = e.getSQLState();
    final String SQLSTATEDUP = "23000"; // from the SQL Standard
    if (sqlState.equals(SQLSTATEDUP)) {
        req.setAttribute("dupuser","yes");
        req.setAttribute("inputuser",userName);
```

So you get the `userName` as before, complete your validation, and then say `upd.setString(1, username );`and because the query is already known all you have to do is execute the update as shown above.

The idea is you do not specify a bad input looks like, you only specify what a good input is, So even if the database, the standard SQL might not approve when written down in a string syntax,

You can use the same idea for example for expenses. At this point we will actually have nearly a full database app in which the only thing that is different is that different users cannot access the system at the same time just yet.

So at this point we you have a reasonably complete app.

We however do not have delete operations such as removing a user but it is no different from what we have done so far. Just as we did register, create another button called unregister which makes the system delete the row from the database with the provided user name.

Once we add login and cookies, our app will essentially be complete and you will have seen all the pieces including various debugging scenarios that show up when a web app has to be developed.