**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Spring 2022, Lab Manual - 10**

|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course: Software Construction and Development** |
| **Instructor(s):** | **Miss Nida Munawar, Miss Abeeha Sattar** |

**Lab # 10**

**Introduction to MySQL**

MySQL is a widely used, free and open-source relational database management system (RDBMS). It is ideal for both small and large applications.

Let’s go through some basic operations for MySQL. We will later use this table to perform CRUD operations using your application on Eclipse.

**CREATE DATABASE**

The CREATE DATABASE statement is used to create a new SQL database. (You can also use the command CREATE SCHEMA)

CREATE DATABASE `lab10 ` ;

**CREATE TABLE**

The CREATE TABLE statement is used to create a new table in a database. It follows the following format:

CREATE TABLE *table\_name*(  
*column1 datatype*,  
*column2 datatype*,  
*column3 datatype*,  
   ....  
);

Create the following table:

CREATE TABLE `person` (   
 `id` INT NOT NULL,  
`name` VARCHAR(45) NULL,  
`address` VARCHAR(45) NULL,  
PRIMARY KEY (`id`)  
);

**INSERT INTO**

The INSERT INTO statement is used to insert new records in a table. You have to specify both the column names and the values to be inserted:

INSERT INTO *table\_name* (*column1*,*column2*,*column3*, ...)  
VALUES (*value1*,*value2*,*value3*, ...);

**NOTE: you must surround string insides single quotations ('), however, you can enter numbers directly.**

Now insert the following rows into your table:

INSERT INTO `person` (`id`, `name`, `address`) VALUES ('10', 'Sophia', 'Manchester');

INSERT INTO `person` (`id`, `name`, `address`) VALUES ('11', 'William', 'Washington');

INSERT INTO `person` (`id`, `name`, `address`) VALUES ('12', 'Alex', 'Boise');

INSERT INTO `person` (`id`, `name`, `address`) VALUES ('13', 'Amelia', 'London');

**SELECT**

The SELECT statement is used to select data from a database. Syntax:

SELECT \* FROM *table\_name*;

Try viewing your table with the statement:

SELECT \* from `person`;

**WHERE, AND, OR & NOT**

The WHERE clause is used to filter records. It can be combined with AND, OR, and NOT operators.

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
**WHERE** *condition1* **AND** *condition2* **OR** *condition3 ...*;

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
**WHERE** **NOT** *condition*;

**ORDER BY**

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

SELECT *column1*,*column2, ...*  
FROM *table\_name*  
ORDER BY *column1, column2, ...*ASC|DESC;

Try sorting by name on your table.

SELECT \* from `person` order by `name` desc;

**UPDATE & SET**

The UPDATE statement is used to modify the existing records in a table. SET is a keyword that is used along with the UPDATE statement in order to set the new values. It has the following syntax:

UPDATE *table\_name*  
SET *column1*=*value1*,*column2*=*value2*, ...  
WHERE *condition*;

**NOTE: If you omit the WHERE clause, all records in the table will be updated!**

**DELETE**

The DELETE statement is used to delete existing records in a table.

DELETE FROM *table\_name*WHERE *condition*;

**NOTE: If you omit the WHERE clause, all records in the table will be deleted!**

**LIMIT**

The LIMIT clause is used to specify the number of records to return. It is useful on large tables with thousands of records. Returning a large number of records can impact performance.

SELECT *column\_name(s)*  
FROM *table\_name*WHERE *condition*  
LIMIT *number*;

Try it for your own table to return the top two records in the table:

SELECT \* FROM `person`  
LIMIT 2;

**MIN() & MAX()**

The MIN() function returns the smallest value of the selected column. Syntax:

SELECT MIN(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

The MAX() function returns the largest value of the selected column. Syntax:

SELECT MAX(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

**COUNT, AVG & SUM**

The COUNT() function returns the number of rows that matches a specified criterion. Syntax:

SELECT COUNT(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

The AVG() function returns the average value of a numeric column.  Syntax:

SELECT AVG(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

The SUM() function returns the total sum of a numeric column. Syntax:

SELECT SUM(*column\_name*)  
FROM *table\_name*  
WHERE *condition*;

**NOTE: Please explore rest of the things on your own! We have covered the very basics here.**

**You can download MySQL from here:** [**https://dev.mysql.com/downloads/windows/installer/8.0.html**](https://dev.mysql.com/downloads/windows/installer/8.0.html)

The mysql connector jar file would be present in **C:\Program Files (x86)\MySQL\Connector J 8.0** if you installed it with default settings.

**JDBC**

**Java Database Connectivity (JDBC)** is an application programming interface (API) for the Java programming language, which defines how a client may access a database. It is a Java-based data access technology used for Java to database connectivity and it is part of the Java Standard Edition platform and given by Oracle corporation. It provides methods to query and update data in a database and is oriented toward relational databases.

JDBC technology allows us to use the Java programming language to exploit “Write Once, Run Anywhere” capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, we can connect all corporate data even in a heterogeneous environment.

For each database, we need a separate JDBC driver. If we want to work with Oracle database then arrange Oracle JDBC driver, for MySQL database arrange MySQL JDBC driver/connector, and for PostgreSQL arrange PostgreSQL JDBC driver.

JDBC Basic components are:

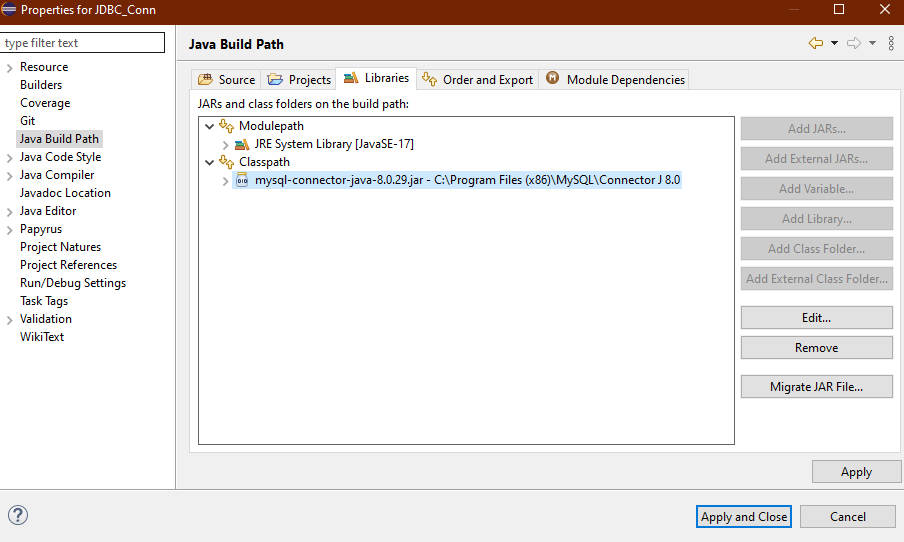
1. **Driver**: It works as a translator which converts Java-specific call to database call and database-specific call to Java call.
2. **Connection**: From Java application to reach database some network connection is required, that network socket is nothing but a connection.
3. **Statement**: It is used to send the SQL queries to the database.
4. **ResultSet**: It holds the result of the SQL query.

JDBC features:

1. It is a standalone API. It is database-independent technologies. JDBC program written for one database can be used for any database.
2. It supports WORA (write once and run anywhere). Most JDBC drivers are written in Java only so it can be used on any platform.
3. Using JDBC we can easily perform CRUD operations. C=> Create, R=> Retrieve, U=> Update, D=> Delete.
4. JDBC technology is open-source so most of the vendors developed their product based on JDBC.

**Using JDBC with Eclipse:**

1. Open Eclipse => chose your workspace => File => New => Other => Search for “Java Project” => Next => Enter project name (Example: MySQLDBConnection) => Finish
2. Now, open the properties of the project. For this you can use shortcut key:- select project then Alt + Enter (Or) Right click on the Project => Properties
3. In the Java Build path, in the Libraries Section => Classpath => Click on “Add External Jars” => Select *mysql-connector-java-<version>.jar* => Apply => Apply and close



1. In the Project Explorer, in that Java Project, in the “Referenced Libraries” section, You can see that *mysql-connector-java-<version>.jar* is available. Eclipse IDE will use it while compilation and execution process.
2. In the Java project => Right-click on “src” => New => Class => ClassName => Finish. Use a reasonable class name, like: *ConnectionTest, Conn\_Class, etc.*
3. Now we need some details of the JDBC driver to develop our program.
   1. JDBC Driver Class Name:: ***com.mysql.cj.jdbc.Driver***
   2. URL for local MySQL:: **jdbc:mysql:///<logical-database-name>**

**NOTE**: You can view the name of the database by using **“show databases”** command

* 1. Username:***<username-of-MySQL-database>***
  2. Password: ***<password-of-MySQL-database>***

1. Write the following program to test your connection:

import java.sql.Connection;

import java.sql.DriverManager;

public class Conn\_class {

public static void main(String[] args) throws Exception{

// see note at bottom

Class.forName("com.mysql.cj.jdbc.Driver");

// variables for connection

final String url = "jdbc:mysql:///test";

final String user = "root";

final String password = "yourpassword";

// establish the connection

Connection con = DriverManager.getConnection(url, user, password);

// display status message

if (con == null) {

System.out.println("JDBC connection is not established");

return;

} else {

System.out.println("Congratulations, JDBC connection is established successfully.\n");

}

//Remember to close the JDBC connection

con.close();

}

}

**NOTE**: When you are using JDBC outside of an application server, the **DriverManager** class manages the establishment of connections.

Specify to the **DriverManager** which JDBC drivers to try to make Connections with. The easiest way to do this is to use  **Class.forName()**  on the class that implements the **java.sql.Driver** interface. With **MySQL Connector/J**, the name of this class is **com.mysql.jdbc.Driver**. With this method, you could use an external configuration file to supply the driver class name and driver parameters to use when connecting to a database.

1. Now let make a class (for example: SelectTest) in which we can try fetching the information from the database.

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class SelectTest {

public static void main(String[] args) throws SQLException {

// variables

final String url = "jdbc:mysql:///test";

final String user = "root";

final String password = "yourpassword";

// establish the connection

Connection con = DriverManager.getConnection(url, user, password);

// create JDBC statement object

Statement st = con.createStatement();

// prepare SQL query

String query = "SELECT ID, NAME, ADDRESS FROM PERSON";

// send and execute SQL query in Database software

ResultSet rs = st.executeQuery(query);

// process the ResultSet object

boolean flag = false;

while (rs.next()) {

flag = true;

System.out.println(rs.getInt(1) + " " + rs.getString(2) +

" " + rs.getString(3));

}

if (flag == true) {

System.out.println("\nRecords retrieved and displayed");

} else {

System.out.println("Record not found");

}

// close JDBC objects

rs.close();

st.close();

con.close();

}

}

1. Now let’s see how we can insert a record. Create a new class (for example: InsertTest)

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

import java.util.Scanner;

public class InsertTest {

// values specific to each database

private static final String URL = "jdbc:mysql:///test";

private static final String USERNAME = "root";

private static final String PASSWORD = "YourPassword";

public static void main(String[] args) {

// declare variables

Scanner scan = null;

int sno = 0;

String sname = null, sadd = null;

float avg = 0.0f;

String query = null;

Connection con = null;

Statement st = null;

int count = 0;

try {

// create Scanner class object

scan = new Scanner(System.in);

// read input

if(scan != null) {

System.out.print("Enter student number: ");

sno = scan.nextInt();

System.out.print("Enter student name: ");

sname = scan.next();

System.out.print("Enter student address: ");

sadd = scan.next();

}

// prepare SQL query

query = "INSERT INTO PERSON VALUES ("

+ sno + ", " + "\'" + sname +"\' ,"

+"\'"+ sadd + "\') ";

// establish the connection

con = DriverManager.getConnection(

URL, USERNAME, PASSWORD);

// create JDBC statement object

if(con != null) {

st = con.createStatement();

}

// execute the SQL query

if(st != null) {

count = st.executeUpdate(query);

}

// display result

System.out.println(count + " record inserted.");

} catch(SQLException se) {

se.printStackTrace();

} catch(Exception e) {

e.printStackTrace();

} // try-catch block end

finally {

// close JDBC objects

try {

if(st != null) st.close();

} catch(SQLException se) {

se.printStackTrace();

}

try {

if(con != null) con.close();

} catch(SQLException se) {

se.printStackTrace();

}

try {

if(scan != null) scan.close();

} catch(Exception e) {

e.printStackTrace();

}

}//finally block end

}

}

**TYPES OF STATEMENTS**

**Statement:**

It is the object of a JDBC driver software supplied Java class that implements java.sql.Statement(I). It is also called as Simple Statement because it is used with simple SQL statements without parameters. We have seen how to use these in our previous examples.

**PreparedStatement object**:

It is the object of a JDBC driver software supplied Java class that implements java.sql.PreparedStatement(I). It is extended from java.sql.Statement(I), and it is used for precompiling SQL statements that might contain input parameters.

**CallableStatment** **object**:

It is the object of a JDBC driver software supplied Java class that implements java.sql.CallableStatement(I). It is also extended from java.sql.Statement(I), and it is used to execute stored procedures that may contain both input and output parameters.

**How to Write a Program that uses Prepared Statements:**

1. Prepare SQL query with positional parameter/place holder/place resolver (?)

String query = "INSERT INTO PERSON VALUES(?,?,?)";

1. Send SQL query to Database software. Make that SQL query as pre-compiled SQL query in database software and get **PreparedStatement** object representing the SQL query.

PreparedStatement ps = con.prepareStatement(query);

1. Set input values to pre-compiled SQL query parameters(?) using setXxx(-) methods.

ps.setInt(1,1001); // ID

ps.setString(2, “Abeeha Sattar”); // product name

ps.setString(3, “My Address”); // price of product

1. Execute the pre-compiled SQL query in database software

int result = ps.executeUpdate();

1. Process the result:

if(result == 0)

System.out.println("Not inserted");

else

System.out.println("record inserted");

1. Execute the above pre-compiled SQL query for multiple times either with same or difference values (repeat 3 to 5 steps)
2. Close the JDBC objects.

ps.close();

con.close();

**Example for Insert Prepared Statement:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Scanner;

public class PreparedInsert {

// SQL query

private static final String INSERT\_PERSON\_QUERY = "INSERT INTO PERSON VALUES(?,?,?)";

private static final String URL = "jdbc:mysql:///test";

private static final String USERNAME = "root";

private static final String PASSWORD = "YourPassword";

public static void main(String[] args) {

// declare variables

Scanner scan = null;

int noRec = 0, sid = 0;

String name = null, address = null;

Connection con = null;

PreparedStatement ps = null;

int result = 0;

try {

// read input

scan = new Scanner(System.in);

if(scan != null) {

System.out.print("Enter the number of records that you want to insert: ");

noRec = scan.nextInt();

}

// establish the connection

con = DriverManager.getConnection(URL, USERNAME, PASSWORD);

// compile SQL query and

// store it in PreparedStatemet object

if(con != null)

ps = con.prepareStatement(INSERT\_PERSON\_QUERY);

// read multiple set of inputs from end-user

// set input values and execute the query

if(scan != null && ps != null) {

for(int i=0; i < noRec; i++) {

// read inputs

System.out.println("\nEnter Record-"+ (i+1) +

" details, ");

System.out.print("ID: ");

sid = scan.nextInt();

System.out.print("Name: ");

name = scan.next();

System.out.print("Address: ");

address = scan.next();

// set input values to query parameters

ps.setInt(1, sid);

ps.setString(2, name);

ps.setString(3, address);

// execute the query

result = ps.executeUpdate();

}

}

// process the result

if(result == 0)

System.out.println("\nRecords not inserted");

else

System.out.println("\nRecords inserted"+

" successfully");

} catch(SQLException se) {

se.printStackTrace();

} catch(Exception e) {

e.printStackTrace();

} // end of try-catch block

}

}

**Example for Update Prepared Statement:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.util.Scanner;

public class PreparedUpdate {

// SQL query

private static final String UPDATE\_PERSON\_QUERY =

"UPDATE PERSON SET NAME = ?, ADDRESS = ?"

+ "WHERE ID = ? ";

private static final String URL = "jdbc:mysql:///test";

private static final String USERNAME = "root";

private static final String PASSWORD = "YourPassword";

public static void main(String[] args) {

// declare variables

Scanner scan = null;

int id = 0;

String name = null;

String address = null;

Connection con = null;

PreparedStatement ps = null;

int result = 0;

try {

//read input

scan = new Scanner(System.in);

if(scan != null) {

System.out.println("Enter the existing person ID to update: ");

id = scan.nextInt();

System.out.println("Enter new details,");

System.out.print("Name of the person: ");

name = scan.next();

System.out.print("Address: ");

address = scan.next();

}

// establish the connection

con = DriverManager.getConnection(URL, USERNAME, PASSWORD);

// compile SQL query and

// store it in PreparedStatemet object

if(con != null)

ps = con.prepareStatement(UPDATE\_PERSON\_QUERY);

// set input values and execute query

if(ps != null) {

// set input values to query parameters

ps.setString(1, name);

ps.setString(2, address);

ps.setInt(3, id);

// execute the query

result = ps.executeUpdate();

}

// process the result

if(result == 0)

System.out.println("Records not updated");

else

System.out.println("Records updated successfully");

} catch(SQLException se) {

se.printStackTrace();

} catch(Exception e) {

e.printStackTrace();

} // end of try-catch block

finally {

// close JDBC objects

try {

if(ps != null) ps.close();

} catch(SQLException se) {

se.printStackTrace();

}

try {

if(con != null) con.close();

} catch(SQLException se) {

se.printStackTrace();

}

try {

if(scan != null) scan.close();

} catch(Exception e) {

e.printStackTrace();

}

} // end of finally block

}

}

**Example for Select and Display Record using Prepared Statement**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Scanner;

public class PreparedSelect {

// SQL query

private static final String *SELECT\_PERSON\_QUERY* =

"SELECT NAME, ADDRESS FROM PERSON"+

" WHERE ID = ?";

private static final String *URL* = "jdbc:mysql:///test";

private static final String *USERNAME* = "root";

private static final String *PASSWORD* = "YourPassword";

public static void main(String[] args) {

// declare variables

Scanner scan = null;

int id = 0;

Connection con = null;

PreparedStatement ps = null;

ResultSet rs = null;

boolean flag = false;

try {

// read input

scan = new Scanner(System.*in*);

if(scan != null) {

System.*out*.print("Enter person ID: ");

id = scan.nextInt();

}

// establish the connection

con = DriverManager.*getConnection*(*URL*, *USERNAME*, *PASSWORD*);

// compile SQL query and

// store it in PreparedStatemet object

if(con != null)

ps = con.prepareStatement(*SELECT\_PERSON\_QUERY*);

// set input value to query parameter

if(ps != null)

ps.setInt(1, id);

// execute the query

if(ps != null)

rs = ps.executeQuery();

// process the result

if(rs != null) {

while(rs.next()) {

flag = true;

System.*out*.println(""

+ rs.getString("NAME") +" "

+ rs.getString("ADDRESS"));

}

}

if(flag)

System.*out*.println("Records fetched & displayed");

else

System.*out*.println("Records not found");

} catch(SQLException se) {

se.printStackTrace();

} catch(Exception e) {

e.printStackTrace();

} // end of try-catch block

finally {

// close JDBC objects

try {

if(ps != null) ps.close();

} catch(SQLException se) {

se.printStackTrace();

}

try {

if(con != null) con.close();

} catch(SQLException se) {

se.printStackTrace();

}

try {

if(scan != null) scan.close();

} catch(Exception e) {

e.printStackTrace();

}

} //end if finally block

}

}