SQL database Management using Node.Js

Assignment No.: 2

Course Code: ASE-9476

Course Name: Factory Information Systems

Submitted By:

Usamah (267590)

Md. Shahedul Alam (267943)

Prasun Biswas (267948)

Environment

- Node.js
- MySQL
- WebStorm IDE

Objective:

- **Understanding Basics of MySQL Database.**
- **Using Node.js to create a connection between the HTML page and the Database**

- **Database Description:**
  Our main task was to create a database using MySQL. Database contained Tables such as Products, Orders, Customers.

  Customer's Table: Here id column stand for customer id. Which is used in another table also for cross referencing. Name is for Customer name and the rest of the columns are evident from the column name.

id	name	address	postalcode
10001	Usama	C_253	33720
10002	Shaheed	C_253	33720
10003	Prasun	A_98	33720

Fig: Customers Table

  Order's Table: In this table we have included order_id which is unique for this table and also a primary key. The table ID is common with customer's table, which can be used for retrieving data from HTML form.

order_id	Product_id	id	Date
100	1	10001	2017-03-08
101	10	10002	2017-03-07
102	11	10003	2017-03-01

Fig: Order's Table

Product's Table: Here we have incorporated product name, product id and status of product which can be manufactured and shipped, In production and ordered. Point to be noted is that product id is common with order's table

product_name	Product_id	Name	Status
Product_A	1	Usama	ordered
Product_B	10	Shaheed	in_production
Product_C	11	Prasun	manufactured_and_shipped

Fig: Product's Table

- **Client Ordering Form:**
  HTML is used to implement the client side ordering form. The form is such where a client can enter his/her information and order products according to wish.

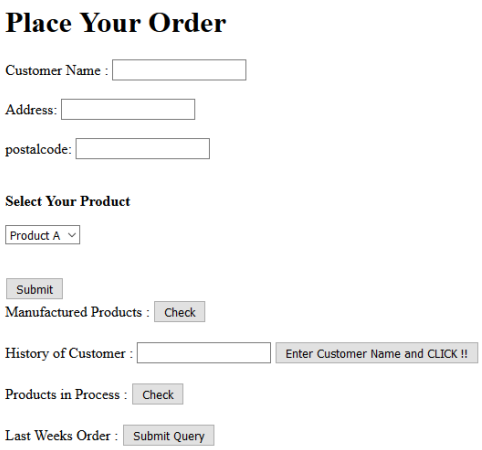

Fig: Client Order Interface

Code for HTML form:

```
<!Doctype HTML>
<HTML>
<head>
<script>
   function validate(){
   }
</script>

</head>
<body>

<form method = POST,
action="127.0.0.1:3001\submit">
<h1> Place Your Order</h1>
<p1>Customer Name : <input type= "txt"
name= "custm.name">
</p1>
<br>
<br>
<p2> Address: <input type="txt"
name="address"> </p2>
<br>
<br>
   postalcode: <input type="txt" name=
"postalcode">
   <br>
   <br>
  <p> <b> Select Your Product </b> <p/>
   <select>

  <option value='Product A'> Product A
</option>
      <option value='Product B'> Product B
</option>
      <option value='Product C'> Product C
</option>

   </select>
</form>
   <br>
   <input type="submit" name="submit"
value="Submit" onclick="return validate() "
/>
<br>
   <form target =_blank action="/check1"
method="GET">
      Manufactured Products : <input
type="submit" name="check1 "
value="Check"/>
   </form>
<br>
   <form target =_blank action="/check2"
method="POST">
      History of Customer : <input type=
"text" name= "custm.name1"> <input
type="submit" name="check2 " value= "Enter
Customer Name and CLICK !!"  />
   </form>
   <br>
   <form target =_blank action="/check3"
method="GET">
      Products in Process : <input
type="submit" name="check3 "
value="Check"  />
   </form>
   <br>
   <form target =_blank action="/check4"
method="GET">
      Last Weeks Order : <input type=
"submit" name= "custm.name2">
   </form>
   <br>
</form>
</body>
</HTML>
```

- **Application Development Process in Node.Js**: afis.js is the application that is implemented in Node.Js platform for implementing the server and checking the validation of xml file. The development process is given below:

```
var express    = require('express'),
    mysql      = require('mysql'),
    bodyParser = require('body-parser'),
    appRootDir = require('app-root-dir');
const path = require('path');
var formidable = require('formidable');
var json2html =  require('json-to-htmltable');
```

Next we create a connection between database and node.js server. Here the name, username and passwords of the database is mentioned.

```
var connection = mysql.createConnection({
    host     : 'localhost',
    user     : 'root',
    password : 'usama999',
    database : 'fis2'
});
connection.connect(function(error){
    if(!error){
        console.log('Successfully connected to database')
    }
        else
    {
        console.log('Error while connecting to the database')
```

- Then our HTML form is loaded.

```
app.get('/', function(req, res) {
    res.sendFile(path.join(__dirname + '/test.html'));
    });
console.log(app.method);
```

- Next we create a query for products which has been produced already with this code

```
app.get('/check1', function(req, res) {
    connection.query("Select * from products where Status=
'manufactured_and_shipped' ", function (error, results, fields) {
        if (error) {
            console.log("Error while performing query")
        }

        console.log(results);
        var obj = JSON.stringify(results);
        console.log(obj);
        var table = json2html(obj);
        res.write(table)
        res.end();
```

- Here we have created an array, through that array we push one value after another. This part of the code is for retrieving Customer's order history. We have used Json.stringify to get the results in a table and organized form.

```
app.post('/check2', function(req, res) {
var fieldvalues=[];
        var form =new formidable.IncomingForm().parse(req).on('field',
function (name, value) {
            fieldvalues.push(value);            })
            .on('end', function () {
                console.log('Field Values Are:');
                console.log(fieldvalues);
                var sql="Select * from customers INNER JOIN Orders On
Orders.id=customers.id INNER JOIN Products On
Orders.Product_id=Products.Product_id WHERE Customers.Name=?" ;
                var values = [fieldvalues[0
connection.query(sql,[values], function (error, results){
if(error)
                    {console.log('Errror while performing query');
                    }
                    var obj = JSON.stringify(results);
                    var table = json2html(obj);
                    res.write(table);
                    res.end();
                });
```

- This part of the code is to retrieve information for products which are in process, that is still in production.

```
app.get('/check3', function(req, res) {
    connection.query("Select * from products where Status=
'in_production' ", function (error, results, fields) {
        if (error) {
            console.log("Error while performing query")
        }
        console.log(results);
        var obj = JSON.stringify(results);
        console.log(obj);
        var table = json2html(obj);
        res.write(table)
        res.end();
    });
    connection.query("Select * from Products", function (error, results,
fields) {
            if (error) {
                console.log("Error while performing query")
            }
            console.log(results);
            var obj = JSON.stringify(results);
            console.log(obj);
            var table = json2html(obj);
            res.write(table)
            res.end();
```

- We were asked to design the system in such a way that we can retract datae like last week's order.

```
app.get('/check4', function(req, res) {
    connection.query("SELECT Customers. ID, Customers.Name FROM Customers
INNER JOIN Orders ON Orders. ID=Customers. ID WHERE  Orders.date BETWEEN
date_sub(now(),INTERVAL 1 WEEK) and now(); ",
        function (error, results, fields) {
        if (error) {
            console.log("Error while performing query")
        }
        console.log(results);
        var obj = JSON.stringify(results);
        console.log(obj);
        var table = json2html(obj);
        res.write(table)
        res.end()
```

Description of Queries :

- **For listing all the products manufactured-**
  Select * from products where Status= 'manufactured_and_shipped'
- **For listing Products that are in Production and Customer's information-**

  "Select * from products where Status= 'in_production' "

Here by the command INNER JOIN we are collecting relevant data from different tables and cross referencing them.

- **Customer's History**
  Select * from customers INNER JOIN Orders On Orders.id=customers.id INNER JOIN Products On Orders.Product_id=Products.Product_id WHERE Customers.Name=?"

- **Order from Last Week**
  "SELECT Customers. ID, Customers.Name FROM Customers INNER JOIN Orders ON Orders. ID=Customers. ID WHERE Orders.date BETWEEN date_sub(now(),INTERVAL 1 WEEK) and now(); "

  Here we have used the Date data type so we can find out last orders by subtracting number of days from the current date to find out previous orders.

- **Involvement of each group member to the assignment:** For this assignment, we had to go through node.js intensively. This assignment took a toll on us. Still we are not that proficient in node.js as we all are from Electrical background. We had to study, discuss about the assignment and watch tutorials regarding node.js. Also,we had to develop this program all together because sometimes what happens is one who is writing the code makes some syntax error which is actually very simple but doesn't comes easily to him. SO,the other member of the team had to intervene. Also,we had to go through different npm packages and their documentation and how they worked as there were extensive amount of information. So, everyone had to chip in. This assignment would have been impossible without the effort of every team member.

Video Links:

1. Assignment Video
   https://drive.google.com/open?id=0B69G-aUYIZtpNUIwZnN5M3Q4eEk

2. SQL Database Dumping error video
   https://drive.google.com/open?id=0B69G-aUYIZtpSFNhZnZ2U0NyUEk