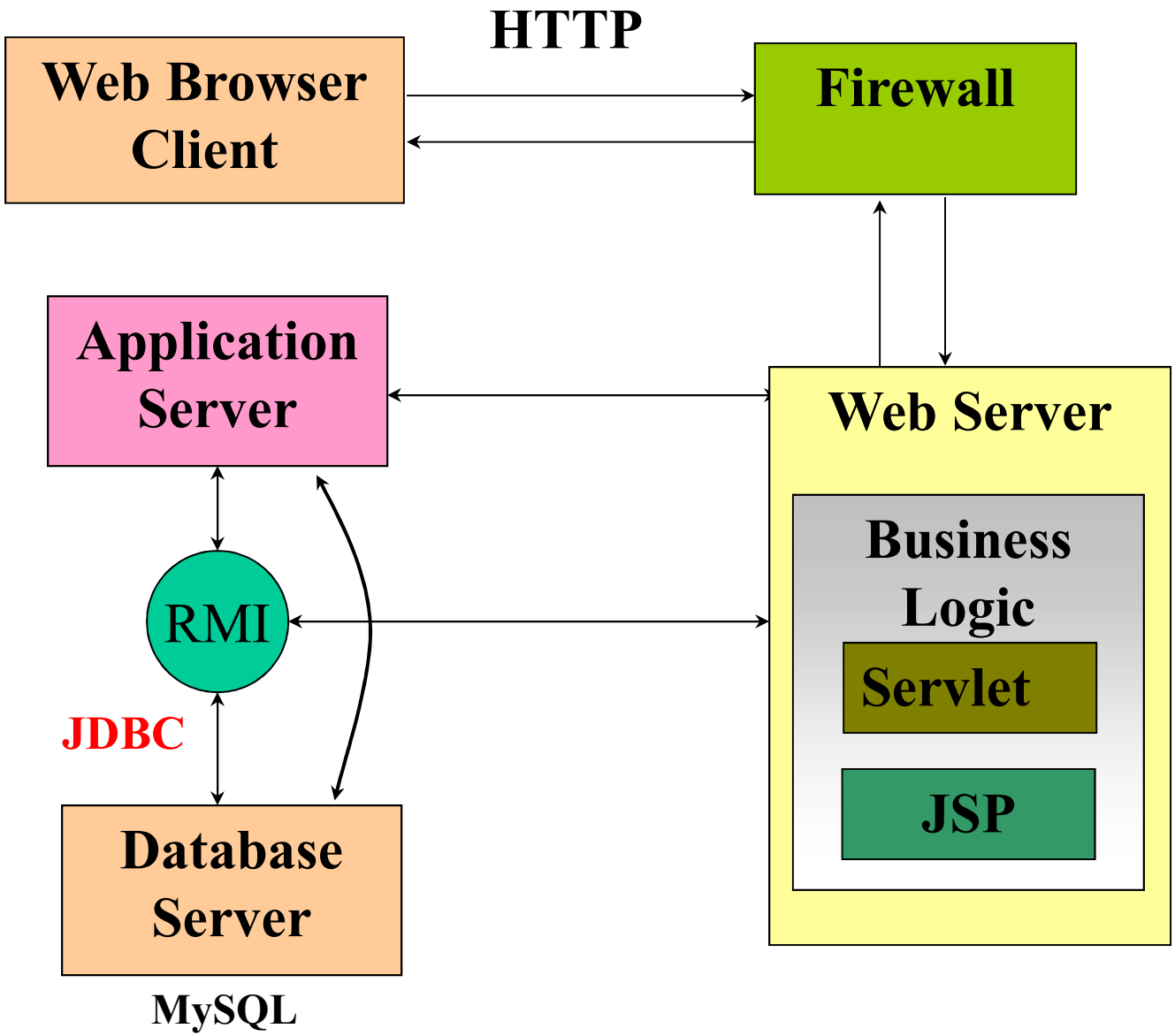
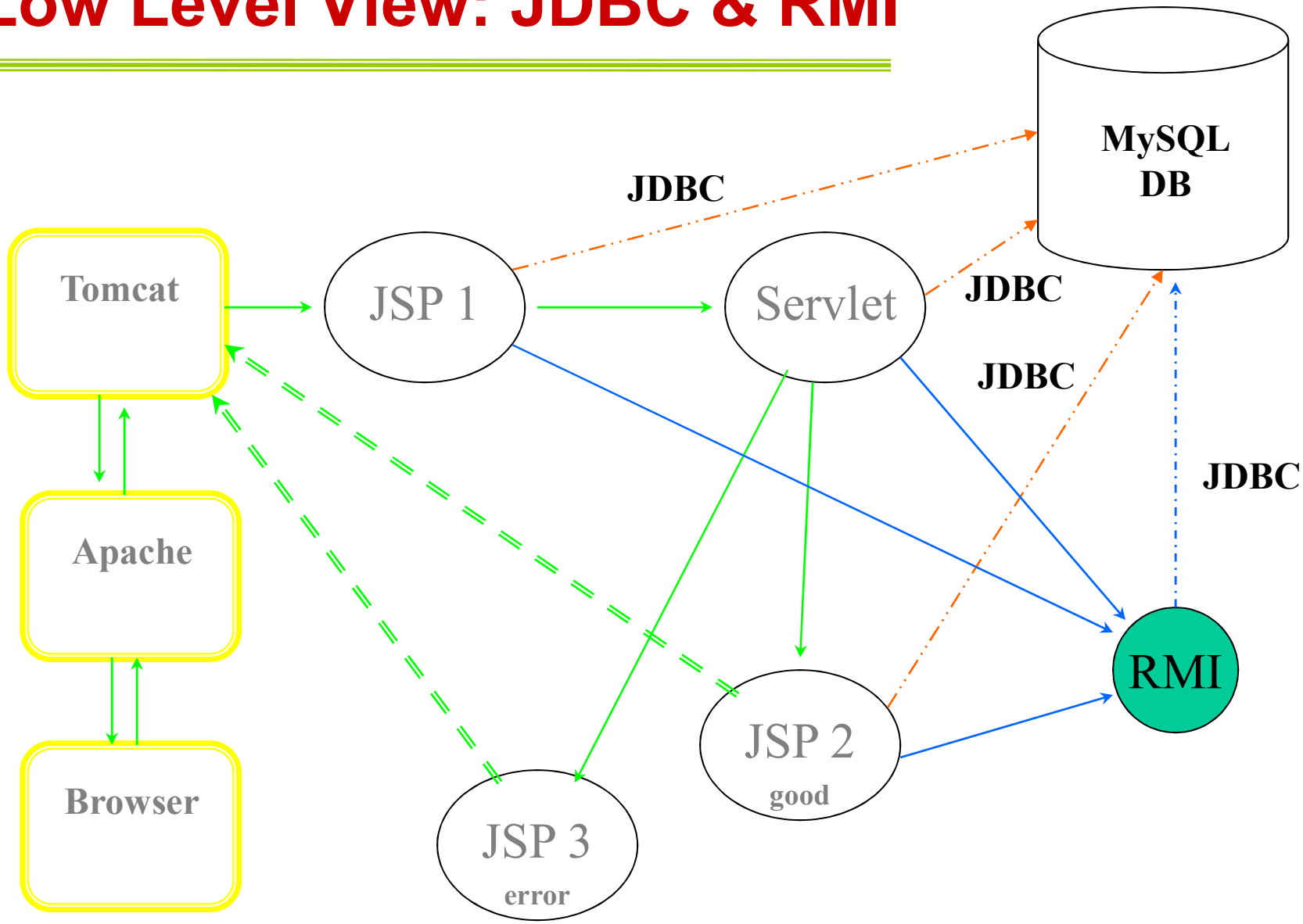
JDBC

Java DataBase Connectivity

JDBC & RMI


HTTP
Web Browser Client
Firewall
Application Server
Web Server
Business Logic
Servlet
JSP
RMI
JDBC
Database Server
MySQL

Low Level View: JDBC & RMI



JDBC

- The most popular style of database system is the relational database. A language called ***structured query language (SQL)*** is used among relational database systems to make queries.

- Java allows the programmers to use SQL to access information in the relational database systems.

Relational Database Model

- A relational database model is developed by Codd is a logical representation of the data that allows the relationships between the data to be concerned without concerning oneself with the physical implementation of the data structure.

Relational Database Model

- A relational database consists of tables.example:EMPLOYEE table

Number	Name	Department	Salary	Location
23603	JONES. A	413	1600	NEW JERSY
24568	KEWIN.R	413	2000	NEW JERSY
34589	LARSON P	642	1800	LOS ANGELES

Relational Database Model

- The name of the above table is EMPLOYEE and its primary purpose is to illustrate the various attributes of an employee and how they are related to a specific employee.

- Any particular row of a table is called record(or row). The above table has three records.

Relational Database Model

- The employee number field of each record in this table is used as the ***primary key.*** Tables in a database normally have primary keys, but primary keys are not required.

- The primary key can be composed of more than one column in the database. Primary key fields in a table cannot have duplicate values

Relational Database Model

- Each column of the table represents a different field.
- We can perform many kinds of operations using the tables. We can find particular fields of a table and combine two or more smaller tables into a one big table.
- The former operation is called ***projection*** and later is called ***join.***

How to use a MySQL database

Objectives

Applied

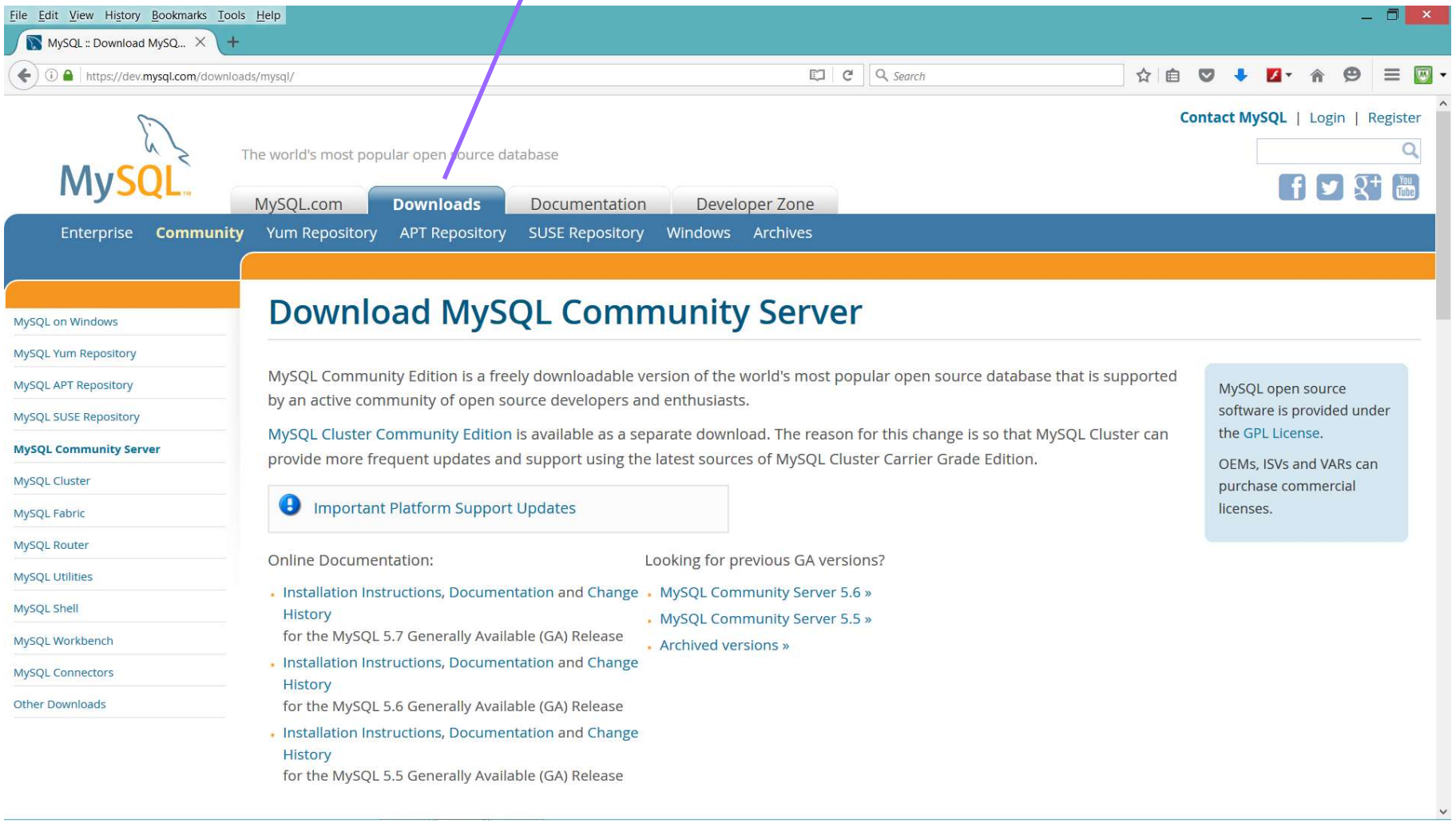
1. Use MySQL Workbench to start and stop the MySQL server.
2. Use MySQL Workbench to run SQL statements.
3. Use MySQL Workbench to run SQL scripts.
4. Code simple SELECT, INSERT, UPDATE, and DELETE statements and use MySQL Workbench to test them.

Knowledge

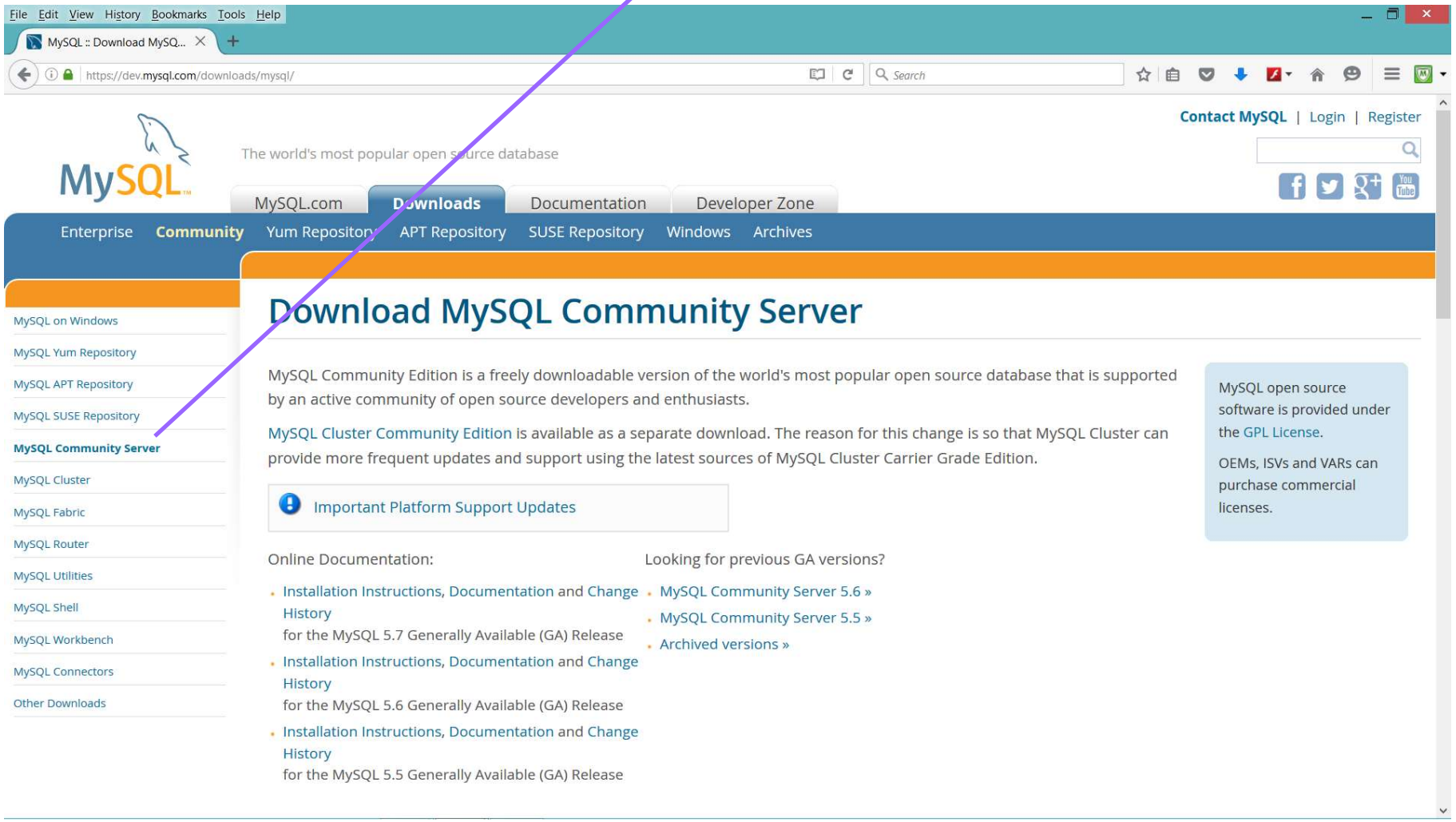
1. Distinguish between SQL's Data Definition Language (DDL) and Data Manipulation Language (DML).
2. Describe the capabilities of a SELECT statement.
3. Describe the capabilities of INSERT, UPDATE, and DELETE statements.
4. Describe what a SQL script does.

Go to MySQL download tab:

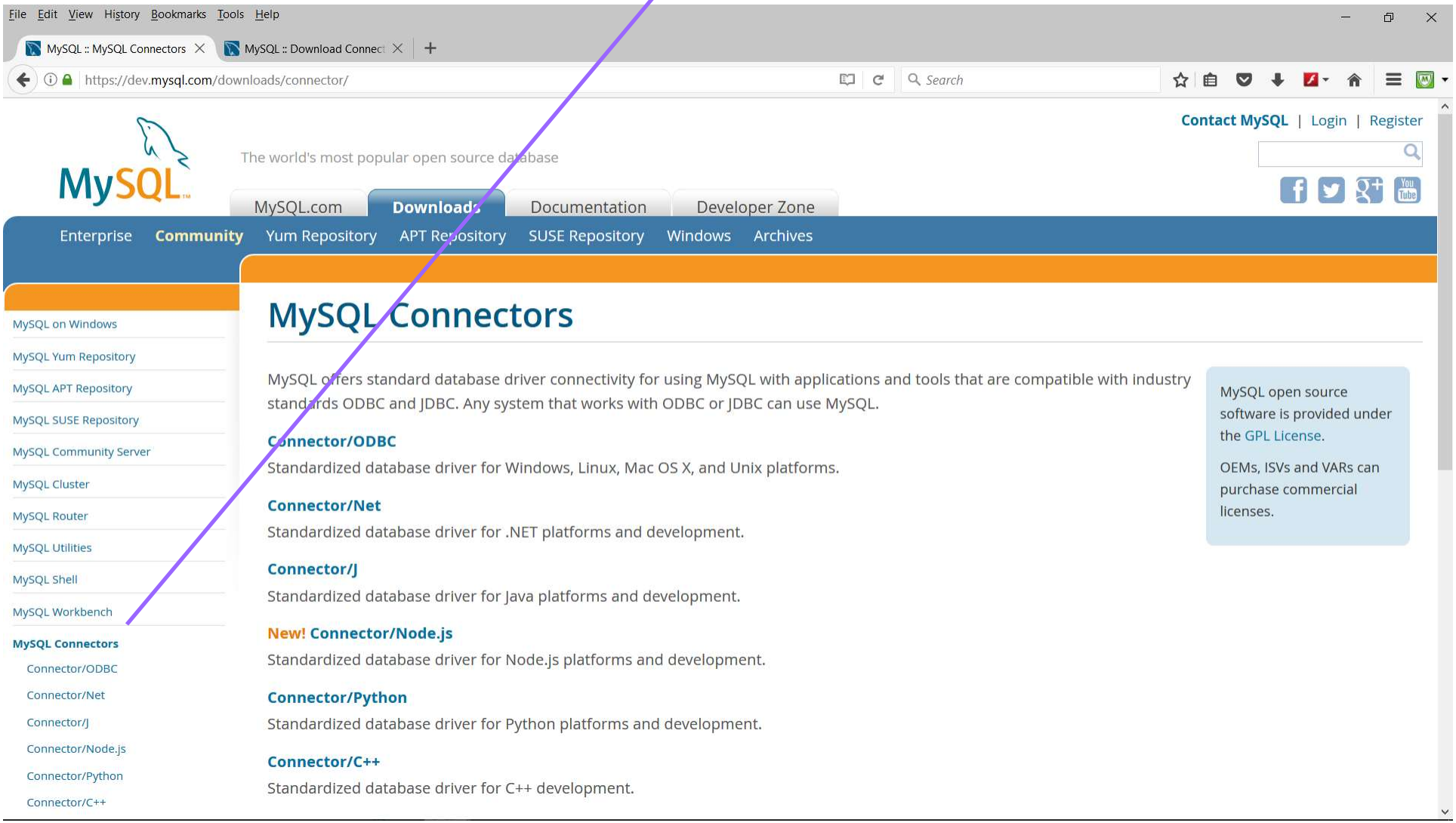
1. **Download the Community server**
2. **Download Connector/J for Java**



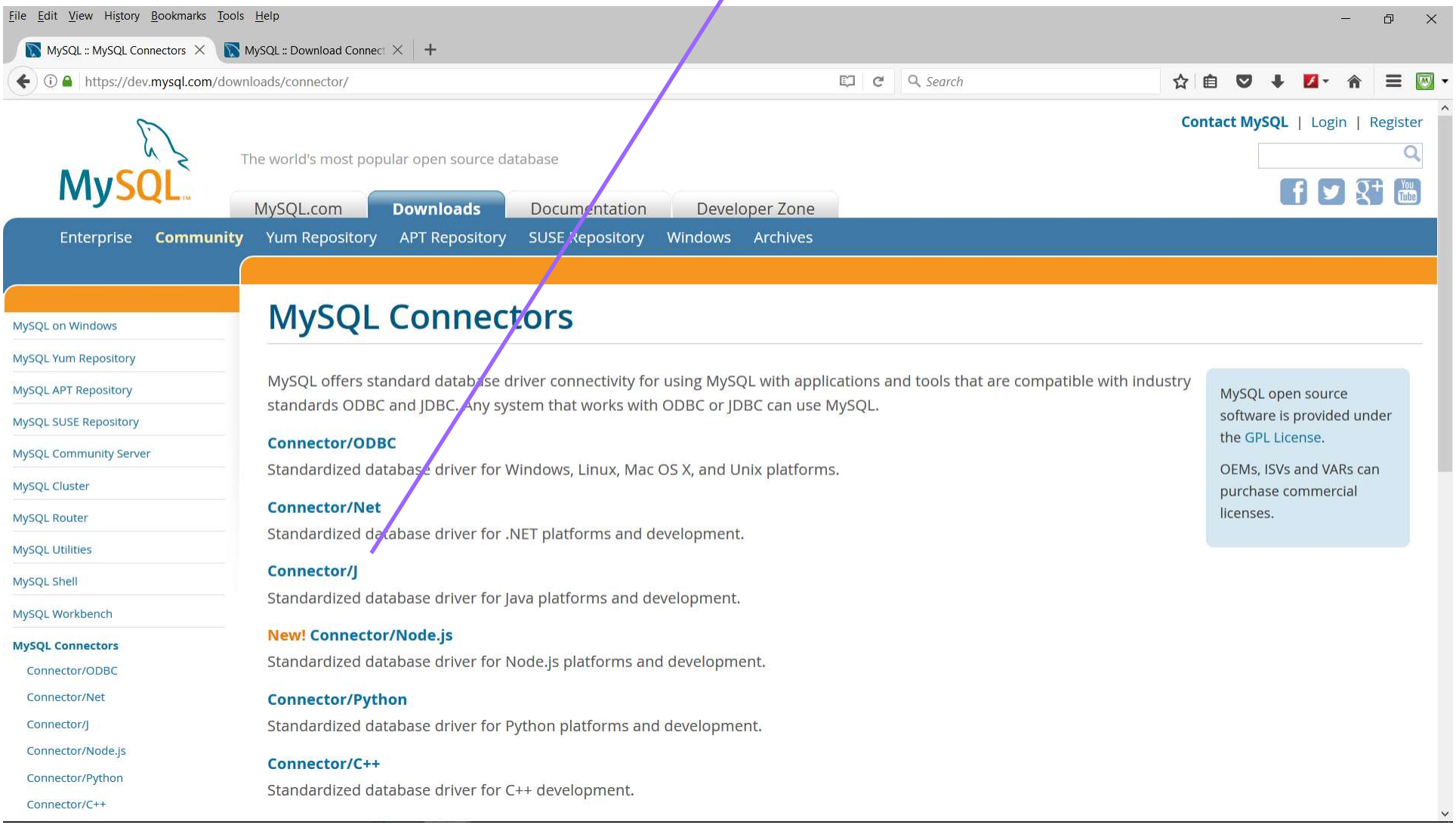
You need to download MySQL Community Server



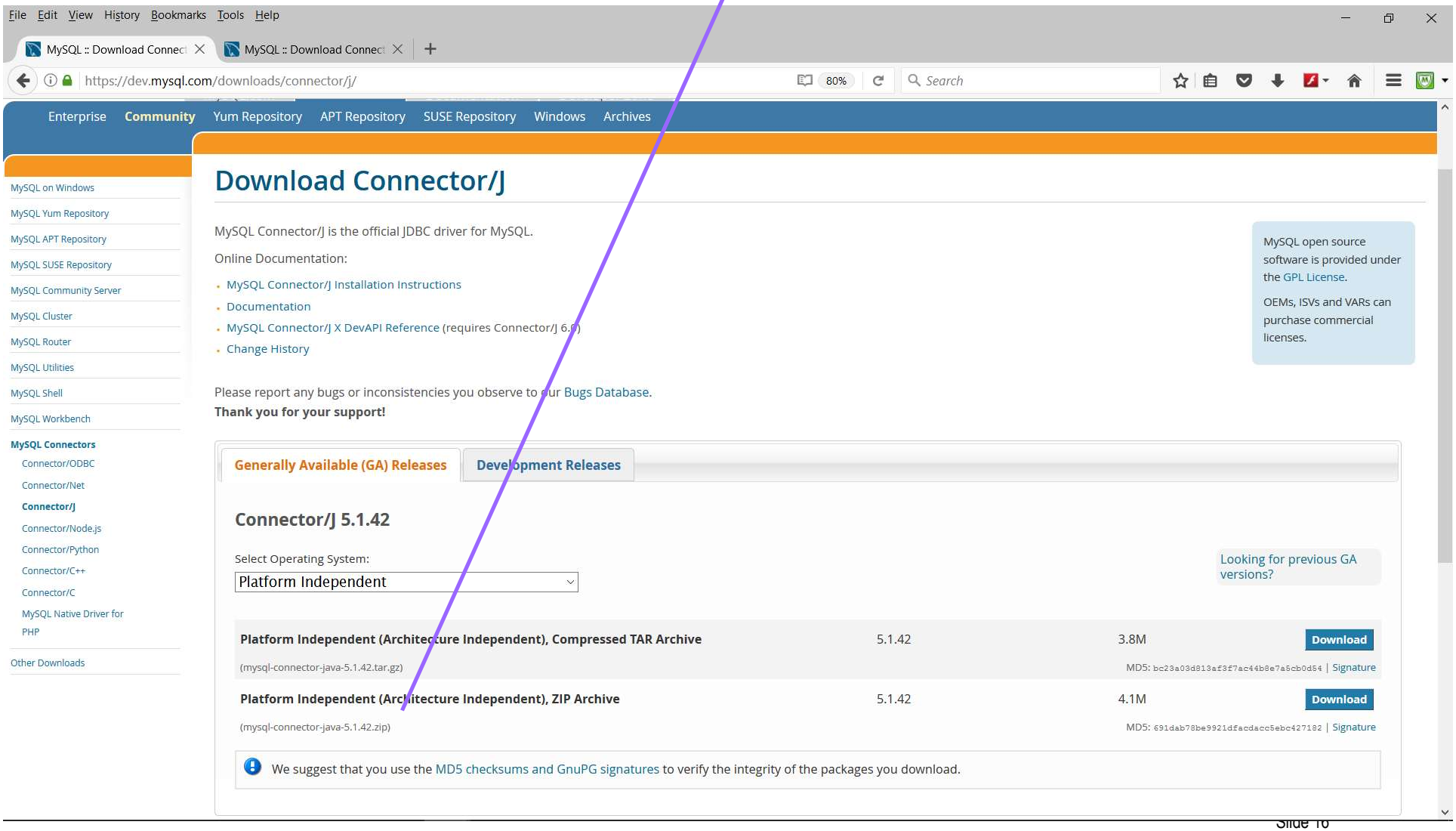
Click MySQL Connectors



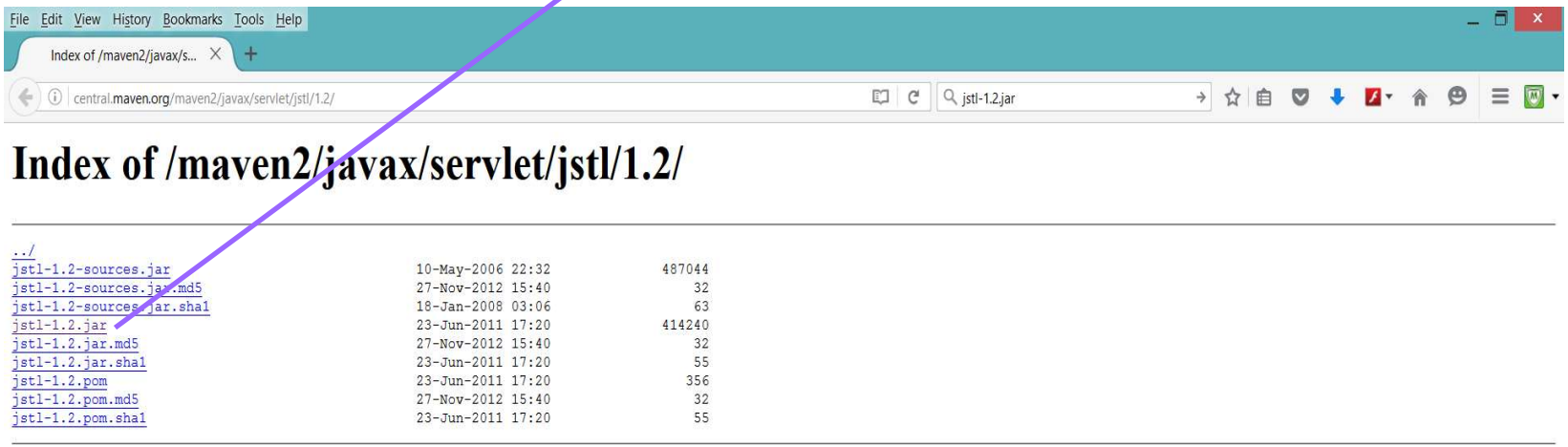
Select Connector/J



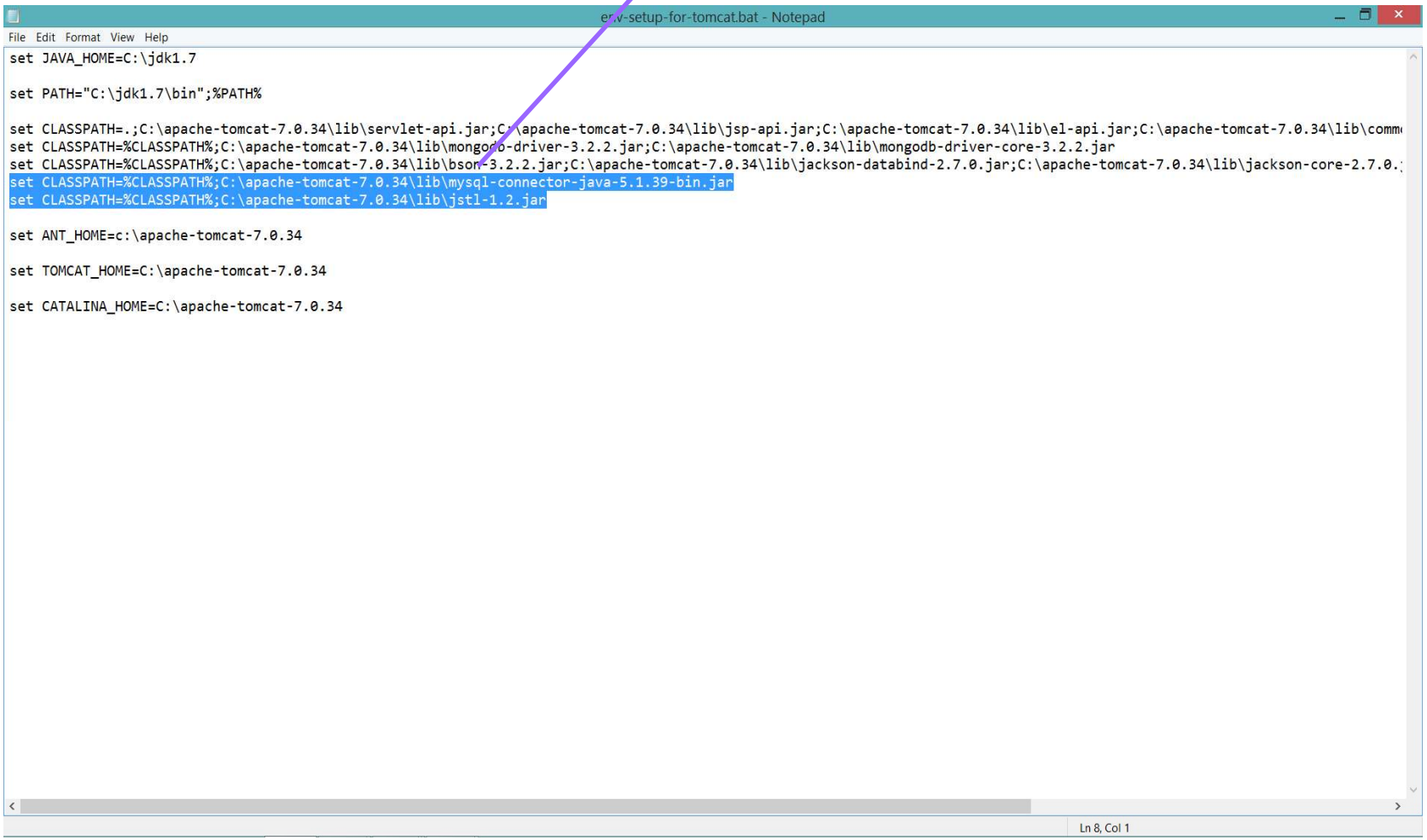
You need to download this and save in tomcat/lib directory



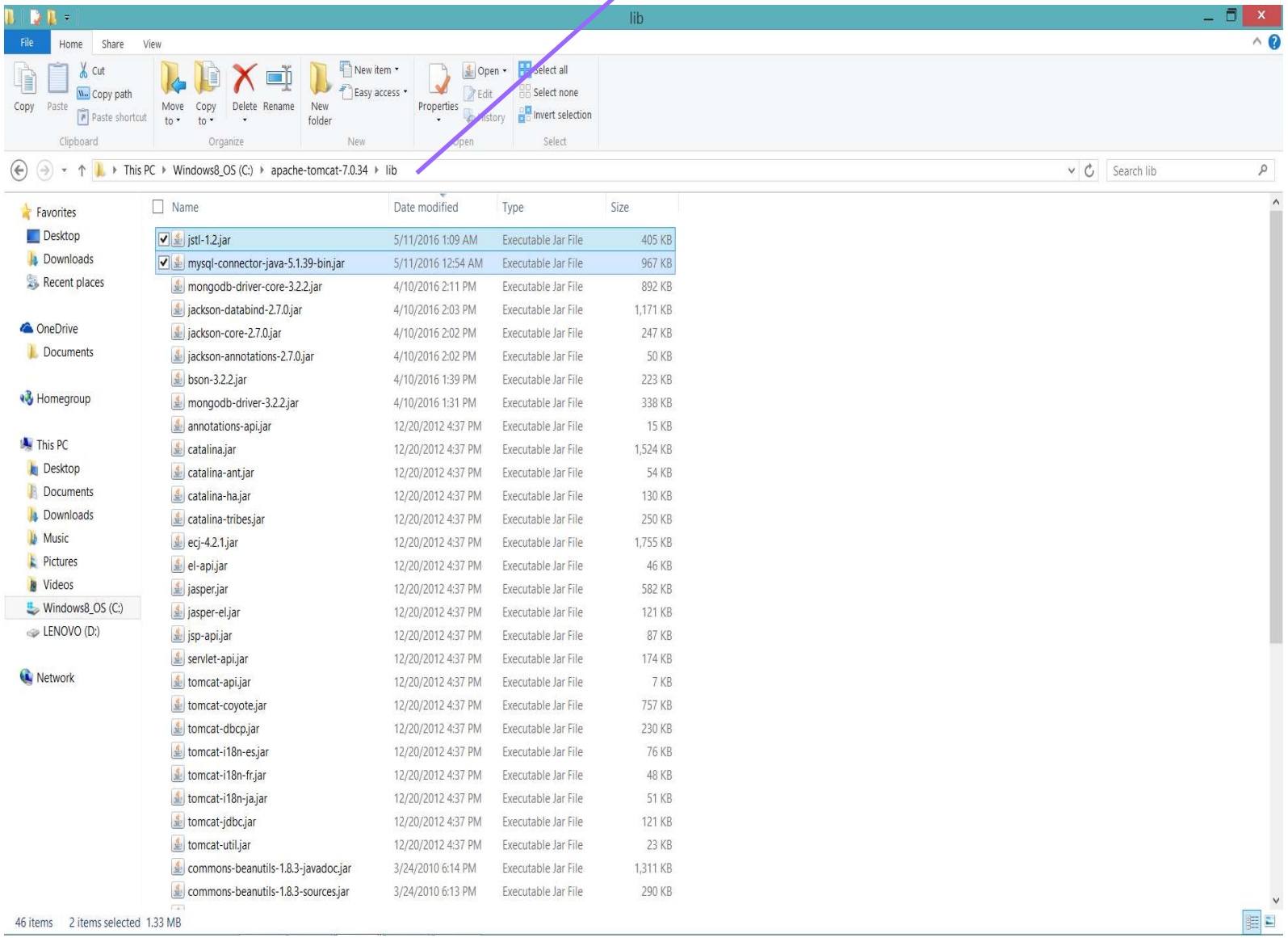
You need to download this and save in tomcat/lib directory



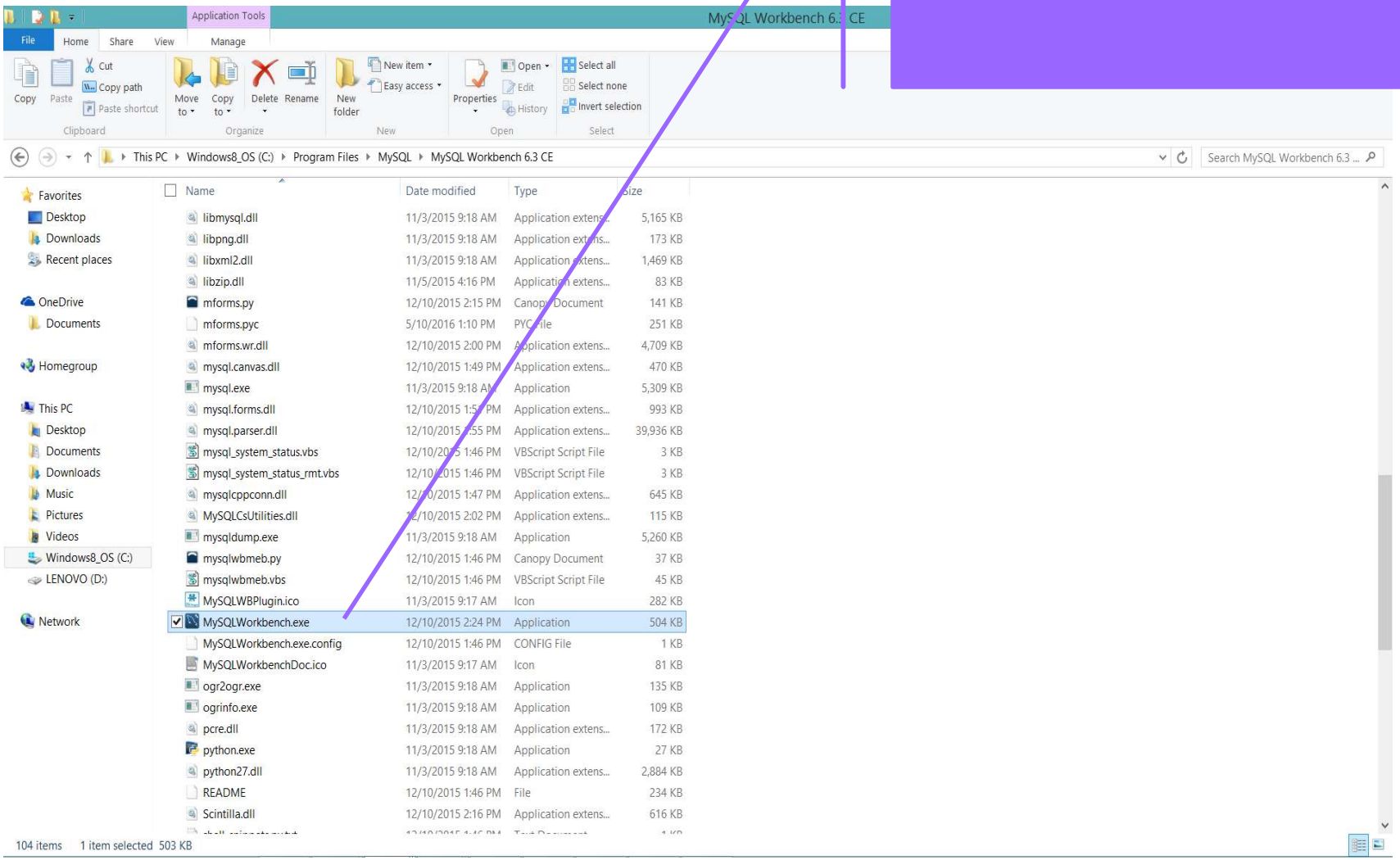
Add these two lines to env-setup-for-tomcat batch file



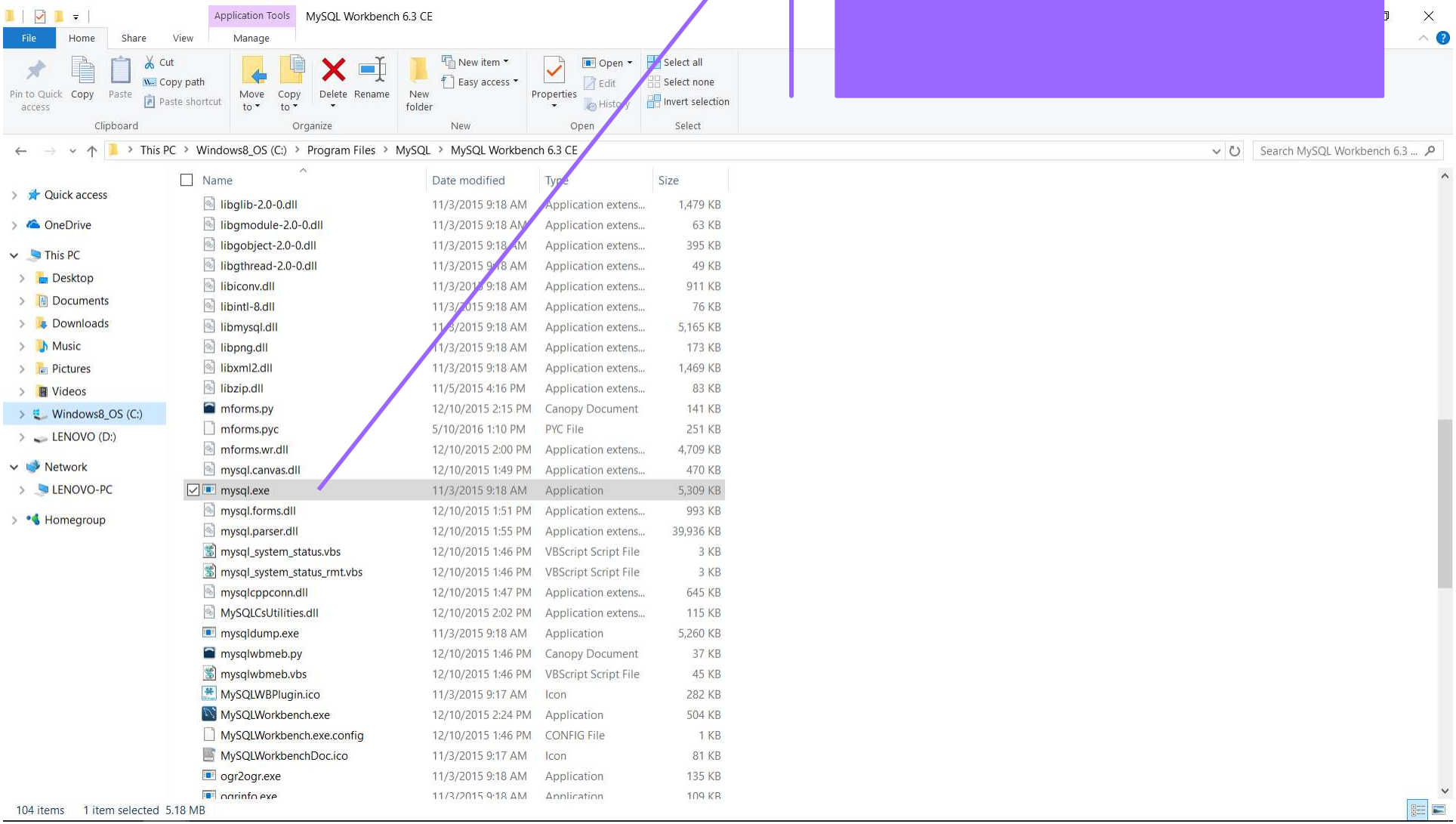
The jar files in lib



MySQL workbench



MySQL shell


Application Tools
MySQL Workbench 6.3 CE
File
Home
Share
View
Manage
Cut
Copy path
Paste shortcut
Pin to Quick access
Copy
Paste
Move to
Copy to
Delete
Rename
New folder
New item
Easy access
Properties
Open
Edit
History
Select all
Select none
Invert selection
Clipboard
Organize
New
Open
Select
This PC
Windows8_OS (C:)
Program Files
MySQL
MySQL Workbench 6.3 CE
Search MySQL Workbench 6.3 ...
Quick access
OneDrive
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Windows8_OS (C:)
LENOVO (D:)
Network
LENOVO-PC
Homegroup
Name
Date modified
Type
Size
libglib-2.0-0.dll 11/3/2015 9:18 AM Application extens... 1,479 KB
libgmodule-2.0-0.dll 11/3/2015 9:18 AM Application extens... 63 KB
libgobject-2.0-0.dll 11/3/2015 9:18 AM Application extens... 395 KB
libgthread-2.0-0.dll 11/3/2015 9:18 AM Application extens... 49 KB
libiconv.dll 11/3/2015 9:18 AM Application extens... 911 KB
libintl-8.dll 11/3/2015 9:18 AM Application extens... 76 KB
libmysql.dll 11/3/2015 9:18 AM Application extens... 5,165 KB
libpng.dll 11/3/2015 9:18 AM Application extens... 173 KB
libxml2.dll 11/3/2015 9:18 AM Application extens... 1,469 KB
libzip.dll 11/5/2015 4:16 PM Application extens... 83 KB
mforms.py 12/10/2015 2:15 PM Canopy Document 141 KB
mforms.pyc 5/10/2016 1:10 PM PYC File 251 KB
mforms.wr.dll 12/10/2015 2:00 PM Application extens... 4,709 KB
mysql.canvas.dll 12/10/2015 1:49 PM Application extens... 470 KB
mysql.exe 11/3/2015 9:18 AM Application 5,309 KB
mysql.forms.dll 12/10/2015 1:51 PM Application extens... 993 KB
mysql.parser.dll 12/10/2015 1:55 PM Application extens... 39,936 KB
mysql_system_status.vbs 12/10/2015 1:46 PM VBScript Script File 3 KB
mysql_system_status_rmt.vbs 12/10/2015 1:46 PM VBScript Script File 3 KB
mysqlcppconn.dll 12/10/2015 1:47 PM Application extens... 645 KB
MySQLCsUtilities.dll 12/10/2015 2:02 PM Application extens... 115 KB
mysqldump.exe 11/3/2015 9:18 AM Application 5,260 KB
mysqlwbmeb.py 12/10/2015 1:46 PM Canopy Document 37 KB
mysqlwbmeb.vbs 12/10/2015 1:46 PM VBScript Script File 45 KB
MySQLWBPlugin.ico 11/3/2015 9:17 AM Icon 282 KB
MySQLWorkbench.exe 12/10/2015 2:24 PM Application 504 KB
MySQLWorkbench.exe.config 12/10/2015 1:46 PM CONFIG File 1 KB
MySQLWorkbenchDoc.ico 11/3/2015 9:17 AM Icon 81 KB
ogr2ogr.exe 11/3/2015 9:18 AM Application 135 KB
104 items
1 item selected 5.18 MB

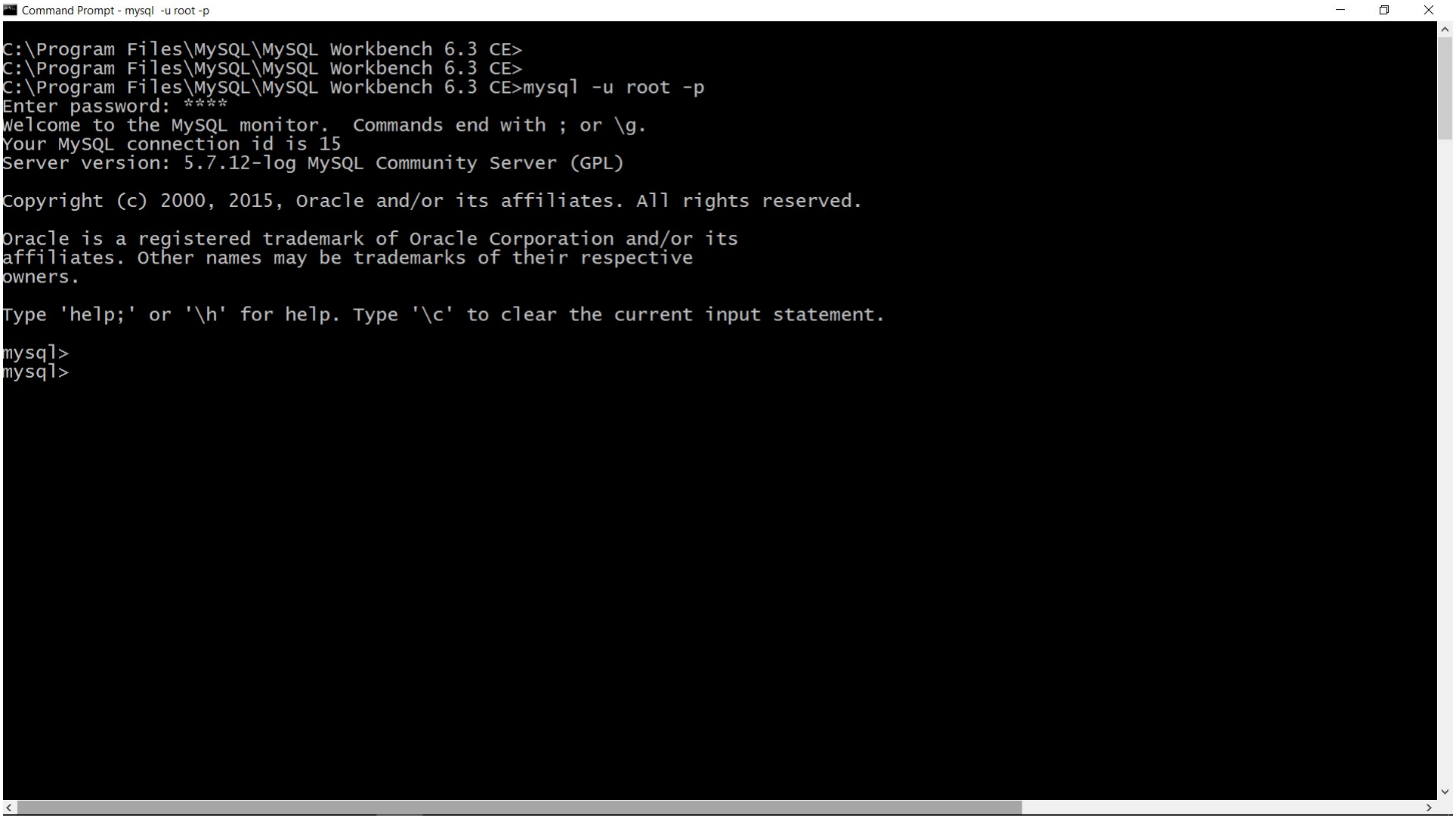
MySQL is…

- **Inexpensive.** Free for most uses and relatively inexpensive for other uses.
- **Fast.** One of the fastest relational databases currently available.
- **Easy to use.** Easy to install and use.
- **Portable.** Runs on most modern operating systems including Windows, OS X, and Linux.

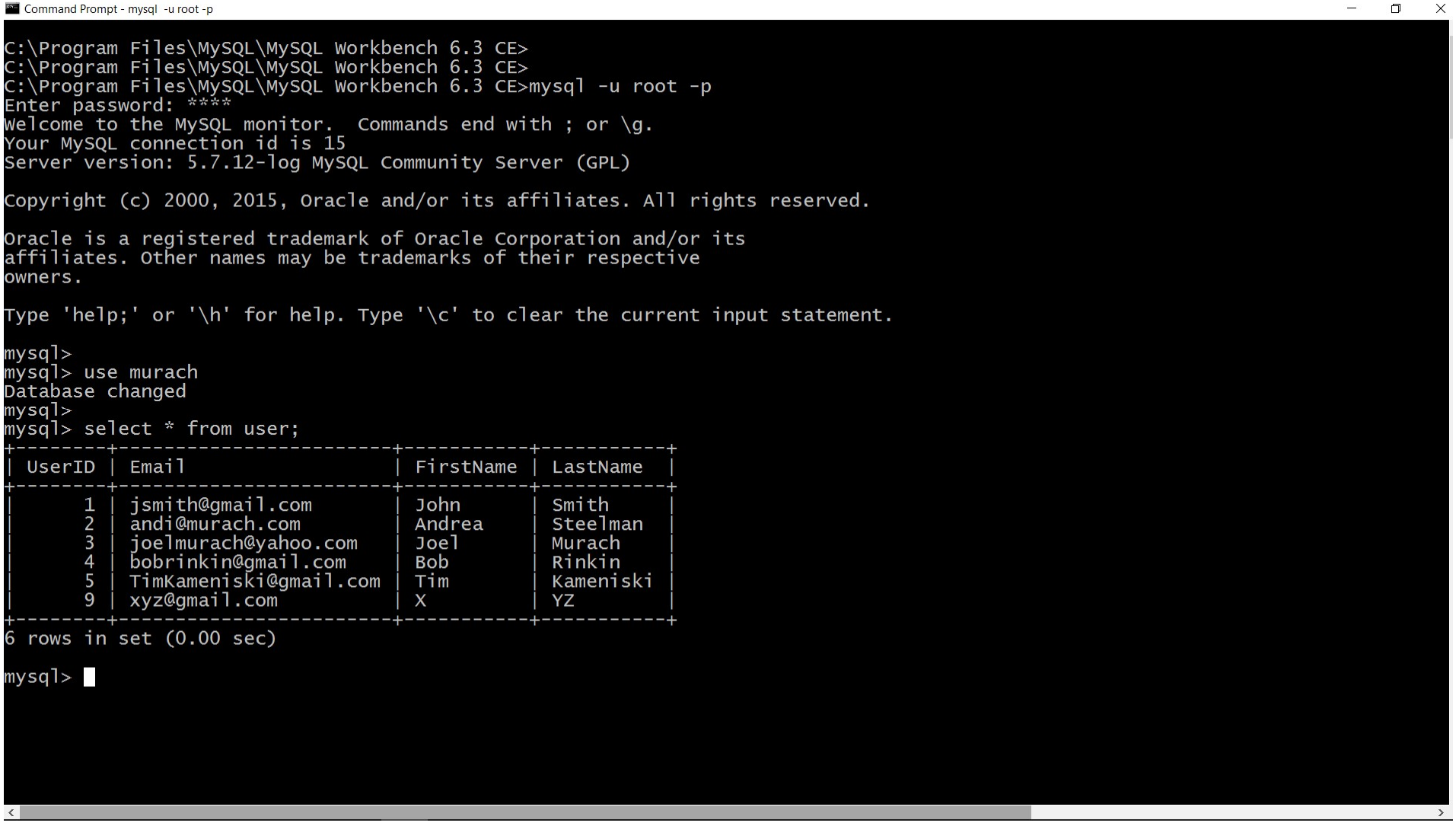
MySQL provides…

- **Support for SQL.** Like any modern database product, MySQL supports SQL.
- **Support for multiple clients.** Supports access from multiple clients from a variety of interfaces and programming languages including Java, PHP, Python, Perl, and C.
- **Connectivity.** Provides access to data via an intranet or the Internet.
- **Security.** Protects access to your data so only authorized users can view the data.
- **Referential integrity.** With MySQL 5.5 and later, InnoDB tables are used by default, which support referential integrity.
- **Transaction processing.** With version 5.5, MySQL uses InnoDB tables by default, which provide support for transaction processing.

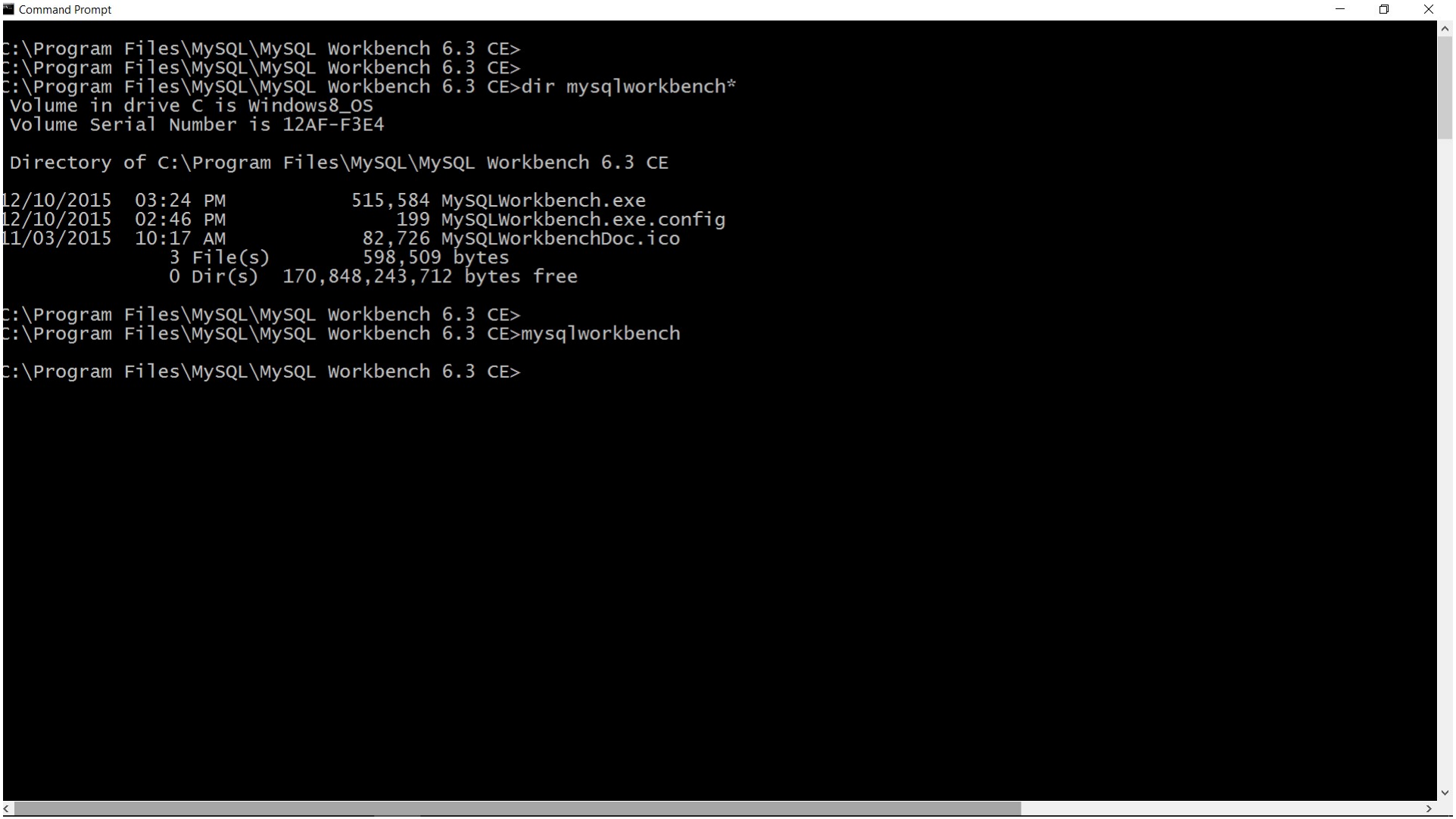
A command-line tool – I logged in as root



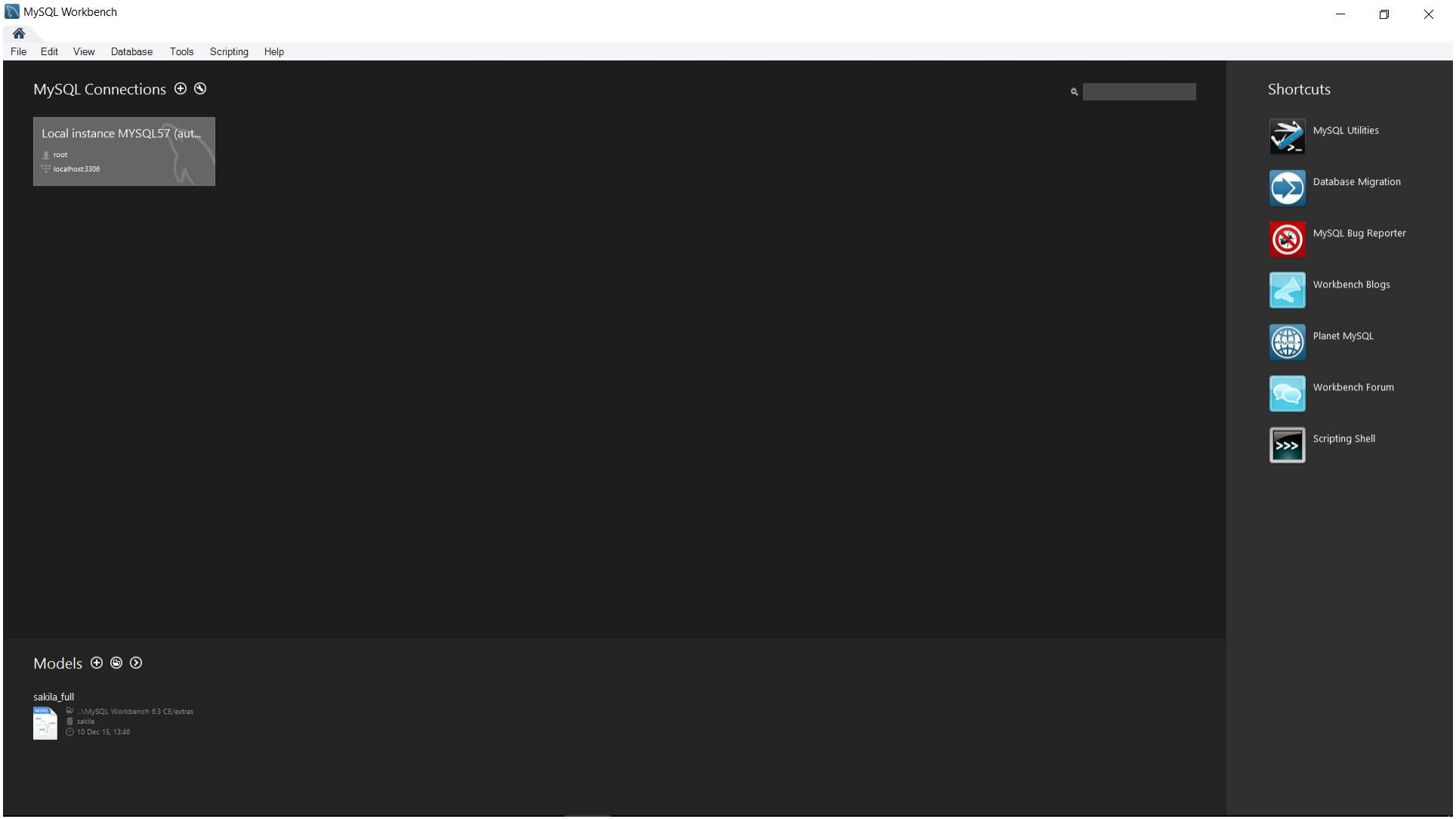
A command-line tool – Use murach



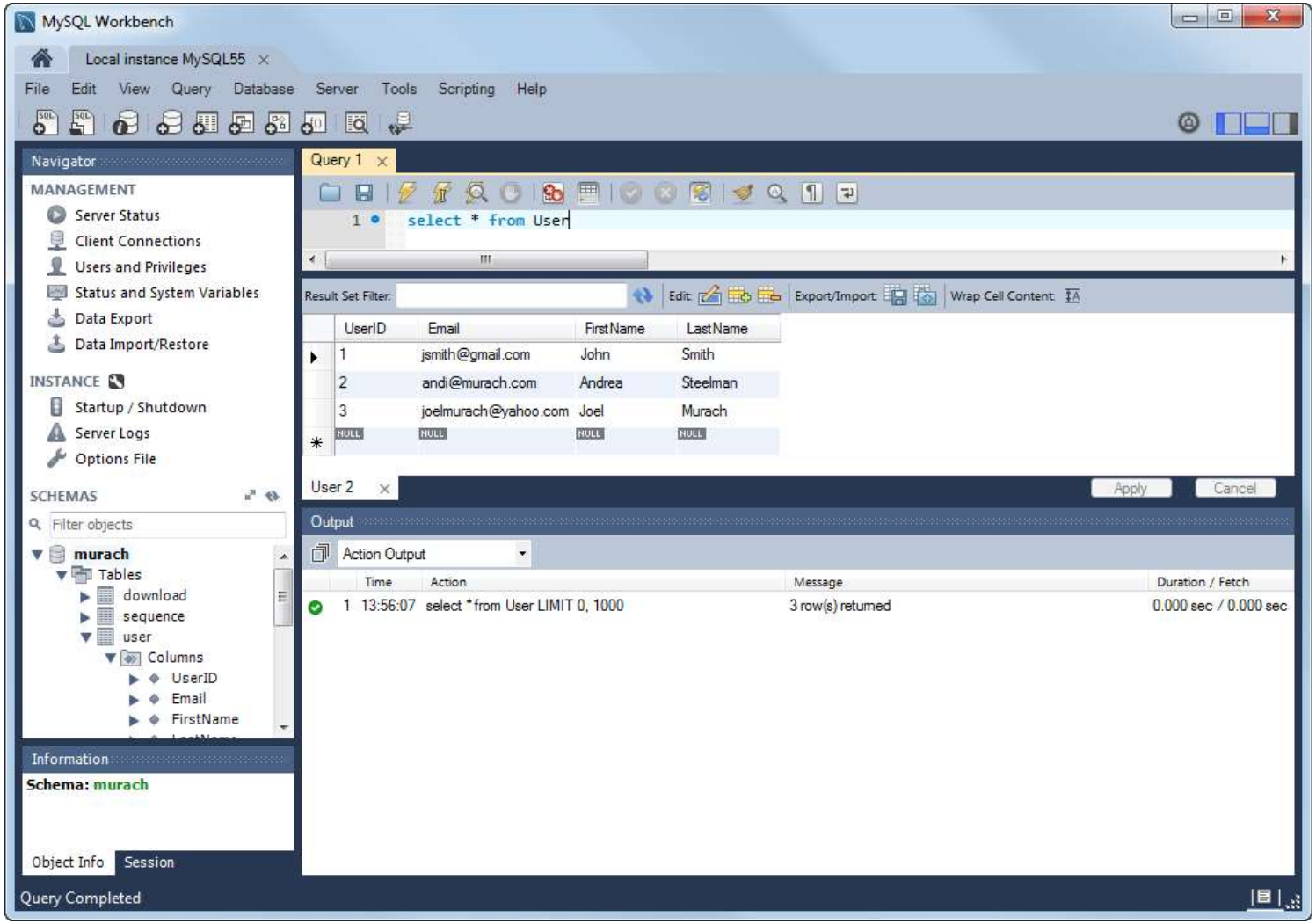
MySQL Workbench



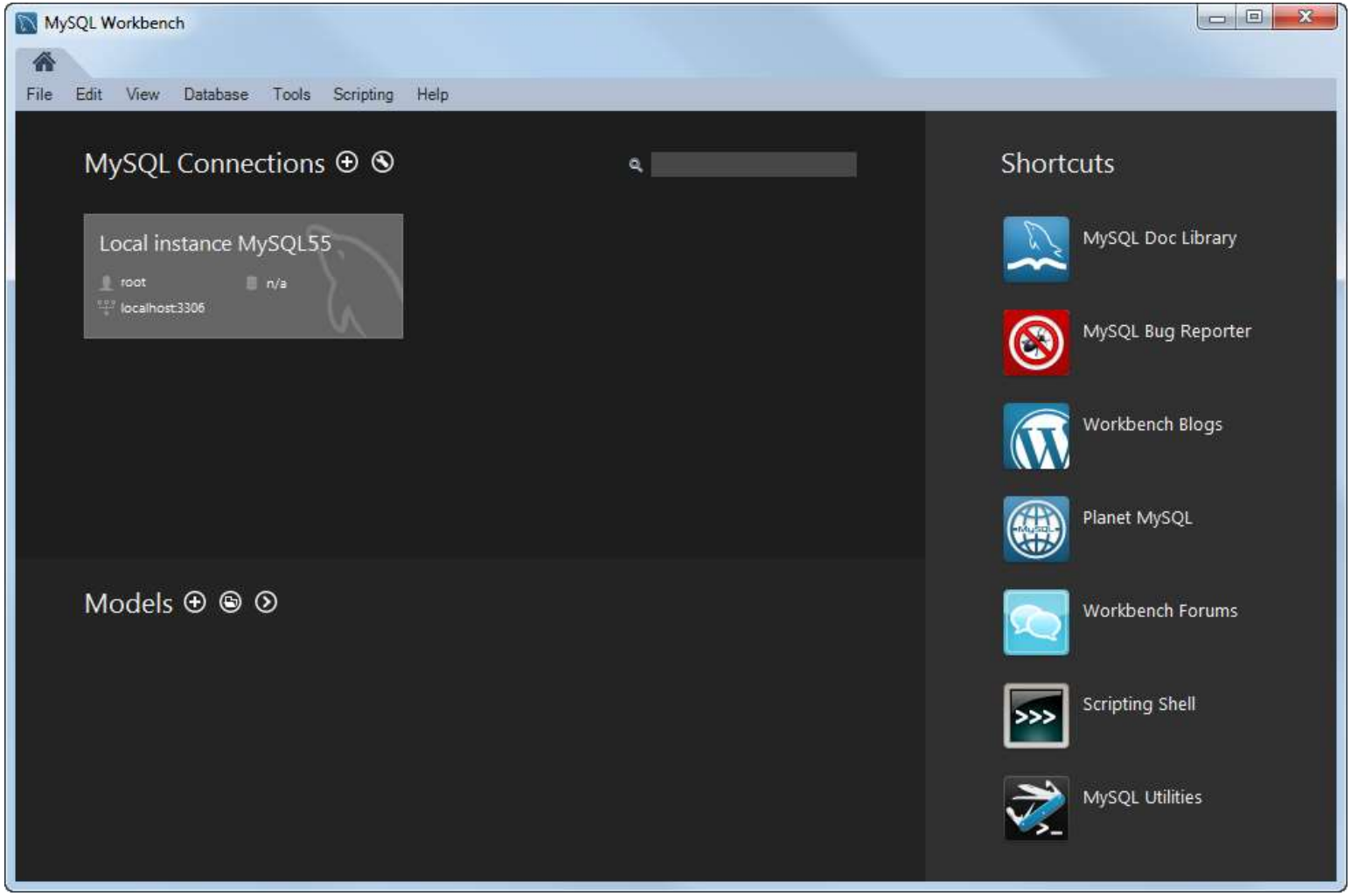
MySQL Workbench



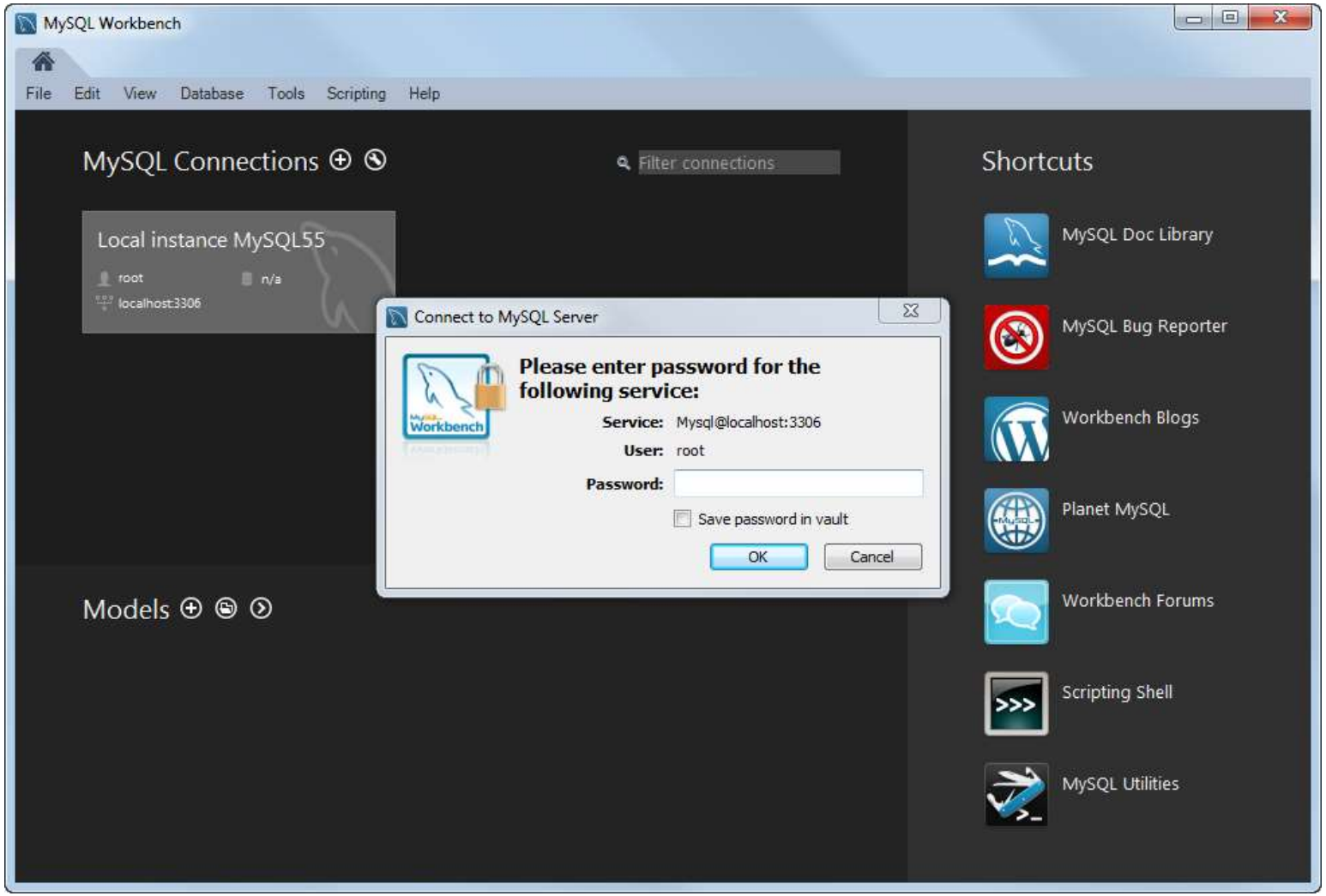
MySQL Workbench



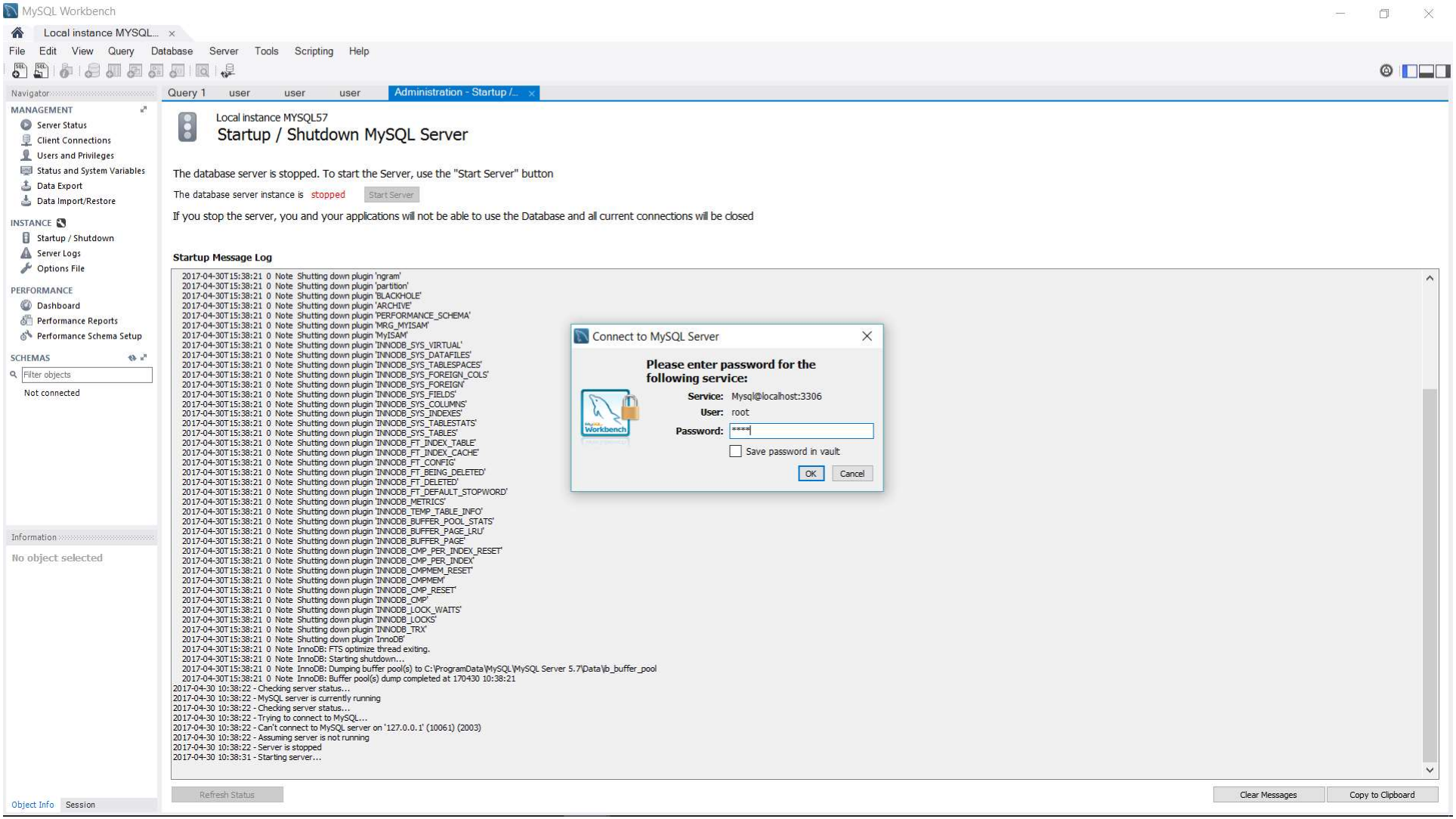
The Home tab of MySQL Workbench



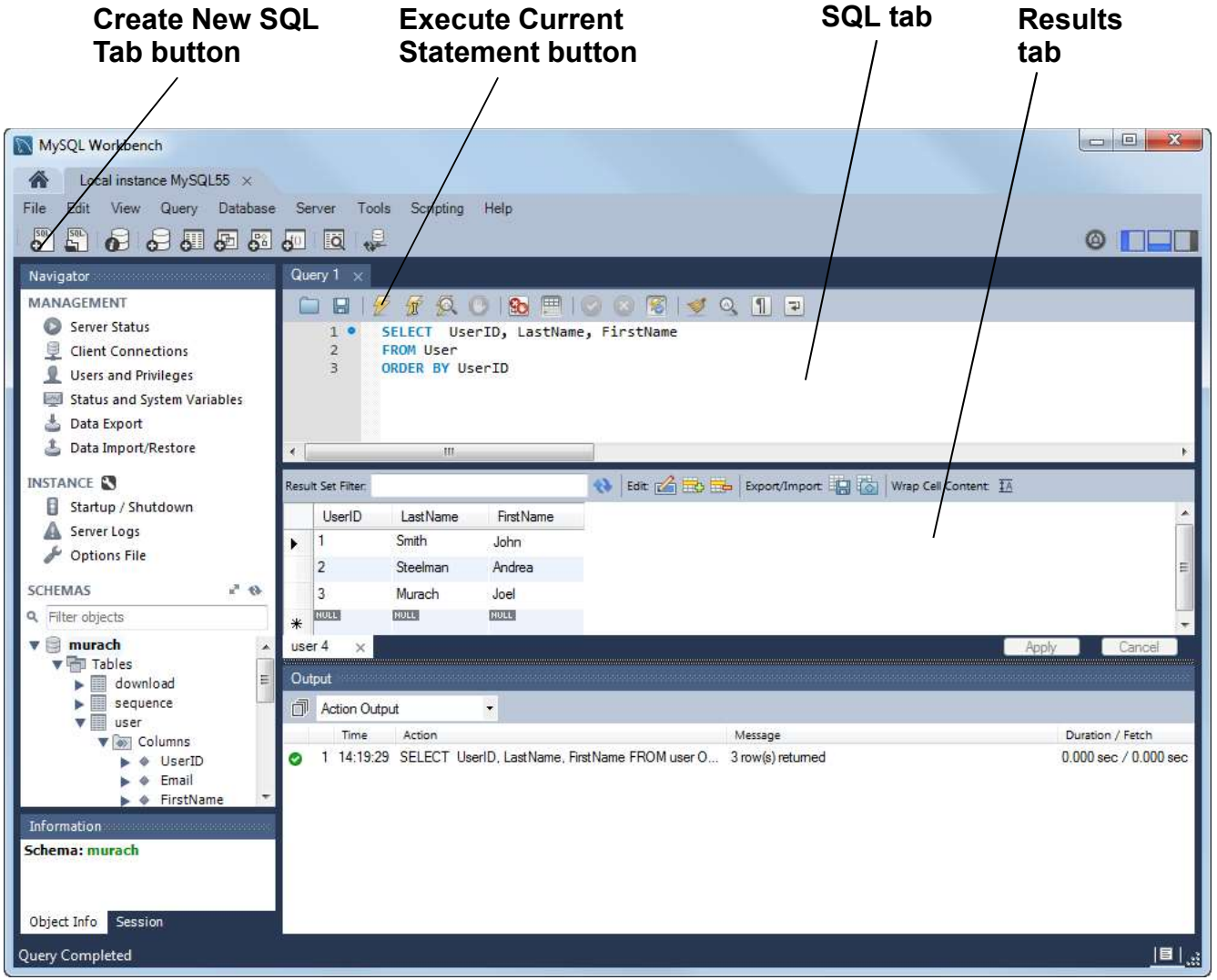
The dialog box for opening database connections



The Startup/Shutdown option



A SELECT statement and its results

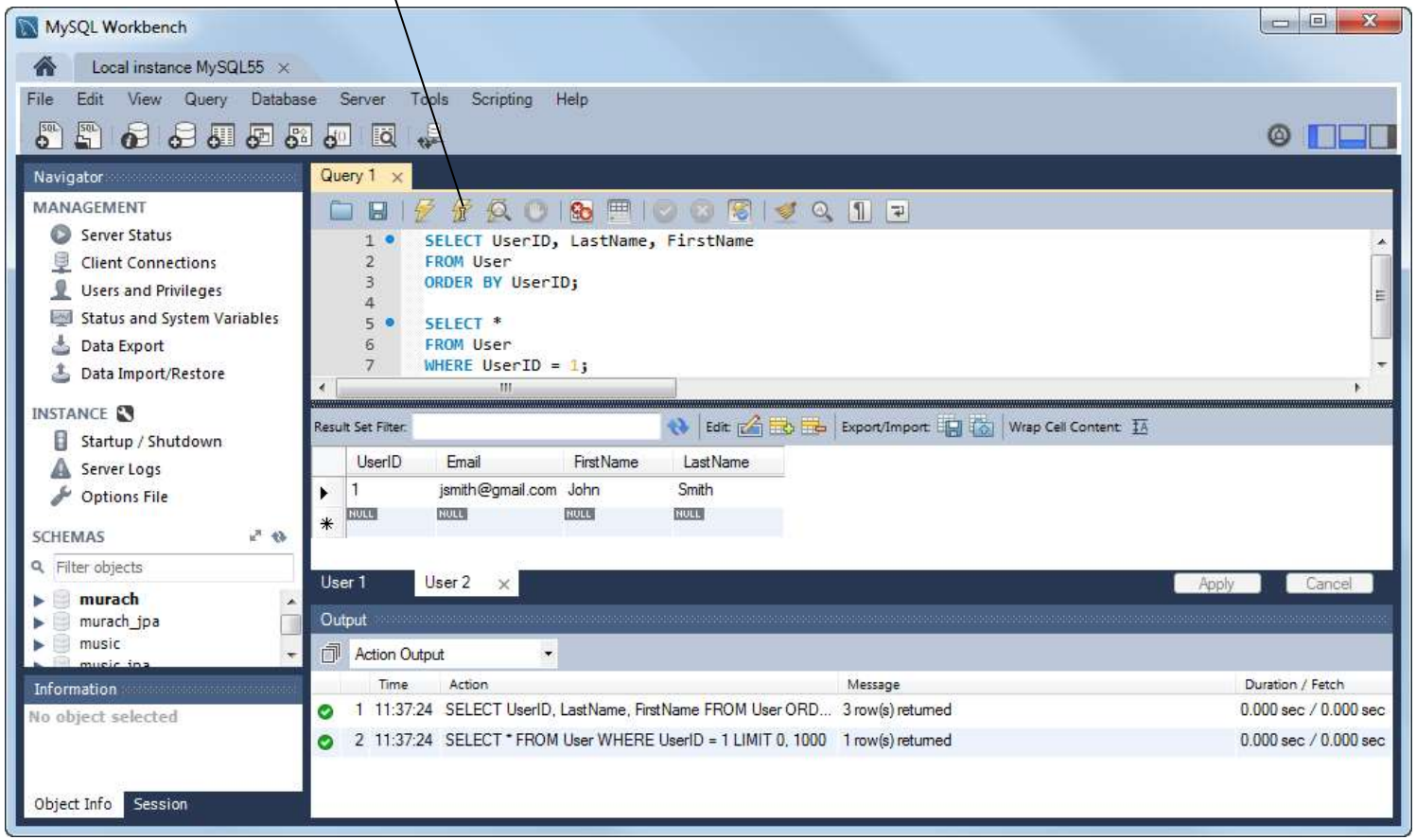


How to enter and execute a SQL statement

- To open a new SQL tab, press Ctrl+T or click the Create New SQL Tab button in the SQL editor toolbar.
- To select the current database, double-click it in the Schemas section of the Navigator window. This displays the selected database in bold.
- To enter a SQL statement, type it into the SQL tab.
- As you enter the text for a statement, the SQL tab applies color to various elements, such as SQL keywords, to make them easy to identify.
- To execute a SQL statement, press Ctrl+Enter, or click the Execute Current Statement button in the SQL editor toolbar. If the statement retrieves data, the data is displayed in a Results tab below the SQL tab.

A SQL script and its results

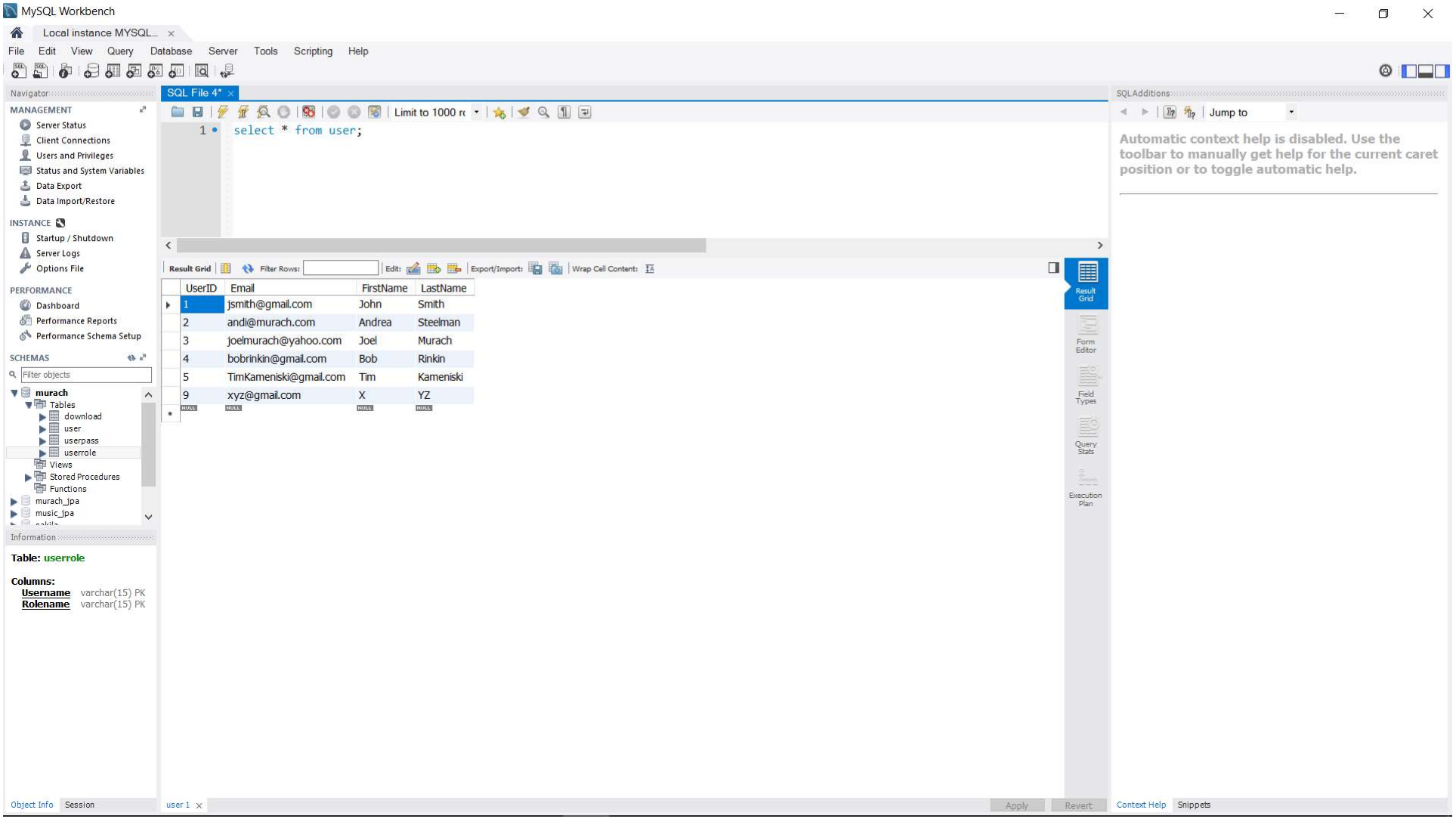
Execute SQL Script button



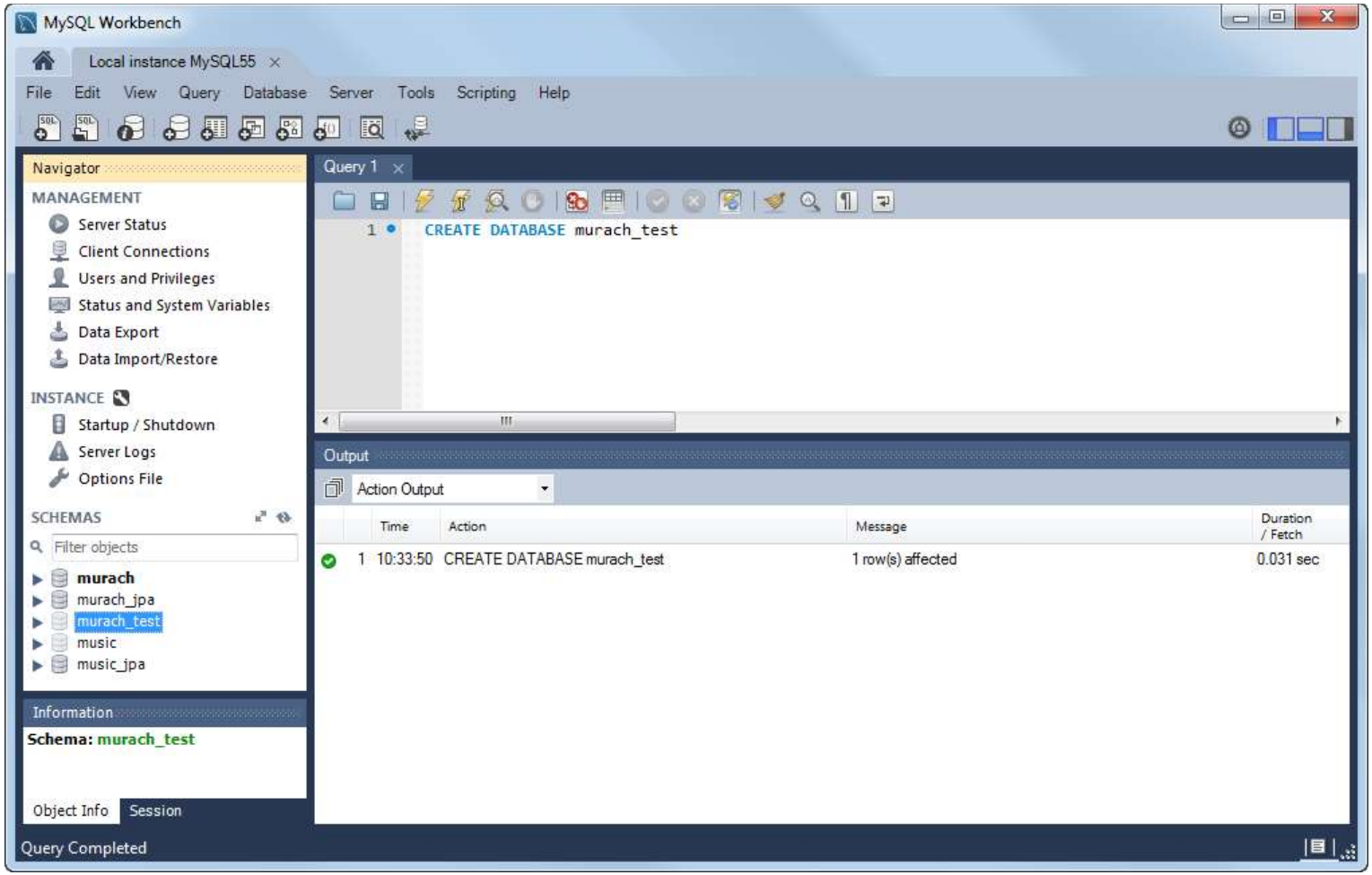
How to enter and execute a SQL script

- When you code a script that contains more than one statement, code a semicolon at the end of each statement.
- To run an entire SQL script, press the Ctrl+Shift+Enter keys or click the Execute SQL Script button that's located just to the left of the Execute Current Statement button in the SQL editor toolbar.
- When you run a SQL script, the results of each statement that returns data are displayed in a separate Results tab.
- To execute one SQL statement within a script, move the insertion point into that statement and press Ctrl+Enter or click the Execute Current Statement button. If the statement retrieves data, the data is displayed in a Results tab.
- To execute two or more statements within a script, select them in the editor and then press Ctrl+Shift+Enter or click the Execute SQL Script button.

After a statement has been executed



After a statement has been executed



How to create a database

```
CREATE DATABASE murach_test
```

How to select a database for use

```
USE murach_test
```

How to drop a database

```
DROP DATABASE murach_test
```

How to create, select, and drop a database

- Use the CREATE DATABASE statement to create a database and the DROP DATABASE statement to delete a database. These are *SQL statements*.
- Use the USE command to select the database that you want to work with. This is a *MySQL command*.
- You can also select a database by double-clicking on it in the Schemas section in MySQL Workbench.
- The selected database will appear in bold in the Schemas section.

How to create a table

```
CREATE TABLE User (
    UserID INT NOT NULL AUTO_INCREMENT,
    Email VARCHAR(50),
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    PRIMARY KEY(UserID)
)
```

How to drop a table

```
DROP TABLE User
```

How to drop a table only if it exists

```
DROP TABLE IF EXISTS User
```

Tables, rows, columns, etc.

- A *relational database* consists of one or more *tables* that consist of *rows* (*records*) and *columns* (*fields*).
- The *primary key* in a table is the one that uniquely identifies each of the rows in the table.
- A *foreign key* is used to relate the rows in one table to the rows in another table.
- When you create a table, you define each of its columns and you identify its primary key.
- To define a column, you must supply the name and the data type, whether it's automatically generated for new rows, and so on.
- On Unix systems, the table and column names are case-sensitive.

An INSERT statement that inserts multiple rows

```
INSERT INTO User
  (FirstName, LastName, Email)
VALUES
  ('John', 'Smith', 'jsmith@gmail.com'),
  ('Andrea', 'Steelman', 'andi@murach.com'),
  ('Joel', 'Murach', 'joelmurach@yahoo.com')
```

How to insert multiple rows into a table

- The INSERT statement lets you insert one or more rows into one table of a database. When you code it, you need to include data for all columns that aren't defined with default values or aren't automatically generated.
- On a Unix system, table and column names are case-sensitive.

A SELECT statement that gets all columns

Syntax

```
SELECT *
FROM table-1
[WHERE selection-criteria]
[ORDER BY column-1 [ASC|DESC] [, column-2 [ASC|DESC] ...]]
```

A statement that selects all rows and columns

```
SELECT * FROM User
```

Result set

	UserID	Email	FirstName	LastName
▶	1	jsmith@gmail.com	John	Smith
	2	andi@murach.com	Andrea	Steelman
	3	joelmurach@yahoo.com	Joel	Murach
*	NULL	NULL	NULL	NULL

A SELECT statement that gets selected columns

Syntax

```
SELECT column-1 [,column-2] ...
FROM table-1
[WHERE selection-criteria]
[ORDER BY column-1 [ASC|DESC] [,column-2 [ASC|DESC] ...]]
```

A statement that selects two rows and two columns

```
SELECT FirstName, LastName
FROM User
WHERE UserID < 3
ORDER BY LastName ASC
```

Result set

	FirstName	LastName
▶	John	Smith
	Andrea	Steelman

How to select data from a single table

- A SELECT statement is a SQL DML statement that returns a *result set* (or *result table*) that consists of the specified rows and columns.
- To specify the columns, use the SELECT clause.
- To specify the rows, use the WHERE clause.
- To specify the table that the data should be retrieved from, use the FROM clause.
- To specify how the result set should be sorted, use the ORDER BY clause.

A SELECT statement that joins two tables

Syntax

```
SELECT column-1 [,column-2] ...
FROM table-1
   {INNER | LEFT OUTER | RIGHT OUTER} JOIN table-2
   ON table-1. column-1 {=|<|>|<=|>=|<>} table-2.column-2
[WHERE selection-criteria]
[ORDER BY column-1 [ASC|DESC] [,column-2 [ASC|DESC] ...]]
```

A SELECT statement that joins two tables (cont.)

A statement that joins the User and Download tables

```
SELECT Email, DownloadFilename, DownloadDate
FROM User
   INNER JOIN Download
   ON User.UserID = Download.UserID
WHERE DownloadDate > '2014-01-01'
ORDER BY Email ASC
```

Result set

Email	DownloadFilename	DownloadDate
andi@murach.com	jr01_filter.mp3	2014-03-27 13:05:39
joelmurach@yahoo.com	jr01_so_long.mp3	2014-03-27 13:05:39
jsmith@gmail.com	jr01_so_long.mp3	2014-02-01 00:00:00
jsmith@gmail.com	jr01_filter.mp3	2014-03-27 13:05:39

How to select data from multiple tables

- To return a result set that contains data from two tables, join the tables. To do that, use a JOIN clause. Most of the time, you'll want to code an *inner join* so that rows are only included when the key of a row in the first table matches the key of a row in the second table.
- In a *left outer join*, the data for all of the rows in the first table (the one on the left) are included in the table, but only the data for matching rows in the second table are included. In a *right outer join*, the reverse is true.
- An inner join is the default type of join. As a result, it's common to omit the INNER keyword from a SELECT statement for an inner join.

The INSERT statement

Syntax

```
INSERT INTO table-name [(column-list)]
VALUES (value-list)
```

A statement that adds one row to the Download table

```
INSERT INTO Download
    (UserID, DownloadDate, DownloadFilename, ProductCode)
VALUES
    (1, '2014-05-01', 'jr01_so_long.mp3', 'jr01')
```

A statement that uses MySQL's NOW function to get the current date

```
INSERT INTO Download
    (UserID, DownloadDate, DownloadFilename, ProductCode)
VALUES
    (1, NOW(), 'jr01_filter.mp3', 'jr01')
```

The UPDATE statement

Syntax

```
UPDATE table-name
SET expression-1 [, expression-2] ...
WHERE selection-criteria
```

A statement that updates the FirstName column in one row

```
UPDATE User
SET FirstName = 'Jack'
WHERE Email = 'jsmith@gmail.com'
```

A statement that updates the ProductPrice column in selected rows

```
UPDATE Product
SET ProductPrice = 36.95
WHERE ProductPrice = 36.50
```

The DELETE statement

Syntax

```
DELETE FROM table-name
WHERE selection-criteria
```

A statement that deletes one row from the User table

```
DELETE FROM User WHERE Email = 'jsmith@gmail.com'
```

A statement that deletes selected rows from the Downloads table

```
DELETE FROM Download WHERE DownloadDate < '2014-06-01'
```

How to insert, update, and delete data

- INSERT, UPDATE, and DELETE statements modify the data that's stored in a database, but they don't return a result set. Instead, they return the number of rows that were affected by the query.
- These statements are sometimes referred to as *action queries*.

How to use JDBC to work with a database

Objectives

Applied

1. Develop data access classes that use JDBC to provide all of the methods that your servlets need to work with a database.
2. Develop a utility class that allows you to get a connection from a connection pool.
3. Develop servlets that use the methods of your data classes.

Objectives (continued)

Knowledge

1. Describe how a web application can use the DriverManager, Connection, Statement, PreparedStatement, and ResultSet classes to get data from a database.
2. Explain how prepared statements can improve the performance and security of database operations.
3. Describe the use of a ResultSetMetaData object.
4. Explain how connection pooling can improve the performance of a web application.
5. Describe O/R (object-relational) mapping.

The four types of JDBC database drivers

Type 1	A *JDBC-ODBC bridge driver* converts JDBC calls into ODBC calls that access the DBMS protocol.
Type 2	A *native protocol partly Java driver* converts JDBC calls into calls in the native DBMS protocol.
Type 3	A *net protocol all Java driver* converts JDBC calls into a net protocol that's independent of any native DBMS protocol.
Type 4	A *native protocol all Java driver* converts JDBC calls into a native DBMS protocol.

How to make a database driver available to an application

- Before you can use a database driver, you must make it available to your application. To do this, add the JAR file for the driver to your application classpath.
- To add the MySQL JDBC driver to you application, add the jar file to class libraries folder.

Database URL syntax

```
jdbc:subprotocolName:databaseURL
```

How to connect to a MySQL database with automatic driver loading

```
try {
    String dbURL = "jdbc:mysql://localhost:3306/murach";
    String username = "root";
    String password = "sesame";
    Connection connection = DriverManager.getConnection(
        dbURL, username, password);
} catch(SQLException e) {
    for (Throwable t : e)
        t.printStackTrace();
}
```

How to connect to an Oracle database with automatic driver loading

```
    Connection connection = DriverManager.getConnection(
        "jdbc:oracle:thin@localhost/murach", "scott", "tiger");
```

How to explicitly load a database driver

```
try {
    Class.forName("com.mysql.jdbc.Driver");
} catch(ClassNotFoundException e) {
    e.printStackTrace();
}
```

How to connect to a database

- Before you can get or modify the data in a database, you need to connect to it. To do that, use the getConnection method of the DriverManager class to return a Connection object.
- When you use the getConnection method of the DriverManager class, you must supply a URL for the database, a username, and a password. This method throws an SQLException.
- With JDBC 4.0, the SQLException class implements the Iterable interface. As a result, use an enhanced for statement to loop through any nested exceptions.
- **With JDBC 4.0, the database driver is loaded automatically. This is known as *automatic driver loading*.**
- Typically, you only need to connect to one database for an application. However, it's possible to load multiple database drivers and establish connections to multiple types of databases.

How to create a result set with 1 row and 1 column

```
Statement statement = connection.createStatement();
ResultSet userIDResult = statement.executeQuery(
    "SELECT UserID FROM User " +
    "WHERE Email = 'jsmith@gmail.com'");
```

How to create a result set with multiple columns and rows

```
Statement statement = connection.createStatement();
ResultSet products = statement.executeQuery(
    "SELECT * FROM Product ");
```

How to move the cursor to the first row

```
boolean userIDExists = userIDResult.next();
```

How to loop through a result set

```
while (products.next()) {
    // statements that process each row
}
```

ResultSet methods for forward-only, read-only result sets

Method	Description
`next()`	Moves the cursor to the next row in the result set.
`last()`	Moves the cursor to the last row in the result set.
`close()`	Releases the result set's resources.
`getRow()`	Returns an int value that identifies the current row of the result set.

How to return a result set and move the cursor through it

- To return a *result set*, use the createStatement method of a Connection object to create a Statement object. Use the executeQuery method of the Statement object to execute a SELECT statement that returns a ResultSet object.
- By default, the createStatement method creates a forward-only, read-only result set. You can only move the *cursor* through it from the first row to the last and you can't update it. This is appropriate for most web applications.
- When a result set is created, the cursor is positioned before the first row.
- Use the methods of the ResultSet object to move the cursor.
- To move the cursor to the next row, call the next method. If the row is valid, this method moves the cursor to the next row and returns a true value.

Methods of a ResultSet object that return data

Method	Description
`getXXX(int columnIndex)`	Returns data from the specified column number.
`getXXX(String columnName)`	Returns data from the specified column name.

Code that uses indexes to return columns

```
String code = products.getString(1);
String description = products.getString(2);
double price = products.getDouble(3);
```

Code that uses names to return columns

```
String code = products.getString("ProductCode");
String description = products.getString("ProductDescription");
double price = products.getDouble("ProductPrice");
```

Code that creates a Product object from the products result set

```
Product product = new Product(products.getString(1),
                              products.getString(2),
                              products.getDouble(3));
```

The getXXX methods

- The getXXX methods can be used to return all eight primitive types. For example, the getInt method returns the int type and the getLong method returns the long type.
- The getXXX methods can also be used to return strings, dates, and times. For example, the getString method returns any object of the String class, and the getDate, getTime, and getTimestamp methods return objects of the Date, Time, and Timestamp classes of the java.sql package.

How to use the executeUpdate method to…

Add a row

```
String query =
   "INSERT INTO Product (ProductCode, ProductDescription, ProductPrice) " +
   "VALUES ('" + product.getCode() + "', " +
           "'" + product.getDescription() + "', " +
           "'" + product.getPrice() + "')";
Statement statement = connection.createStatement();
int rowCount = statement.executeUpdate(query);
```

Update a row

```
String query = "UPDATE Product SET " +
   "ProductCode = '" + product.getCode() + "', " +
   "ProductDescription = '" + product.getDescription() + "', " +
   "ProductPrice = '" + product.getPrice() + "' " +
   "WHERE ProductCode = '" + product.getCode() + "'";
Statement statement = connection.createStatement();
int rowCount = statement.executeUpdate(query);
```

How to use the executeUpdate method to…(cont.)

Delete a row

```
String query = "DELETE FROM Product " +
               "WHERE ProductCode = '" + productCode + "'";
Statement statement = connection.createStatement();
int rowCount = statement.executeUpdate(query);
```

How to insert, update, and delete data

- The executeUpdate method is an older method that works with most JDBC drivers. Some newer methods require less SQL code, but they may not work properly with all JDBC drivers.
- The executeUpdate method returns an int value that identifies the number of rows affected by the SQL statement.

Warning

- If you build an SQL statement from user input and use a method of the Statement object to execute that SQL statement, you may be susceptible to an SQL injection attack.
- An *SQL injection attack* allows a hacker to bypass authentication or execute SQL statements against your database that can read data, modify data, or delete data.
- To prevent most types of SQL injection attacks, use prepared statements.

How to use a prepared statement

To return a result set

```
String preparedSQL = "SELECT ProductCode, ProductDescription, "
                   + "       ProductPrice "
                   + "FROM Product WHERE ProductCode = ?";
PreparedStatement ps = connection.prepareStatement(preparedSQL);
ps.setString(1, productCode);
ResultSet product = ps.executeQuery();
```

To modify a row

```
String preparedSQL = "UPDATE Product SET "
                   + "   ProductCode = ?, "
                   + "   ProductDescription = ?, "
                   + "   ProductPrice = ?"
                   + "WHERE ProductCode = ?";
PreparedStatement ps = connection.prepareStatement(preparedSQL);
ps.setString(1, product.getCode());
ps.setString(2, product.getDescription());
ps.setDouble(3, product.getPrice());
ps.setString(4, product.getCode());
ps.executeUpdate();
```

How to use a prepared statement (continued)

To insert a row

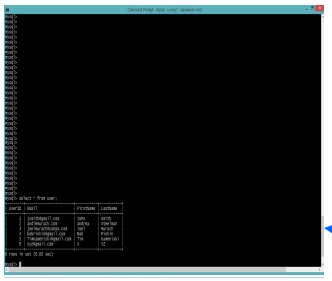
```
String preparedQuery =
    "INSERT INTO Product "
    + "(ProductCode, ProductDescription, ProductPrice) "
    + "VALUES "
    + "(?, ?, ?)";
PreparedStatement ps = connection.prepareStatement(preparedQuery);
ps.setString(1, product.getCode());
ps.setString(2, product.getDescription());
ps.setDouble(3, product.getPrice());
ps.executeUpdate();
```

To delete a row

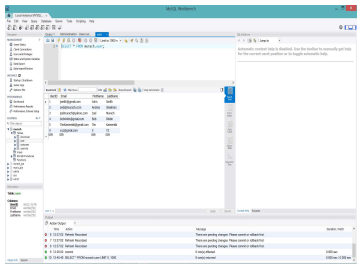
```
String preparedQuery = "DELETE FROM Product "
                     + "WHERE ProductCode = ?";
PreparedStatement ps = connection.prepareStatement(preparedQuery);
ps.setString(1, productCode);
ps.executeUpdate();
```

This is the skyview for Chicago

This is the skyview for mySQLapp1 and mySQLapp2

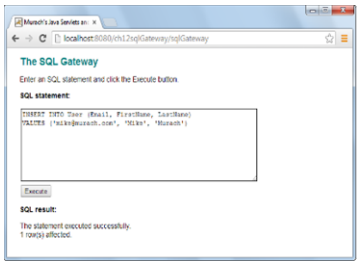


MySQL Shell

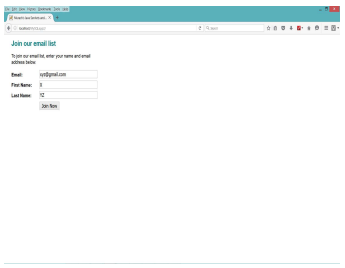


MySQL Workbench

MySQL DB



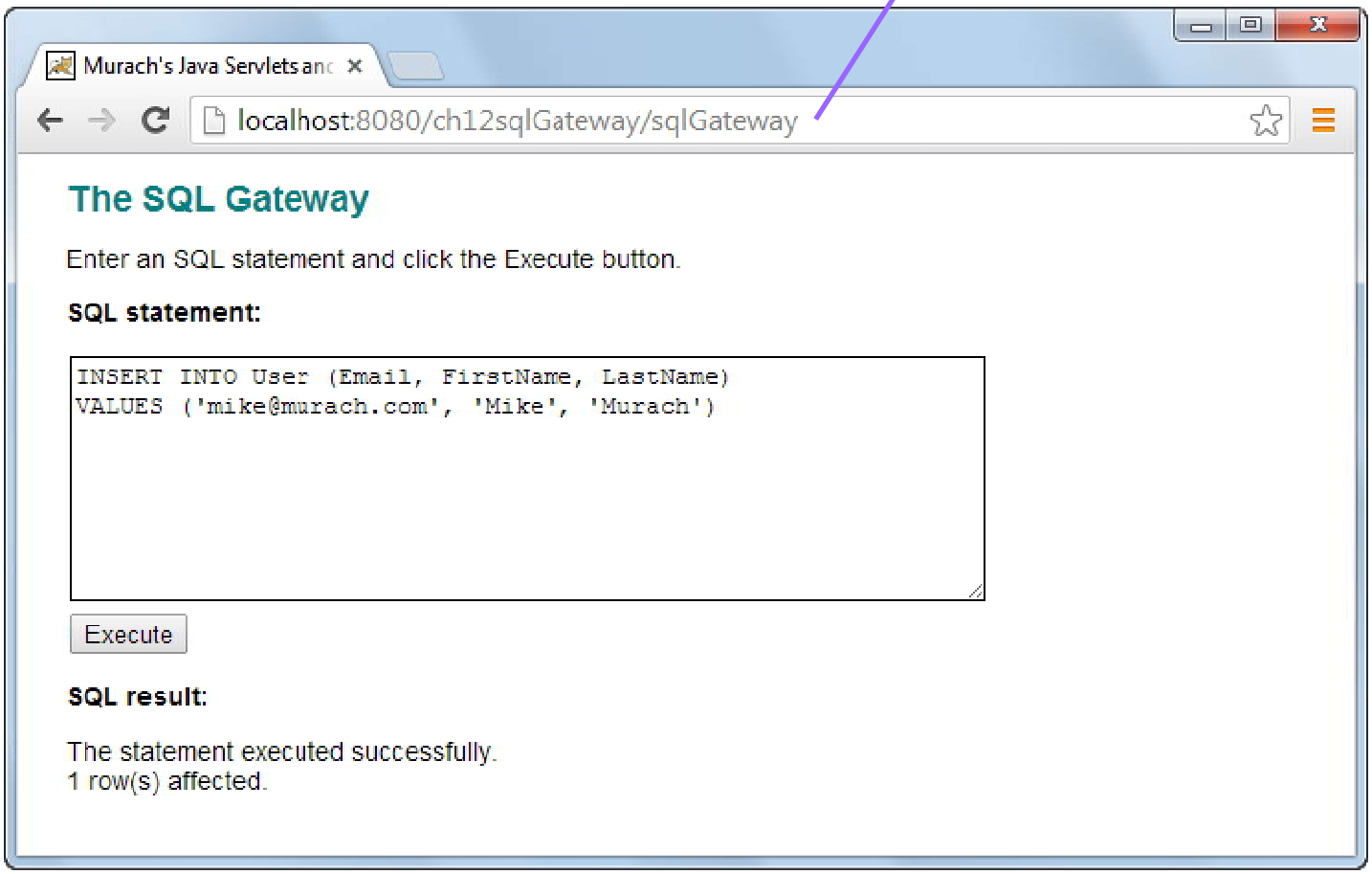
MySQLapp1



MySQLapp2

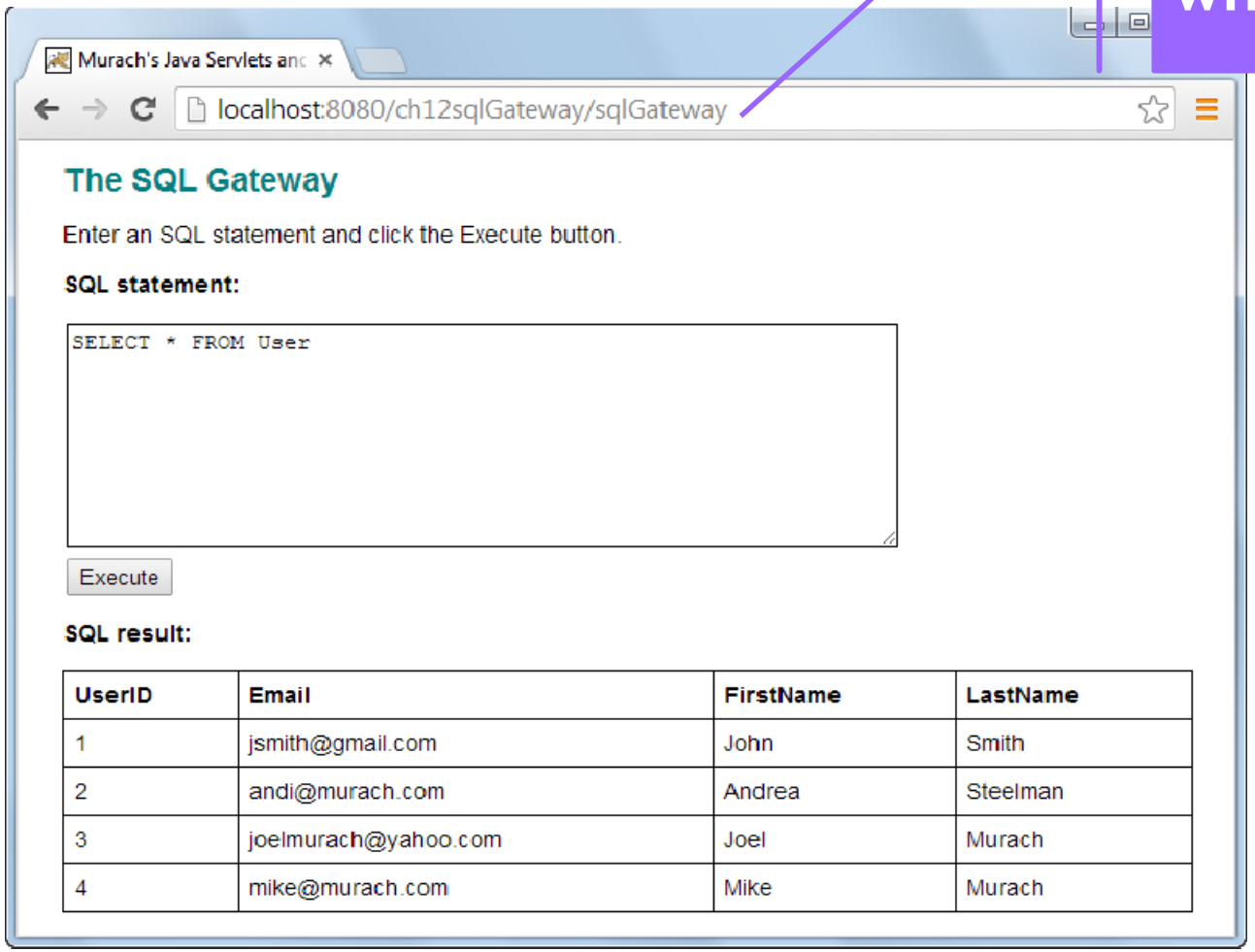
The SQL Gateway application after executing an INSERT statement

This is the servlet used to insert the row below

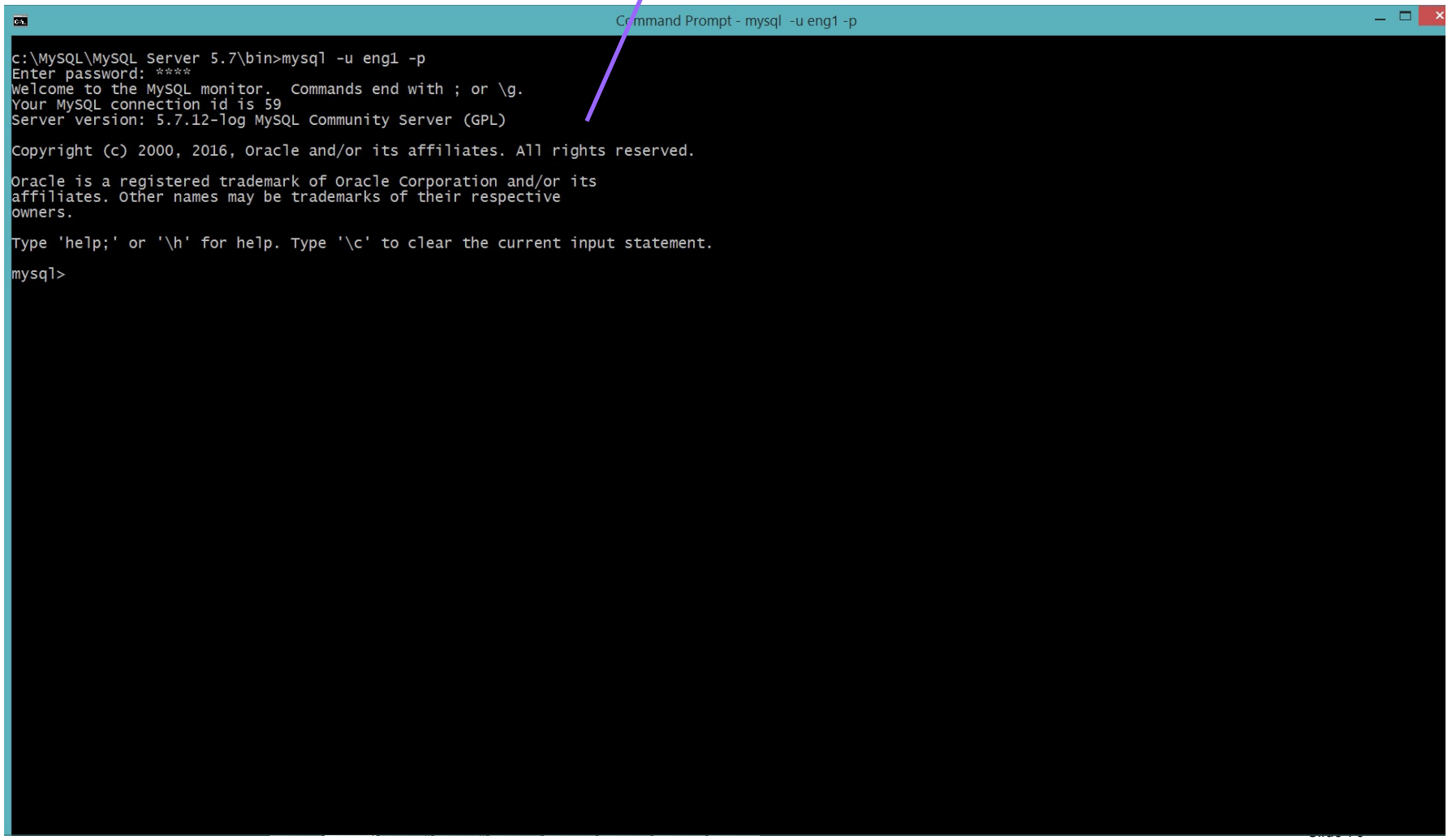


The SQL Gateway application after executing a SELECT statement

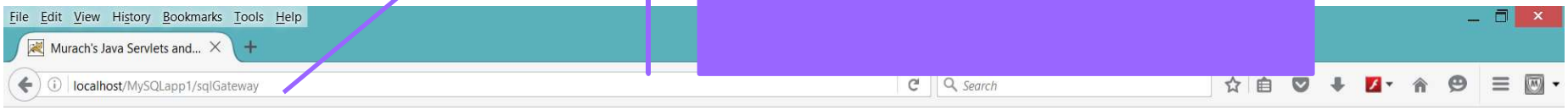
This is the servlet used to retrieve what got inserted



You could also use MySQL shell to see whats in the database



This is the servlet used to query the table before inserting a row



The SQL Gateway

Enter an SQL statement and click the Execute button.

SQL statement:

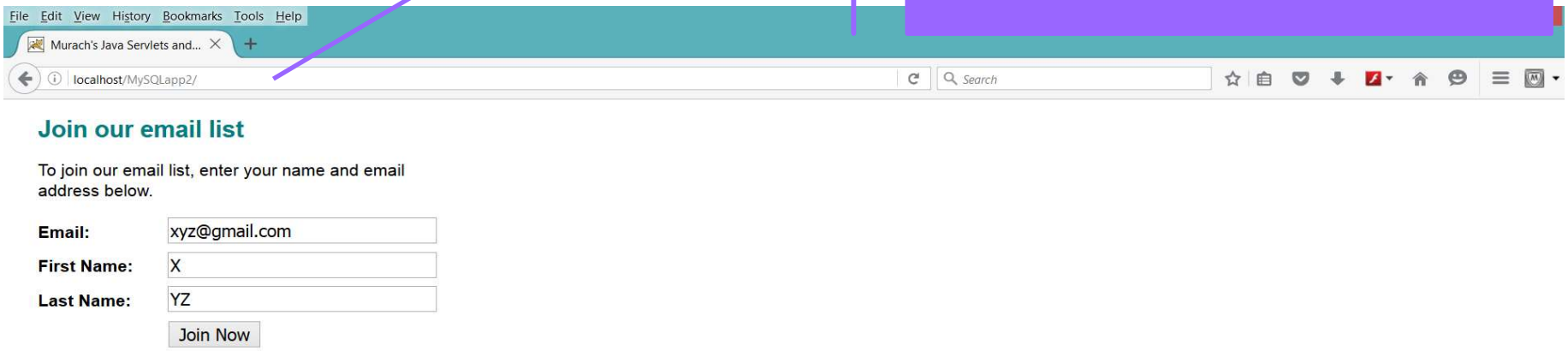
```
select * from User
```

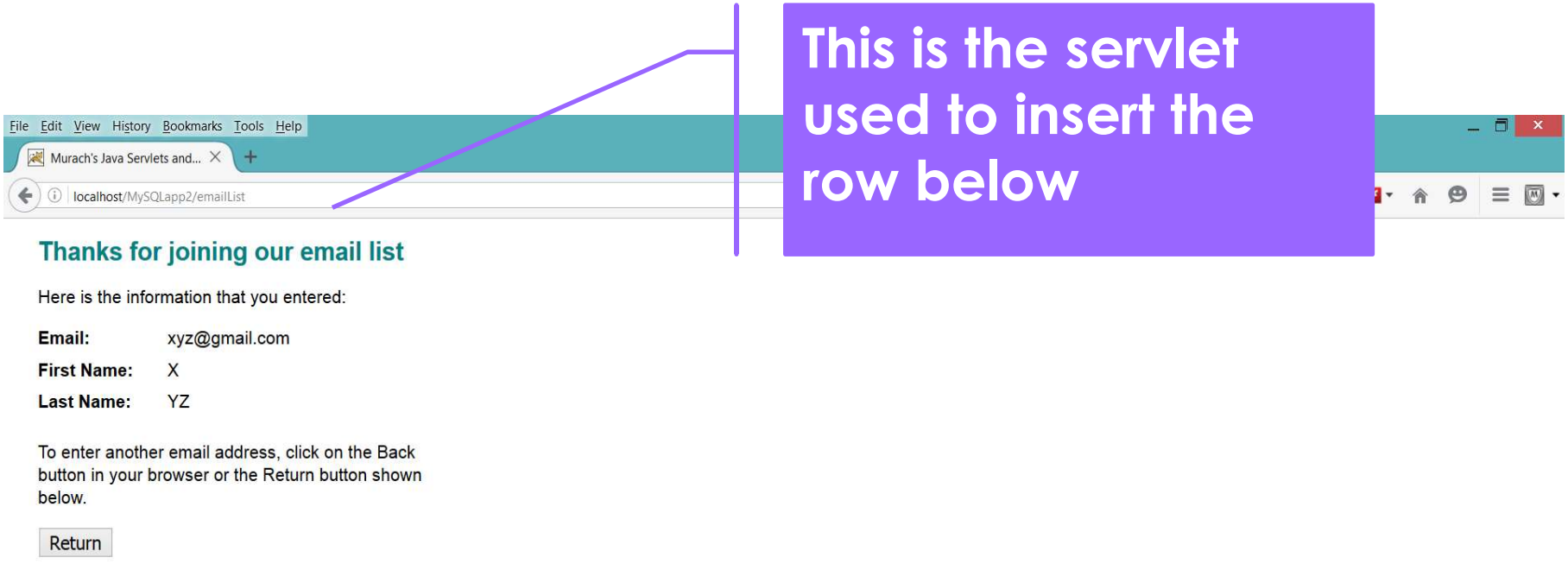
Execute

SQL result:

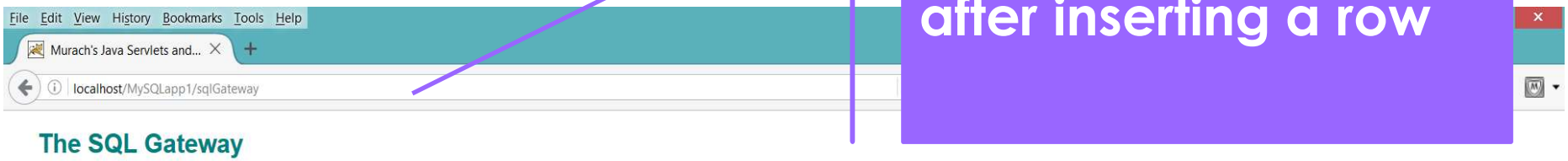
UserID	Email	FirstName	LastName
1	jsmith@gmail.com	John	Smith
2	andi@murach.com	Andrea	Steelman
3	joelmurach@yahoo.com	Joel	Murach
4	bobrinkin@gmail.com	Bob	Rinkin
5	TimKameniski@gmail.com	Tim	Kameniski

This is the index.jsp used to call EmailListServlet to insert a row into DB




File Edit View History Bookmarks Tools Help
Murach's Java Servlets and...
localhost/MySQLapp2/emailList
Thanks for joining our email list
Here is the information that you entered:
Email: xyz@gmail.com
First Name: X
Last Name: YZ
To enter another email address, click on the Back button in your browser or the Return button shown below.
Return
This is the servlet used to insert the row below

This is the servlet used to query the table after inserting a row



The SQL Gateway

Enter an SQL statement and click the Execute button.

SQL statement:

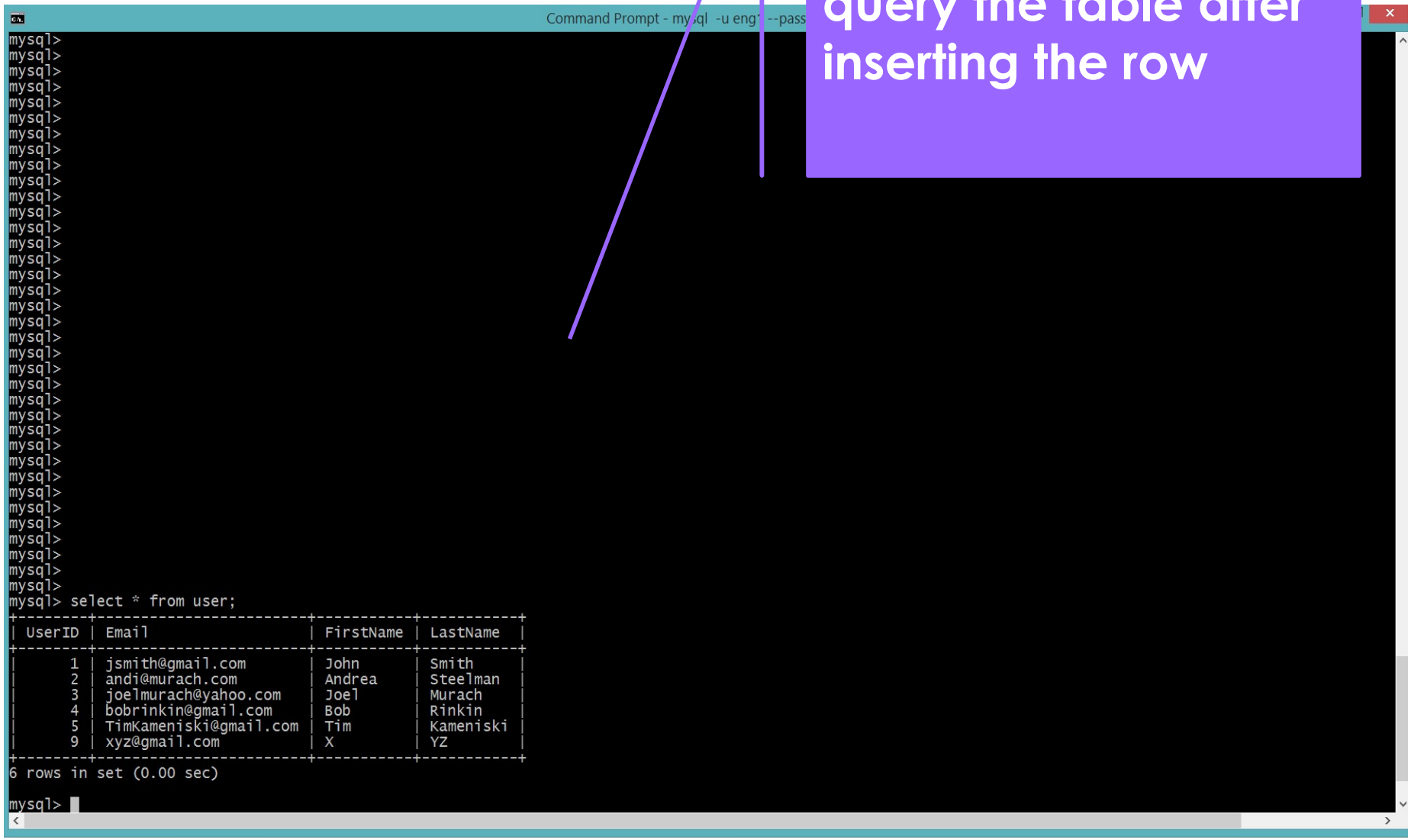
```
select * from User
```

Execute

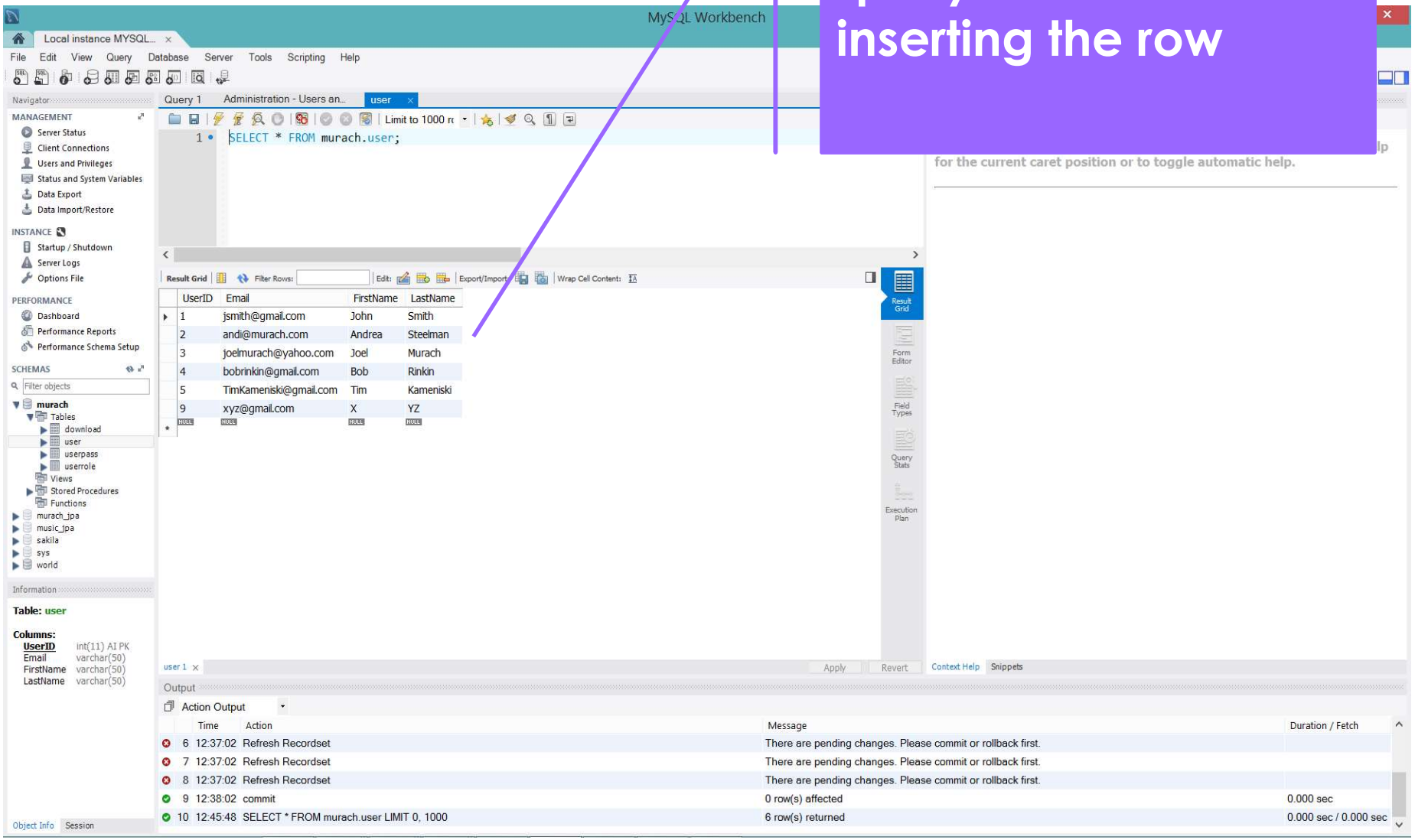
SQL result:

UserID	Email	FirstName	LastName
1	jsmith@gmail.com	John	Smith
2	andi@murach.com	Andrea	Steelman
3	joelmurach@yahoo.com	Joel	Murach
4	bobrinkin@gmail.com	Bob	Rinkin
5	TimKameniski@gmail.com	Tim	Kameniski
9	xyz@gmail.com	X	YZ

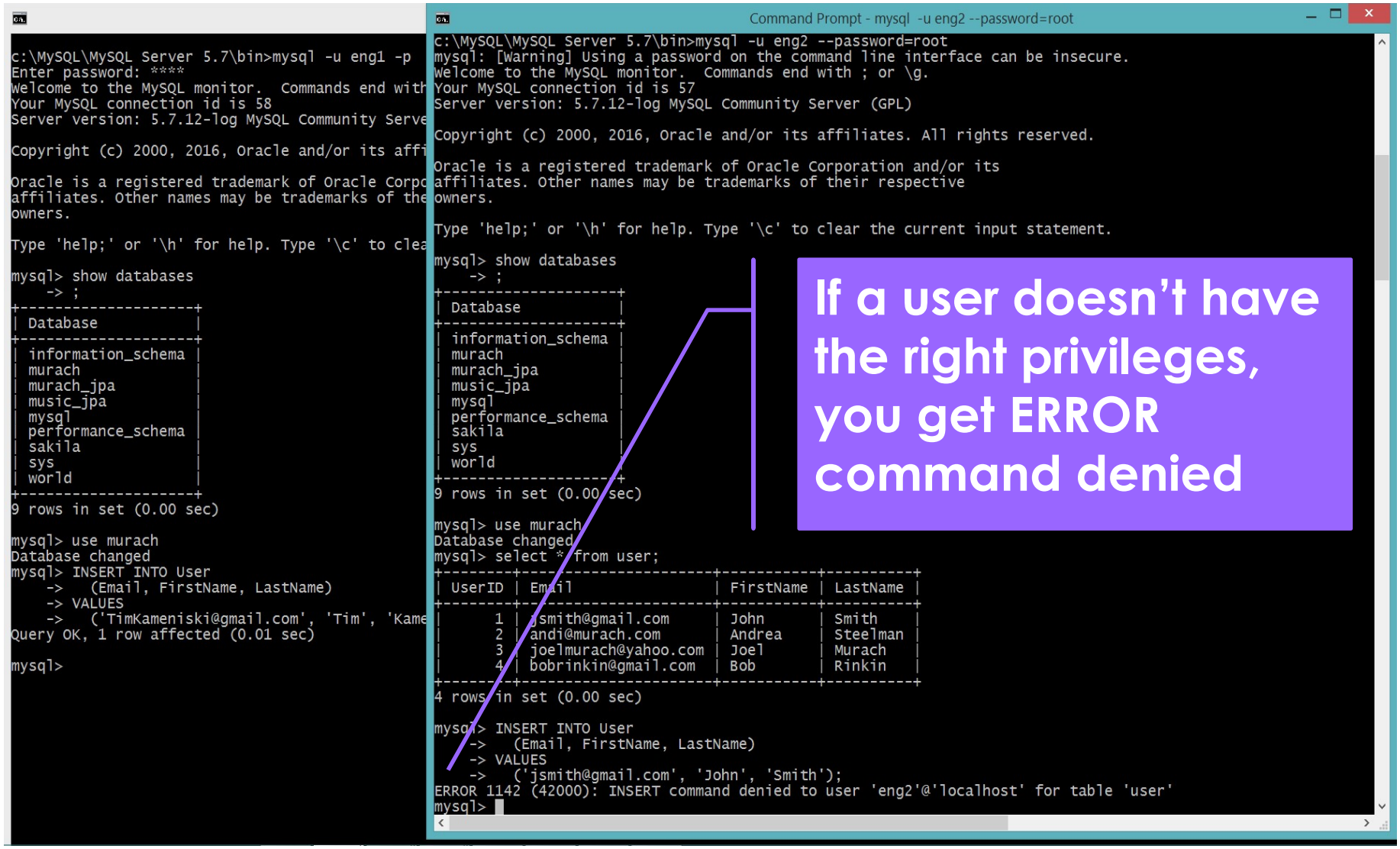
From MySQL shell query the table after inserting the row



From Workbench query the table after inserting the row

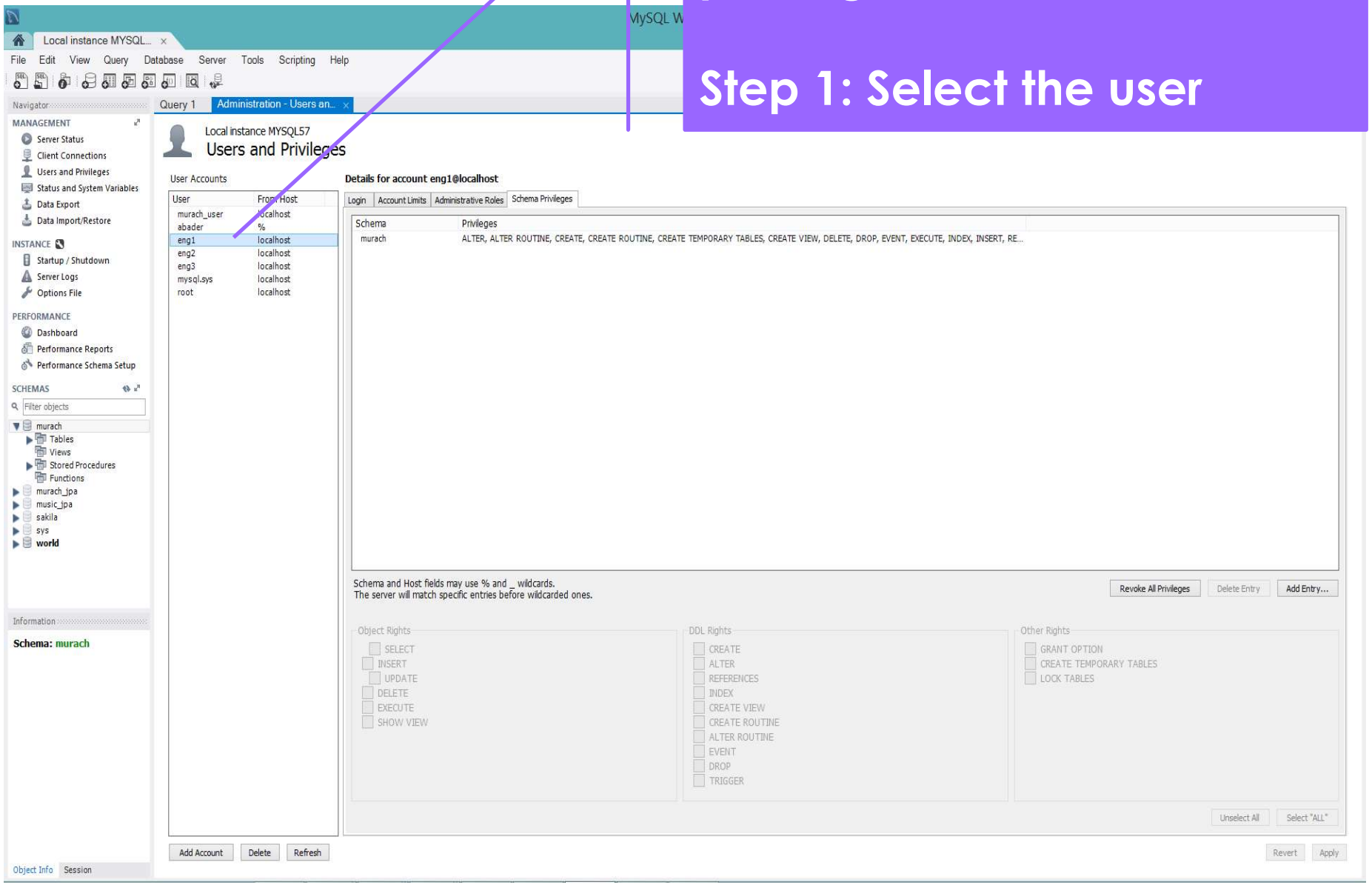


Verify the users, roles, and previliges



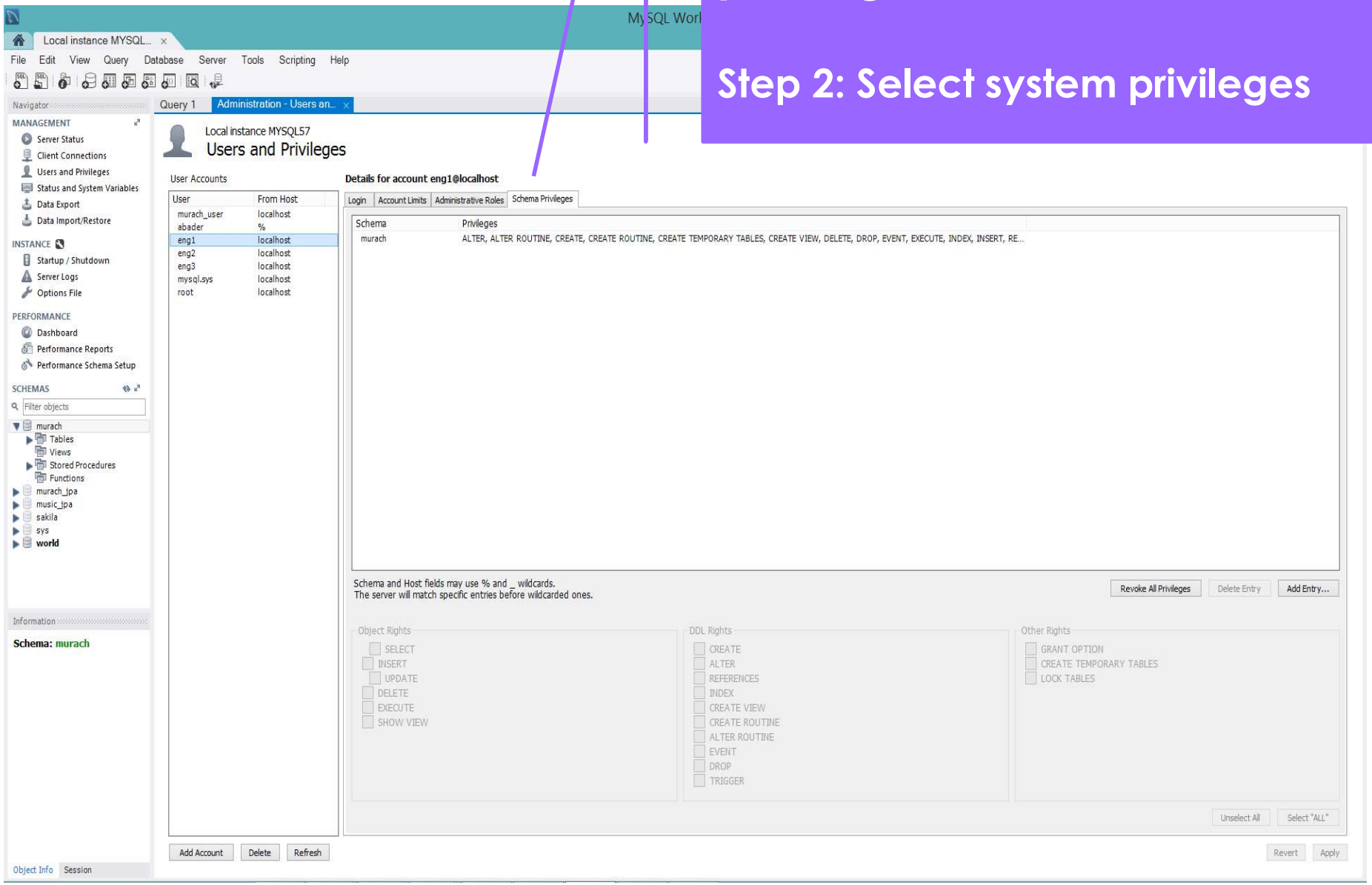
Giving eng1 the right privileges

Step 1: Select the user



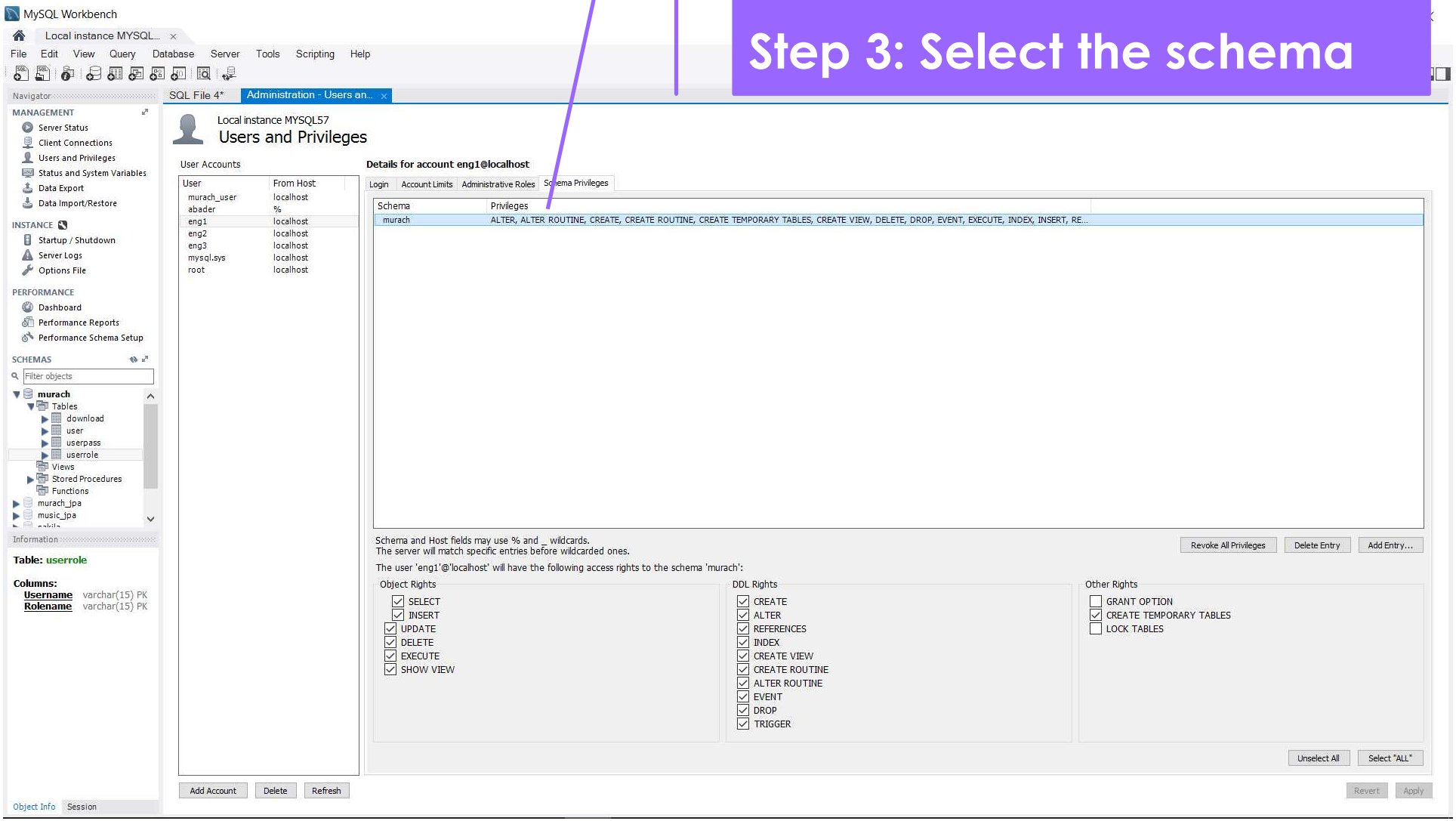
Giving eng1 the right privileges

Step 2: Select system privileges



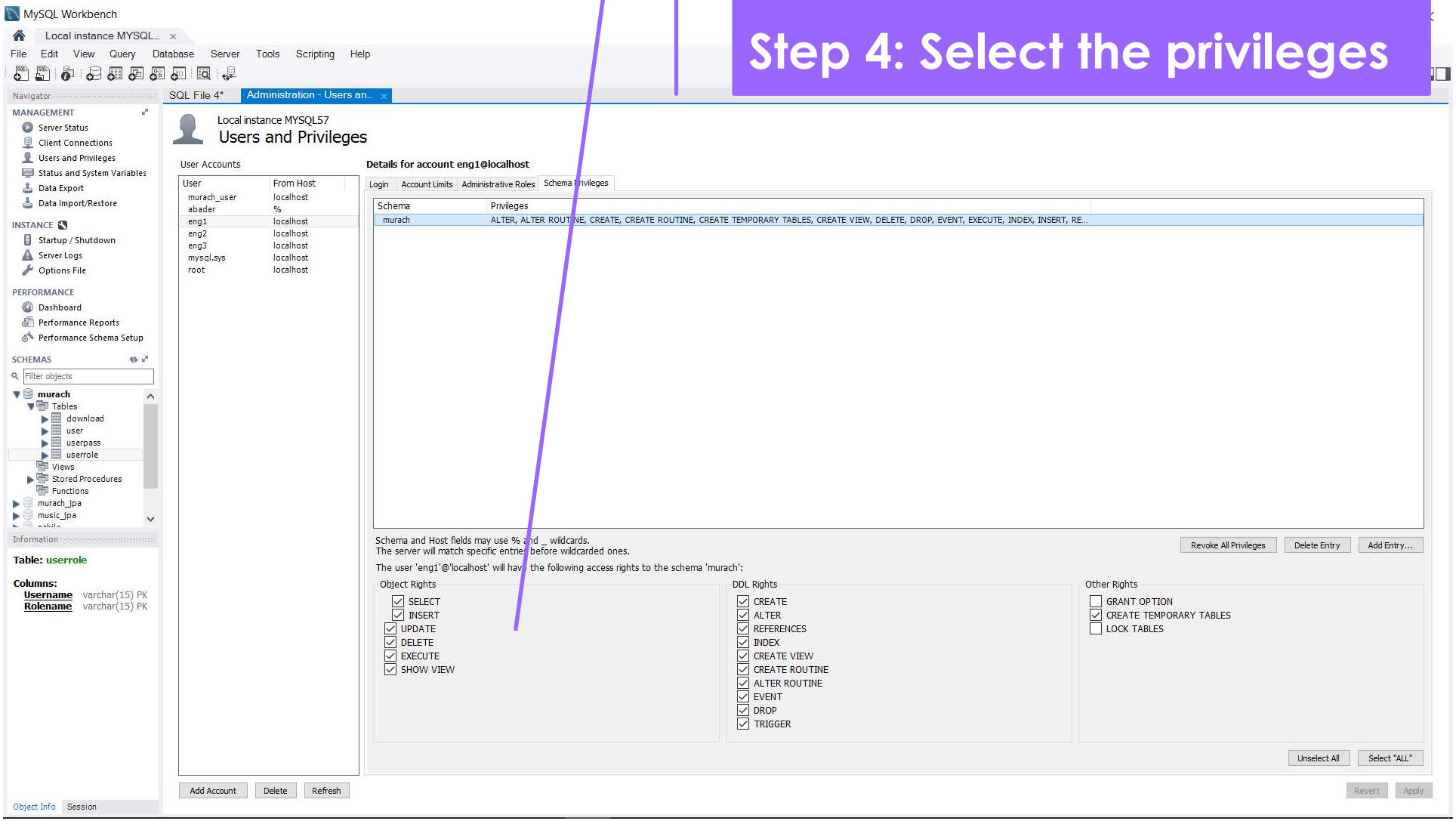
Giving eng1 the right privileges

Step 3: Select the schema

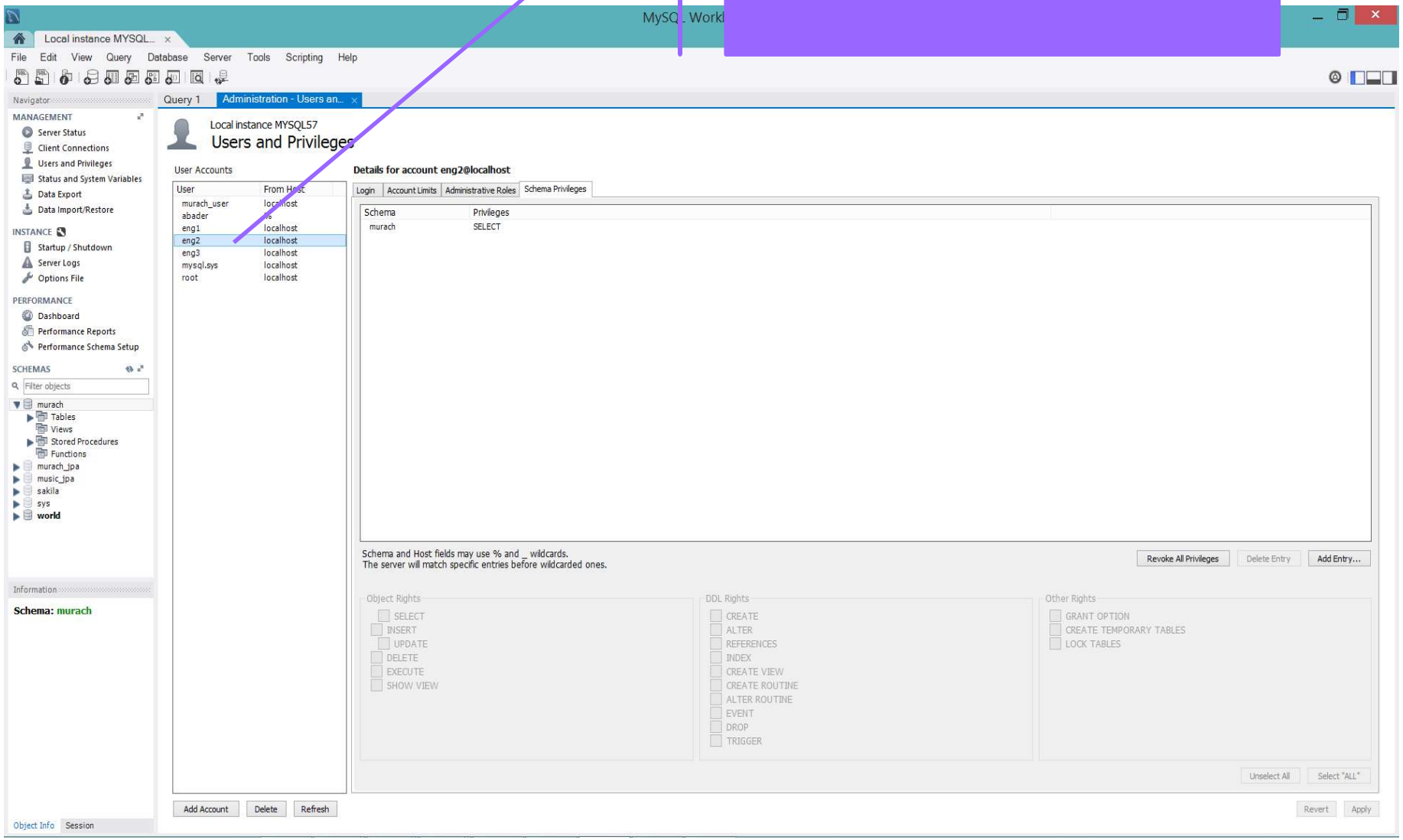


Giving eng1 the right privileges

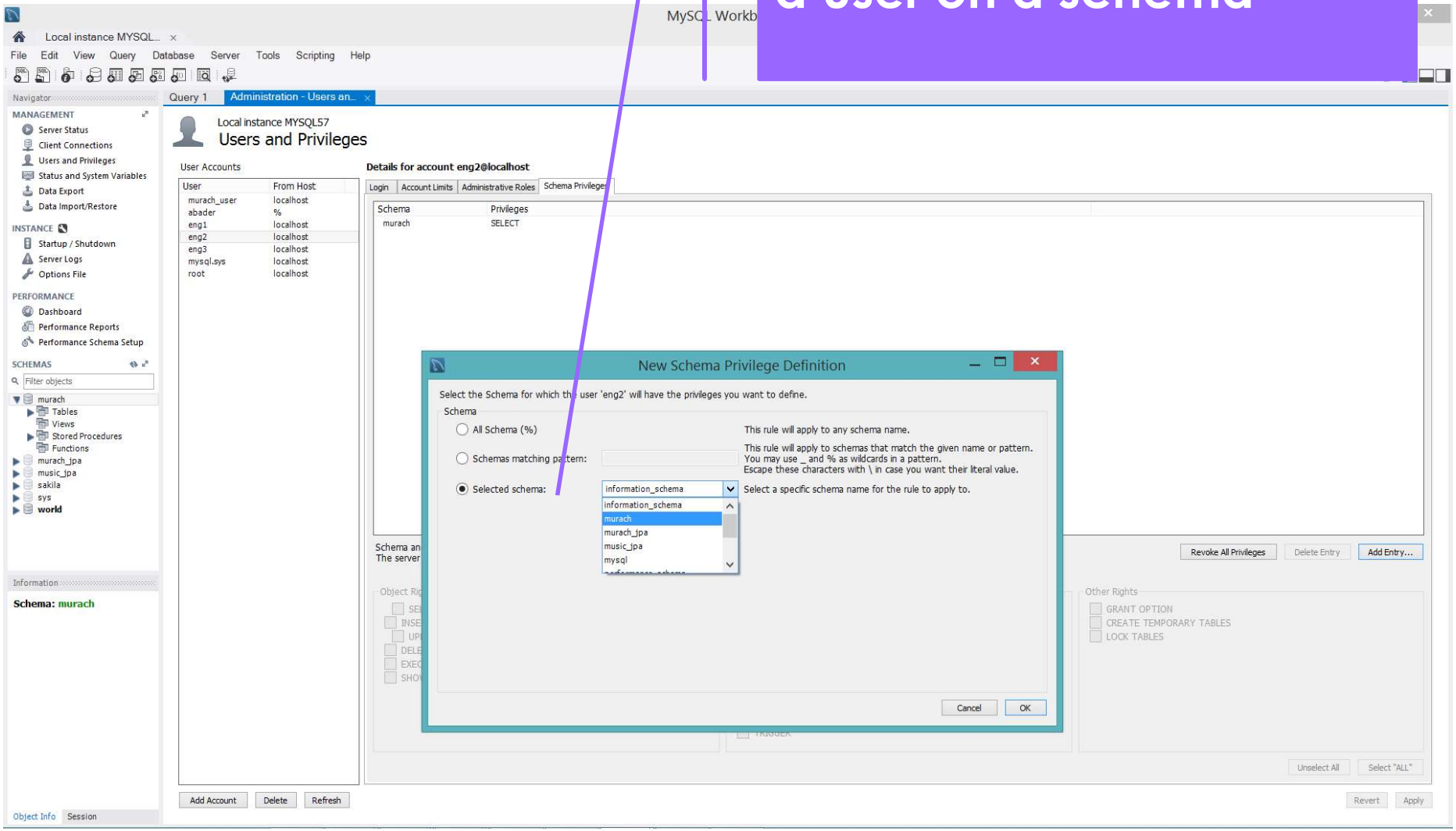
Step 4: Select the privileges



Giving eng2 the right privileges

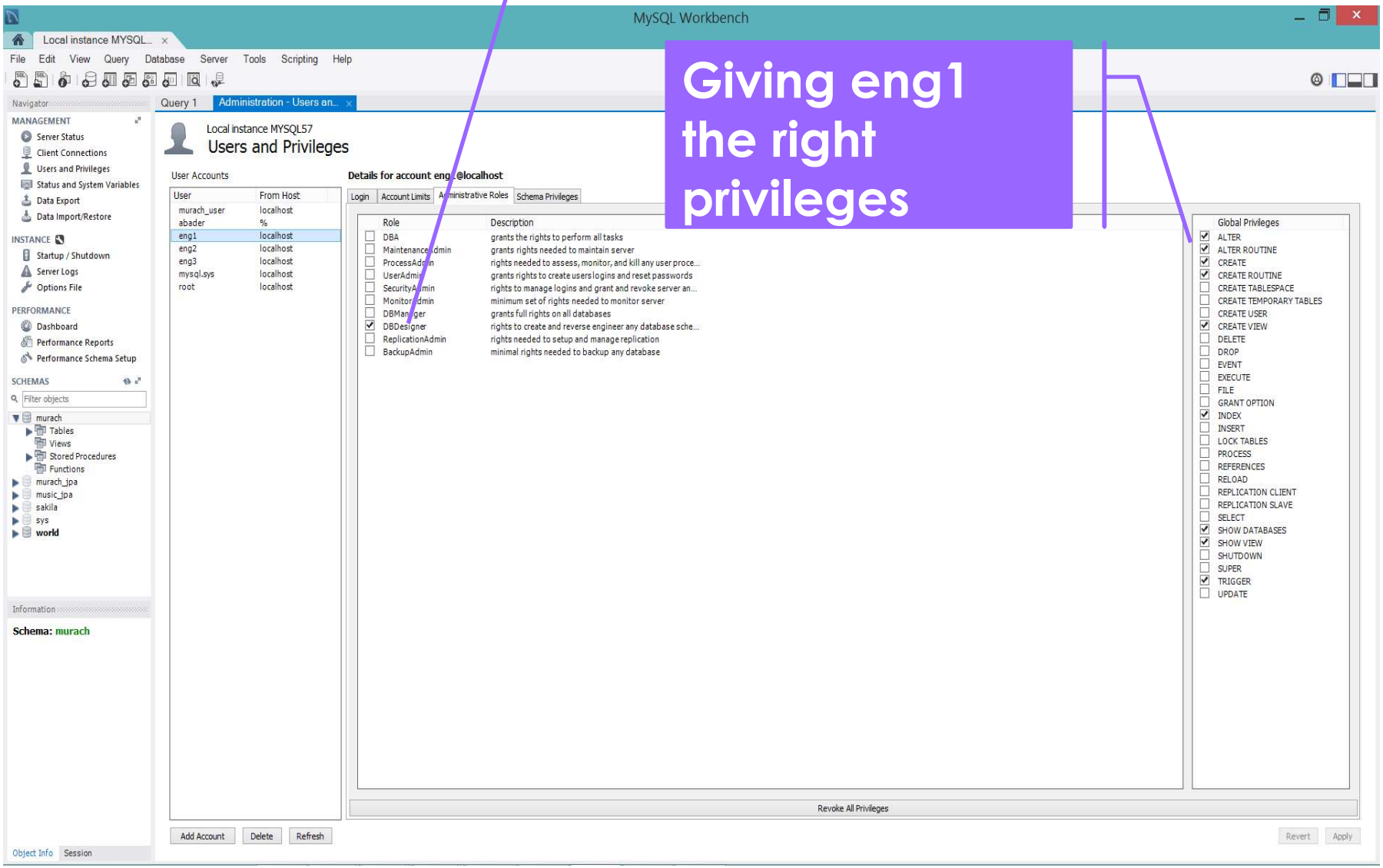


How to give privileges to a user on a schema



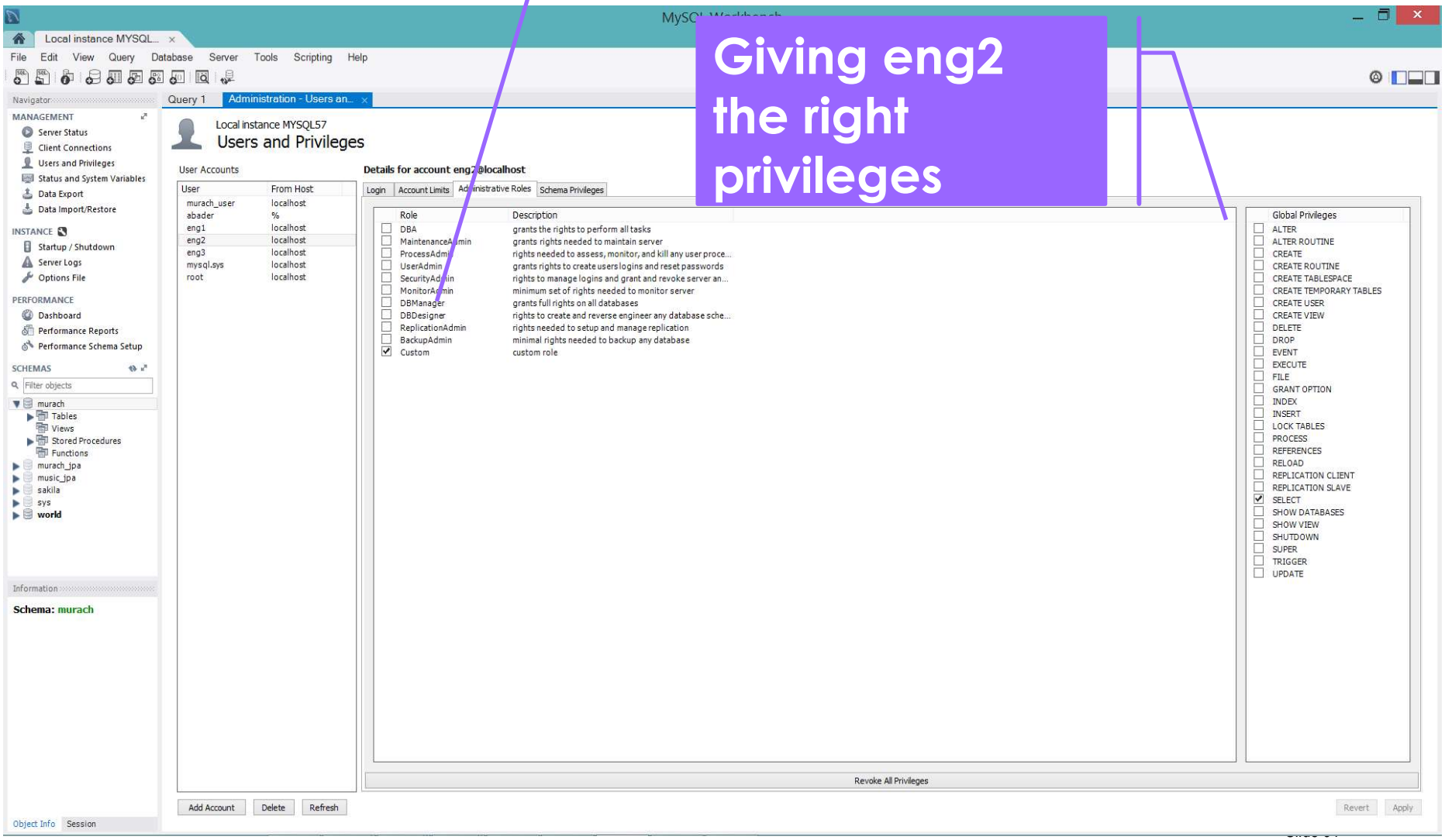
Giving eng1 the right roles

Giving eng1 the right privileges

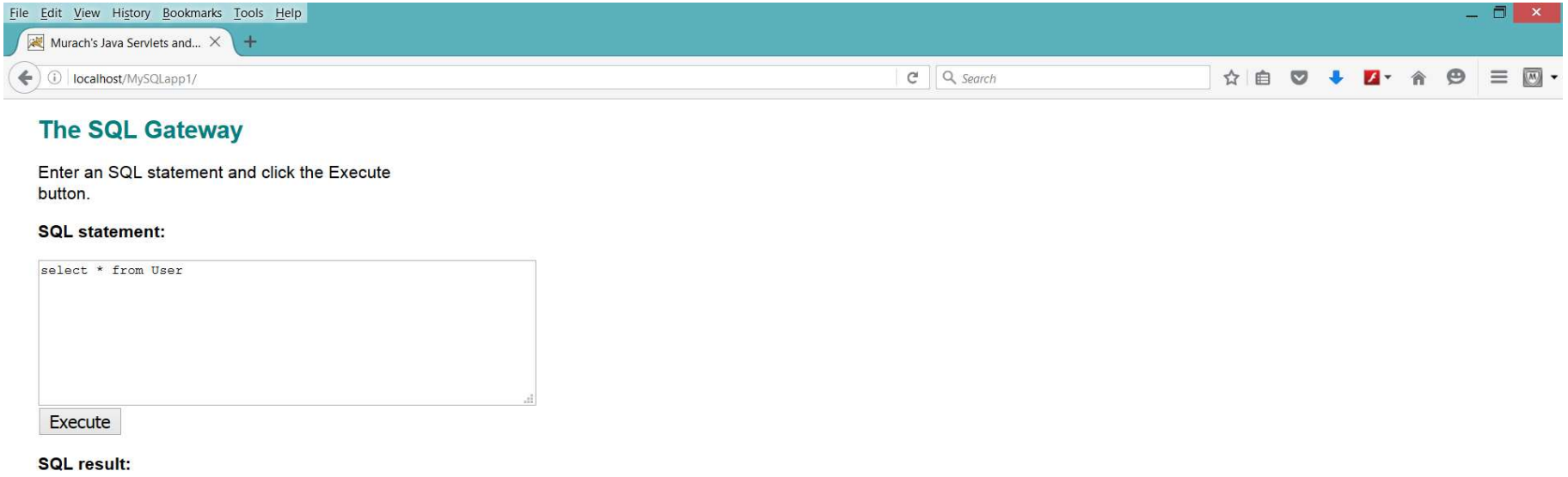


Giving eng2 the right roles

Giving eng2 the right privileges

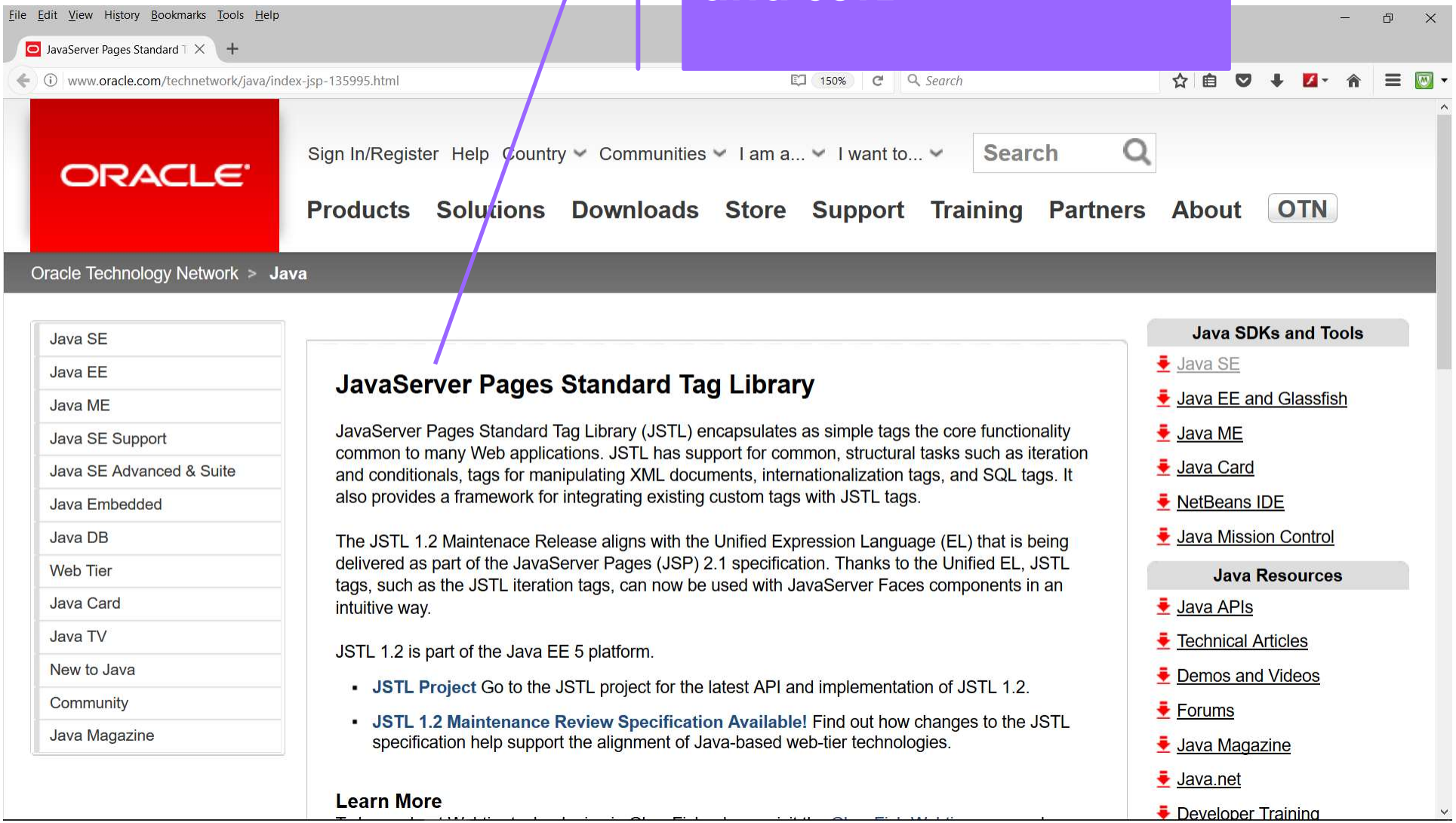


MySQLapp1

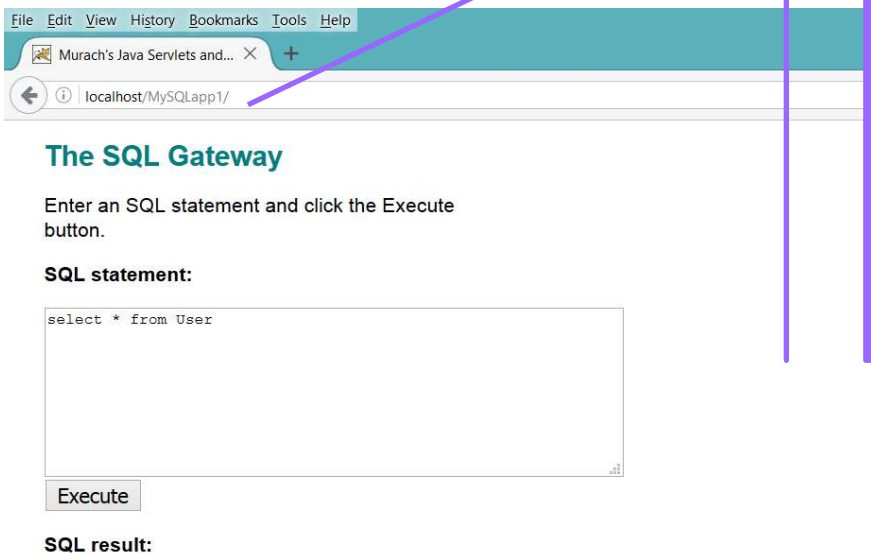

File Edit View History Bookmarks Tools Help
Murach's Java Servlets and...
localhost/MySQLapp1/
Search
The SQL Gateway
Enter an SQL statement and click the Execute button.
SQL statement:
select * from User
Execute
SQL result:

MySQLapp1

MySQLapp1 uses JSP and JSTL



MySQLapp1



Remember Tomcat has default page names that it defines in the web.xml of the web-app

```
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

The code for the JSP

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:if test="${sqlStatement == null}">
    <c:set var="sqlStatement" value="select * from User" />
</c:if>

<h1>The SQL Gateway</h1>
<p>Enter an SQL statement and click the Execute button.</p>
```

The code for the JSP (continued)

```
<p><b>SQL statement:</b></p>
<form action="sqlGateway" method="post">
   <textarea name="sqlStatement" cols="60"
rows="8">${sqlStatement}</textarea>
   <input type="submit" value="Execute">
</form>

<p><b>SQL result:</b></p>
${sqlResult}

</body>
</html>
```

The SQLGatewayServlet class

```
package murach.sql;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import java.sql.*;

public class SqlGatewayServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {

        String sqlStatement = request.getParameter("sqlStatement");
        String sqlResult = "";
        try {
            // load the driver
            Class.forName("com.mysql.jdbc.Driver");
```

The SQLGatewayServlet class (continued)

```
// get a connection
            String dbURL = "jdbc:mysql://localhost:3306/murach";
            String username = "murach_user";
            String password = "sesame";
            Connection connection = DriverManager.getConnection(
                    dbURL, username, password);

            // create a statement
            Statement statement = connection.createStatement();

            // parse the SQL string
            sqlStatement = sqlStatement.trim();
            if (sqlStatement.length() >= 6) {
                String sqlType = sqlStatement.substring(0, 6);

                if (sqlType.equalsIgnoreCase("select")) {
                    // create the HTML for the result set
                    ResultSet resultSet
                            = statement.executeQuery(sqlStatement);
                    sqlResult = SQLUtil.getHtmlTable(resultSet);
                    resultSet.close();
```

The SQLGatewayServlet class (continued)

```
            } else {
                int i = statement.executeUpdate(sqlStatement);
                if (i == 0) { // a DDL statement
                    sqlResult =
                      "<p>The statement executed successfully.</p>";
                } else { // an INSERT, UPDATE, or DELETE statement
                    sqlResult =
                      "<p>The statement executed successfully.<br>"
                            + i + " row(s) affected.</p>";
                }
            }
        }
        statement.close();
        connection.close();
    } catch (ClassNotFoundException e) {
        sqlResult = "<p>Error loading the database driver: <br>"
                + e.getMessage() + "</p>";
    } catch (SQLException e) {
        sqlResult = "<p>Error executing the SQL statement: <br>"
                + e.getMessage() + "</p>";
    }
```

The SQLGatewayServlet class (continued)

```
        HttpSession session = request.getSession();
        session.setAttribute("sqlResult", sqlResult);
        session.setAttribute("sqlStatement", sqlStatement);

        String url = "/index.jsp";
        getServletContext()
                .getRequestDispatcher(url)
                .forward(request, response);
    }
}
```

Note

- The web.xml file for this application maps the SQLGatewayServlet class to the /sqlGateway URL.

The SQLUtil class

```
package murach.sql;

import java.sql.*;

public class SQLUtil {

    public static String getHtmlTable(ResultSet results)
            throws SQLException {

        StringBuilder htmlTable = new StringBuilder();
        ResultSetMetaData metaData = results.getMetaData();
        int columnCount = metaData.getColumnCount();

        htmlTable.append("<table>");
```

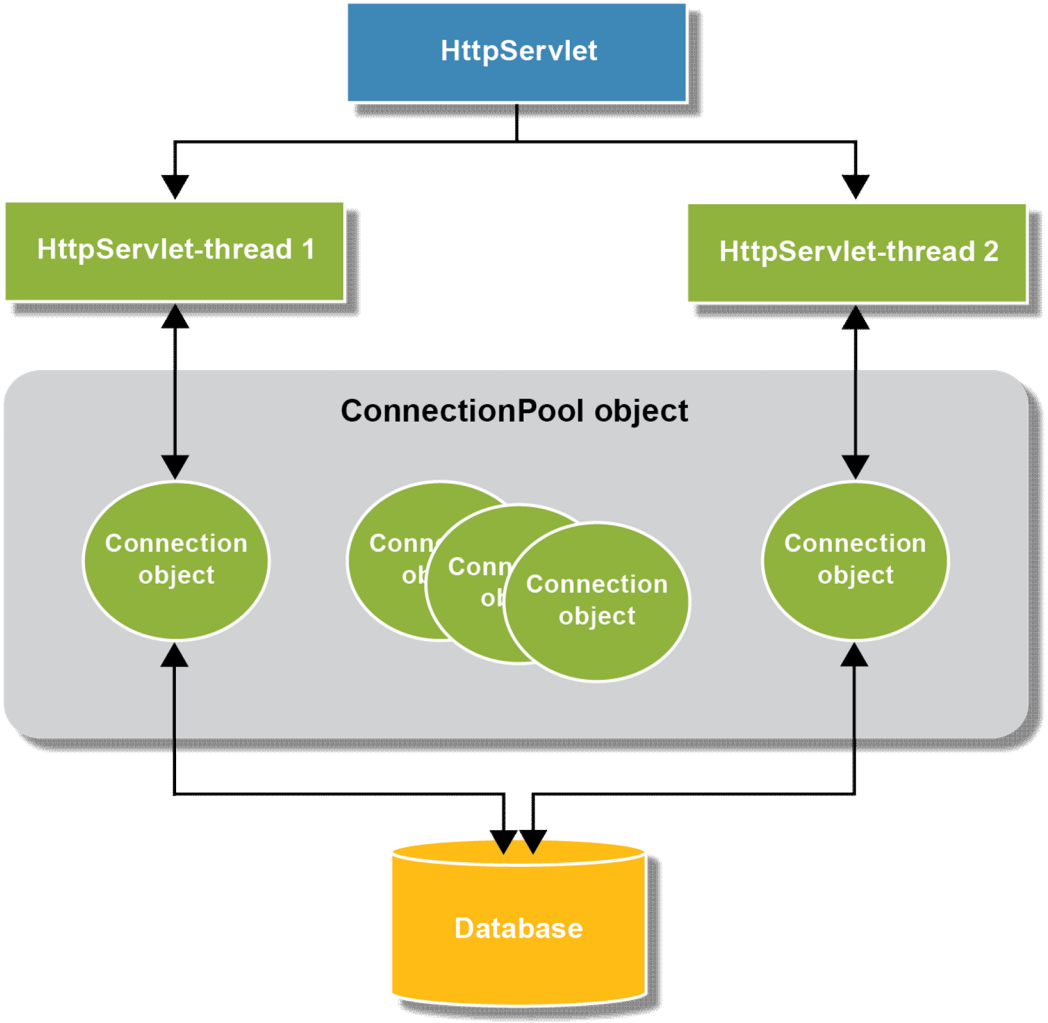
The SQLUtil class (continued)

```
        // add header row
        htmlTable.append("<tr>");
        for (int i = 1; i <= columnCount; i++) {
            htmlTable.append("<th>");
            htmlTable.append(metaData.getColumnName(i));
            htmlTable.append("</th>");
        }
        htmlTable.append("</tr>");

        // add all other rows
        while (results.next()) {
            htmlTable.append("<tr>");
            for (int i = 1; i <= columnCount; i++) {
                htmlTable.append("<td>");
                htmlTable.append(results.getString(i));
                htmlTable.append("</td>");
            }
            htmlTable.append("</tr>");
        }

        htmlTable.append("</table>");
        return htmlTable.toString();
    }
}
```

The SQLUtil class (continued)

- The getHtmlTable method in this class accepts a ResultSet object and returns a String object that contains the HTML code for the result set so it can be displayed by a browser.
- The getMetaData method of a ResultSet object returns a ResultSetMetaData object.
- The getColumnCount method of a ResultSetMetaData object returns the number of columns in the result set.
- The getColumnName method of a ResultSetMetaData object returns the name of a column in the result set.

How connection pooling works



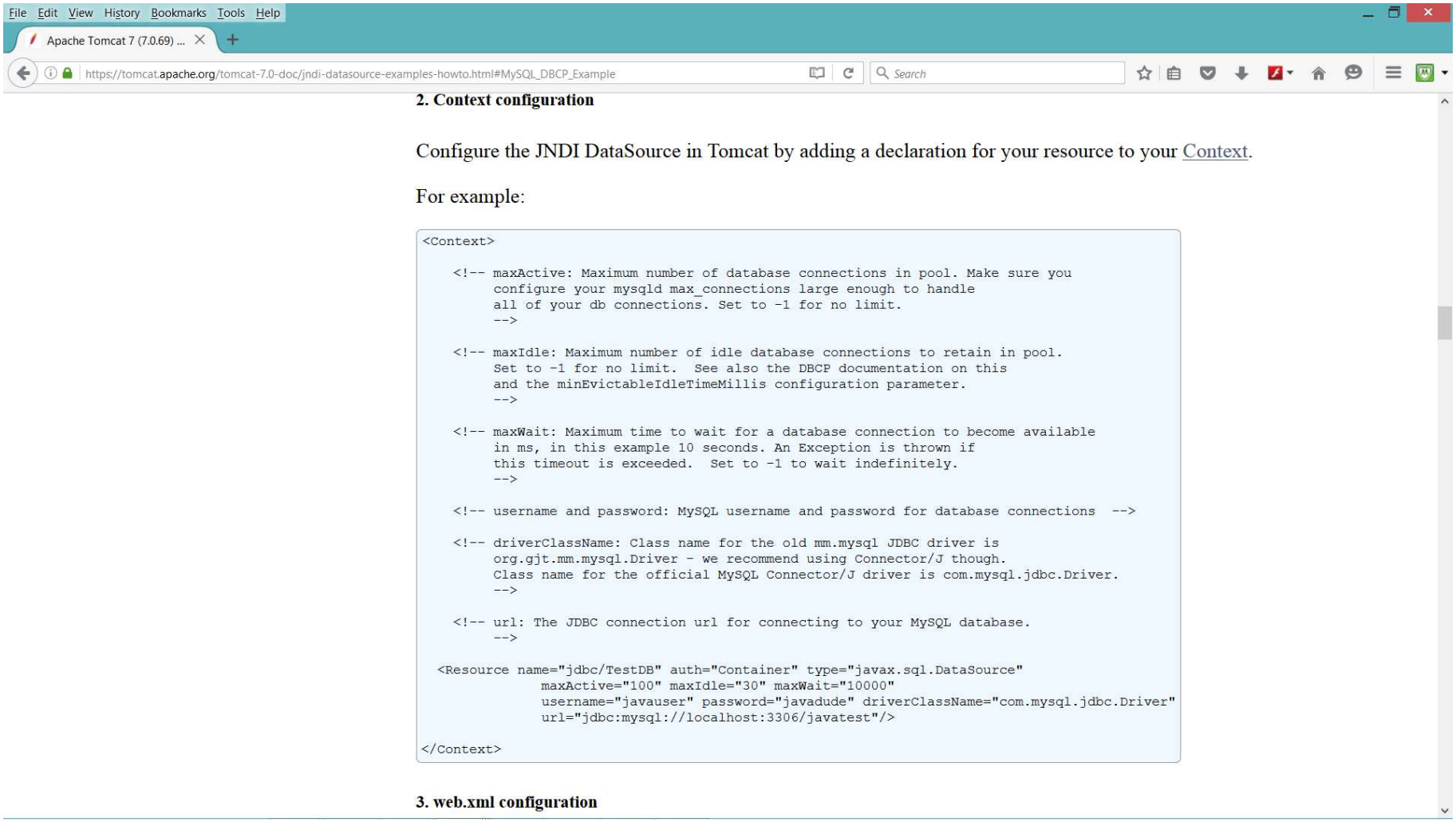
How connection pooling works (continued)

- When *database connection pooling* (*DBCP*) is used, a limited number of connections are opened for a database and are shared by the users who connect to the database. This improves the performance of the database operations.
- When one of the threads of a servlet needs to perform a database operation, the thread gets a Connection object from the ConnectionPool object and uses that Connection object to do the operation.
- When a thread finishes using a Connection object, it returns the object to the pool.
- Before you can use the connection pool, you must make it available to your application.

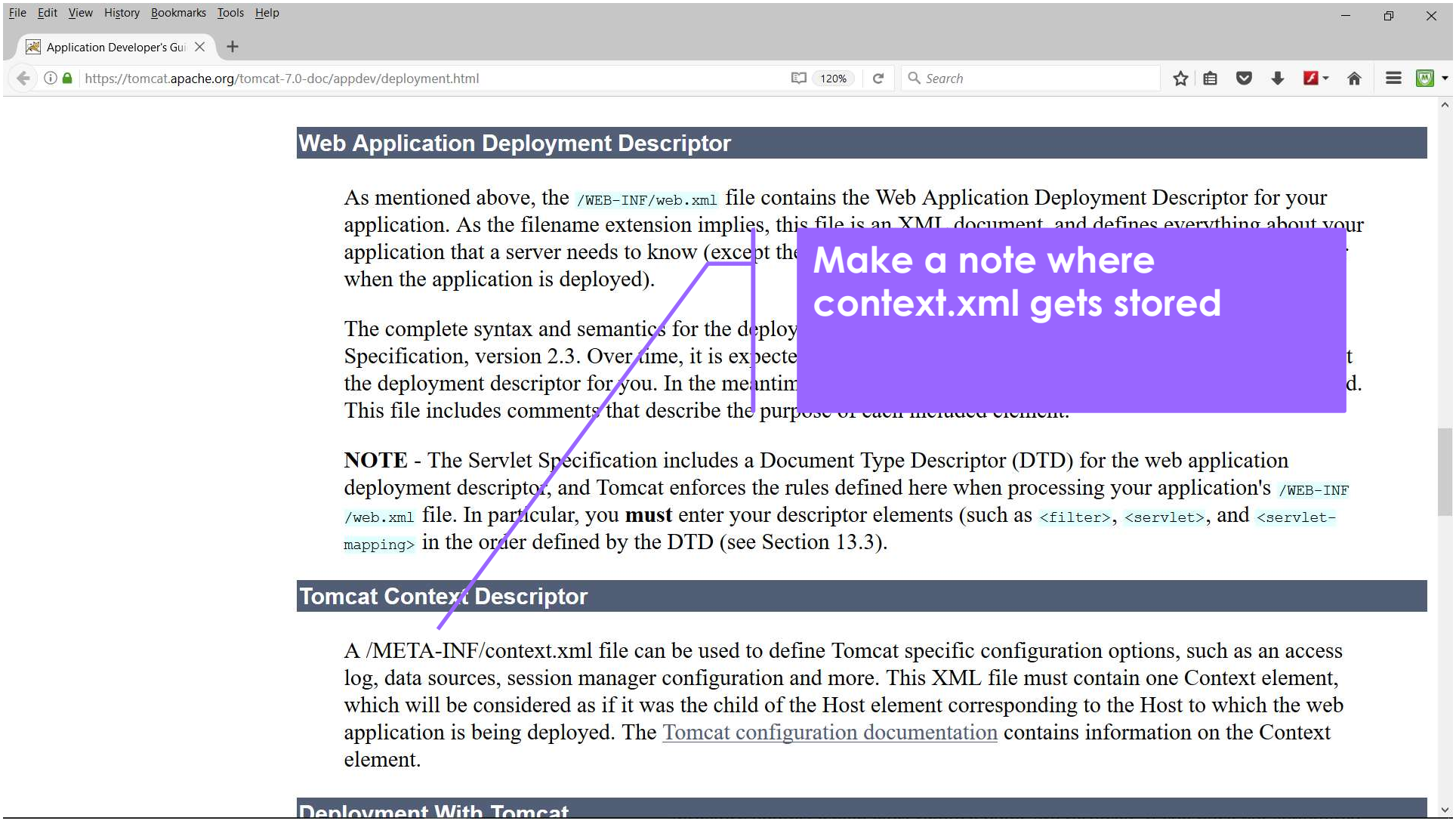
A context.xml file that configures a connection pool

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ch12email">

    <Resource name="jdbc/murach" auth="Container"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/murach?autoReconnect=true"
        username="murach_user" password="sesame"
        maxActive="100" maxIdle="30" maxWait="10000"
        logAbandoned="true" removeAbandoned="true"
        removeAbandonedTimeout="60" type="javax.sql.DataSource" />

</Context>
```

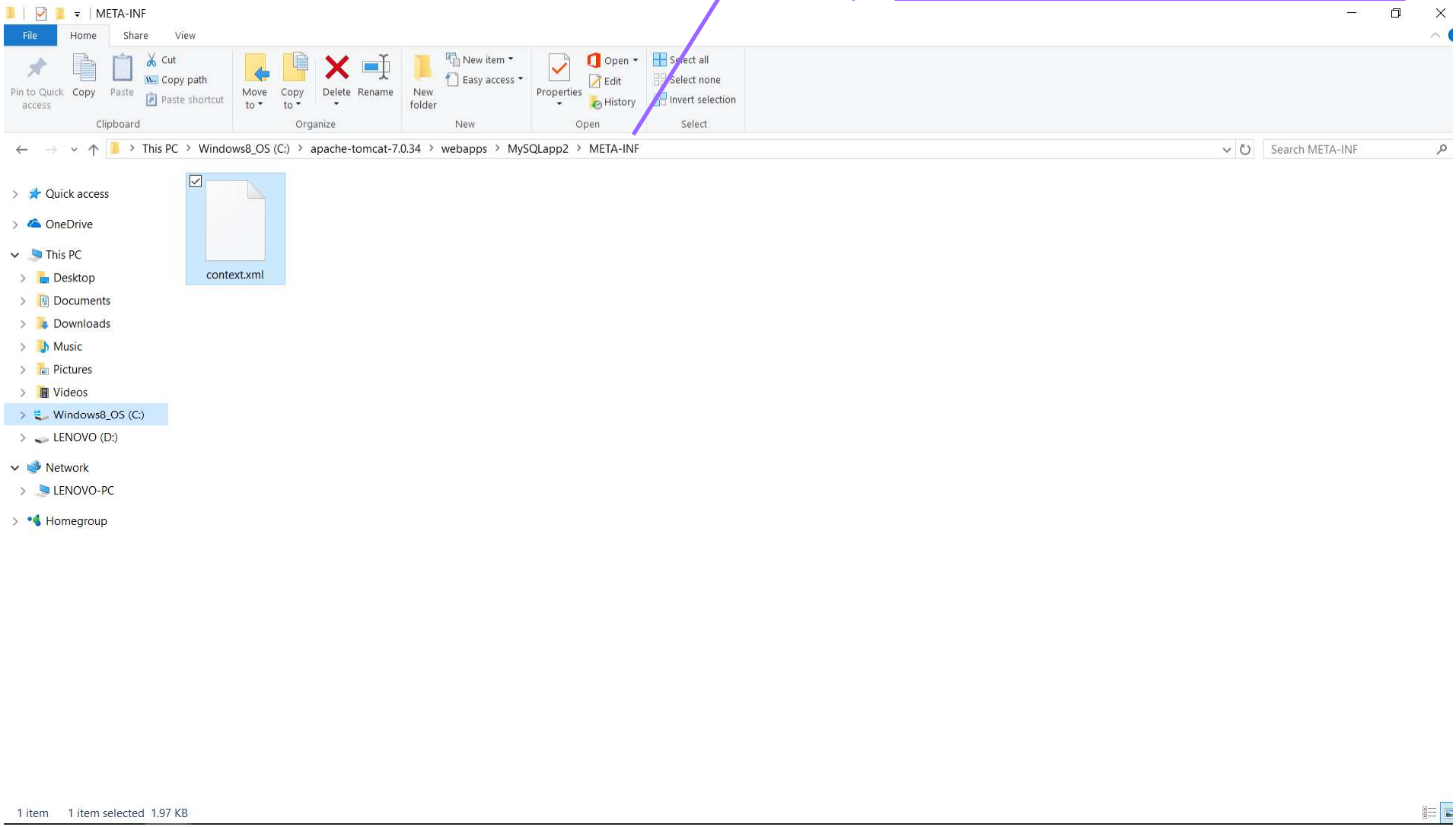
A context.xml file that configures a connection pool



A context.xml file that configures a connection pool



Make a note where context.xml gets stored



A class that defines a connection pool

```
package murach.data;

import java.sql.*;
import javax.sql.DataSource;
import javax.naming.InitialContext;
import javax.naming.NamingException;

public class ConnectionPool {

    private static ConnectionPool pool = null;
    private static DataSource dataSource = null;

    private ConnectionPool() {
        try {
            InitialContext ic = new InitialContext();
            dataSource = (DataSource)
                ic.lookup("java:/comp/env/jdbc/murach");
        } catch (NamingException e) {
            System.out.println(e);
        }
    }
```

A class that defines a connection pool (continued)

```
    public static synchronized ConnectionPool getInstance() {
        if (pool == null) {
            pool = new ConnectionPool();
        }
        return pool;
    }

    public Connection getConnection() {
        try {
            return dataSource.getConnection();
        } catch (SQLException e) {
            System.out.println(e);
            return null;
        }
    }

    public void freeConnection(Connection c) {
        try {
            c.close();
        } catch (SQLException e) {
            System.out.println(e);
        }
    }
}
```

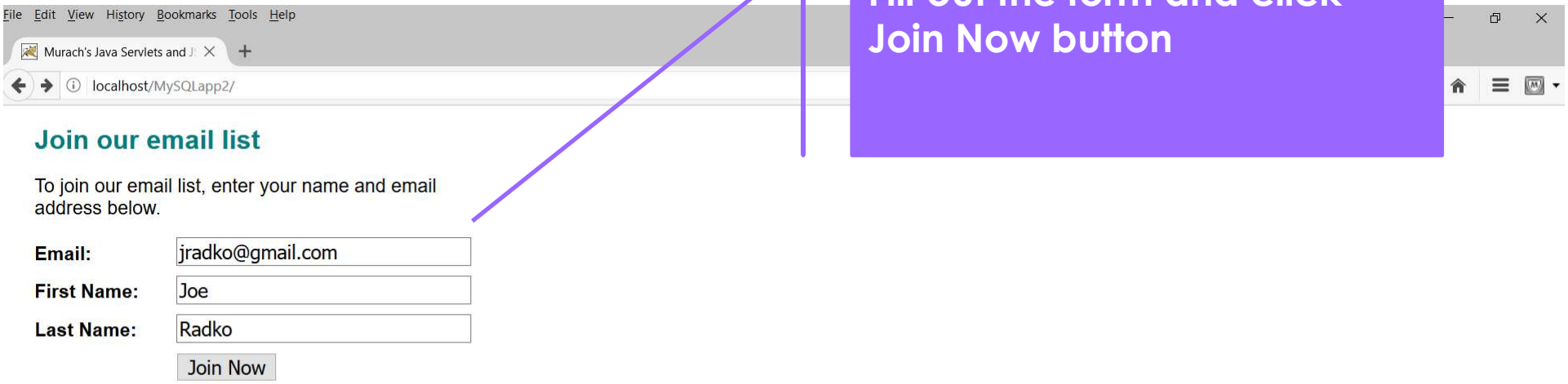
Code that uses the connection pool

```
ConnectionPool pool = ConnectionPool.getInstance();
Connection connection = pool.getConnection();

// code that uses the connection to work with the database

pool.freeConnection(connection);
```

How to implement and use a connection pool

- With Tomcat 6 and later, you can use the context.xml file to configure connection pooling for an application.
- The ConnectionPool class provides the getConnection and freeConnection methods that make it easy for programmers to get connections and to return connections to the connection pool.

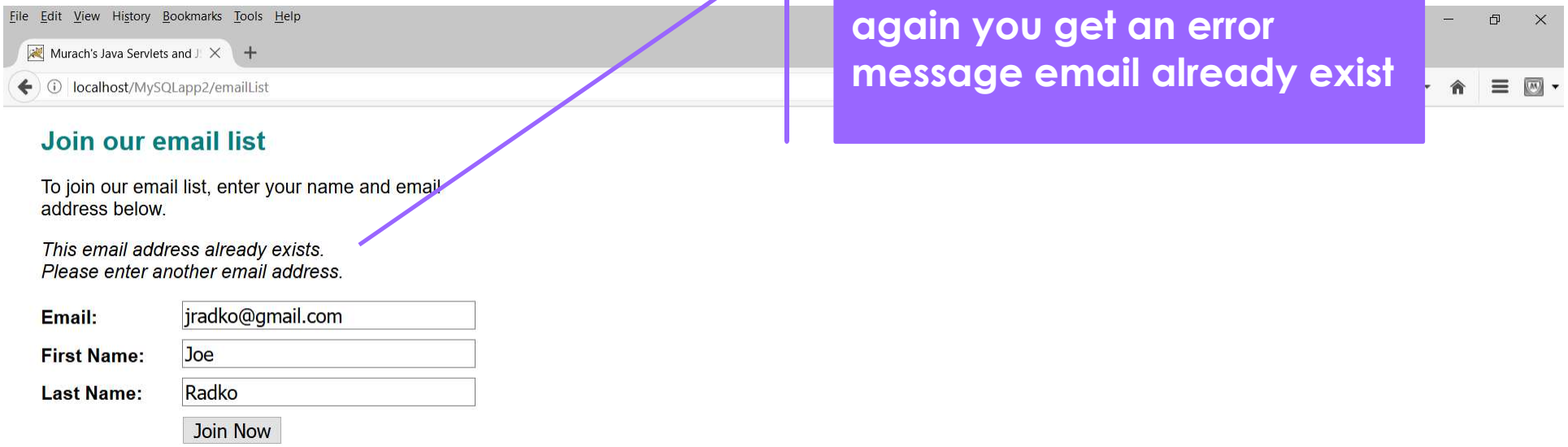
An Email List application that displays an error message if the email already inserted into DB



Fill out the form and click Join Now button

Confirmation Email Entered

Enter the same Record Again you get the Error message email address already exists



Try To enter Same email again you get an error message email already exist

The code for the JSP

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Murach's Java Servlets and JSP</title>
    <link rel="stylesheet" href="styles/main.css" type="text/css"/>
</head>
<body>
    <h1>Join our email list</h1>
    <p>To join our email list, enter your name and
       email address below.</p>
    <p><i>${message}</i></p>
    <form action="emailList" method="post">
        <input type="hidden" name="action" value="add">

        <label class="pad_top">Email:</label>
        <input type="email" name="email" value="${user.email}"
               required><br>
```

The code for the JSP (continued)

```
        <label class="pad_top">First Name:</label>
        <input type="text" name="firstName" value="${user.firstName}"
               required><br>

        <label class="pad_top">Last Name:</label>
        <input type="text" name="lastName" value="${user.lastName}"
               required><br>

        <label> </label>
        <input type="submit" value="Join Now" class="margin_left">
    </form>
</body>
</html>
```

The code for the servlet

```
package murach.email;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import murach.business.User;
import murach.data.UserDB;

public class EmailListServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        String url = "/index.html";

        // get current action
        String action = request.getParameter("action");
        if (action == null) {
            action = "join";  // default action
        }
```

The code for the servlet (continued)

```
// perform action and set URL to appropriate page
 if (action.equals("join")) {
     url = "/index.jsp";    // the "join" page
 }
 else if (action.equals("add")) {
     // get parameters from the request
     String firstName = request.getParameter("firstName");
     String lastName = request.getParameter("lastName");
     String email = request.getParameter("email");

     // store data in User object
     User user = new User(firstName, lastName, email);
```

The code for the servlet (continued)

```
            // validate the parameters
            if (UserDB.emailExists(user.getEmail())) {
                message = "This email address already exists.<br>" +
                          "Please enter another email address.";
                url = "/index.jsp";
            }
            else {
                message = "";
                url = "/thanks.jsp";
                UserDB.insert(user);
            }
            request.setAttribute("user", user);
            request.setAttribute("message", message);
        }
        getServletContext()
                .getRequestDispatcher(url)
                .forward(request, response);
    }
}
```

The UserDB class

```
package murach.data;

import java.sql.*;

import murach.business.User;

public class UserDB {

    public static int insert(User user) {
        ConnectionPool pool = ConnectionPool.getInstance();
        Connection connection = pool.getConnection();
        PreparedStatement ps = null;

        String query
                = "INSERT INTO User (Email, FirstName, LastName) "
                + "VALUES (?, ?, ?)";
```

The UserDB class (continued)

```
        try {
            ps = connection.prepareStatement(query);
            ps.setString(1, user.getEmail());
            ps.setString(2, user.getFirstName());
            ps.setString(3, user.getLastName());
            return ps.executeUpdate();
        } catch (SQLException e) {
            System.out.println(e);
            return 0;
        } finally {
            DBUtil.closePreparedStatement(ps);
            pool.freeConnection(connection);
        }
    }
```

The UserDB class (continued)

```
public static int update(User user) {
    ConnectionPool pool = ConnectionPool.getInstance();
    Connection connection = pool.getConnection();
    PreparedStatement ps = null;

    String query = "UPDATE User SET "
            + "FirstName = ?, "
            + "LastName = ? "
            + "WHERE Email = ?";
    try {
        ps = connection.prepareStatement(query);
        ps.setString(1, user.getFirstName());
        ps.setString(2, user.getLastName());
        ps.setString(3, user.getEmail());

        return ps.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e);
        return 0;
    } finally {
        DBUtil.closePreparedStatement(ps);
        pool.freeConnection(connection);
    }
}
```

The UserDB class (continued)

```
public static int delete(User user) {
    ConnectionPool pool = ConnectionPool.getInstance();
    Connection connection = pool.getConnection();
    PreparedStatement ps = null;

    String query = "DELETE FROM User "
            + "WHERE Email = ?";
    try {
        ps = connection.prepareStatement(query);
        ps.setString(1, user.getEmail());

        return ps.executeUpdate();
    } catch (SQLException e) {
        System.out.println(e);
        return 0;
    } finally {
        DBUtil.closePreparedStatement(ps);
        pool.freeConnection(connection);
    }
}
```

The UserDB class (continued)

```
public static boolean emailExists(String email) {
    ConnectionPool pool = ConnectionPool.getInstance();
    Connection connection = pool.getConnection();
    PreparedStatement ps = null;
    ResultSet rs = null;

    String query = "SELECT Email FROM User "
            + "WHERE Email = ?";
    try {
        ps = connection.prepareStatement(query);
        ps.setString(1, email);
        rs = ps.executeQuery();
        return rs.next();
    } catch (SQLException e) {
        System.out.println(e);
        return false;
    } finally {
        DBUtil.closeResultSet(rs);
        DBUtil.closePreparedStatement(ps);
        pool.freeConnection(connection);
    }
}
```

The UserDB class (continued)

```
public static User selectUser(String email) {
    ConnectionPool pool = ConnectionPool.getInstance();
    Connection connection = pool.getConnection();
    PreparedStatement ps = null;
    ResultSet rs = null;

    String query = "SELECT * FROM User "
            + "WHERE Email = ?";
    try {
        ps = connection.prepareStatement(query);
        ps.setString(1, email);
        rs = ps.executeQuery();
        User user = null;
        if (rs.next()) {
            user = new User();
            user.setFirstName(rs.getString("FirstName"));
            user.setLastName(rs.getString("LastName"));
            user.setEmail(rs.getString("Email"));
        }
        return user;
```

The UserDB class (continued)

```
} catch (SQLException e) {
            System.out.println(e);
            return null;
        } finally {
            DBUtil.closeResultSet(rs);
            DBUtil.closePreparedStatement(ps);
            pool.freeConnection(connection);
        }
    }
}
```

The DBUtil class

```
package murach.data;

import java.sql.*;

public class DBUtil {

    public static void closeStatement(Statement s) {
        try {
            if (s != null) {
                s.close();
            }
        } catch (SQLException e) {
            System.out.println(e);
        }
    }
```

The DBUtil class (continued)

```
    public static void closePreparedStatement(Statement ps) {
        try {
            if (ps != null) {
                ps.close();
            }
        } catch (SQLException e) {
            System.out.println(e);
        }
    }

    public static void closeResultSet(ResultSet rs) {
        try {
            if (rs != null) {
                rs.close();
            }
        } catch (SQLException e) {
            System.out.println(e);
        }
    }
}
```