10.8: Introduction to Database Connection

IT 1406 – Introduction to Programming

Level I - Semester 1

Introduction to Database Connection

What is Database ?

Database is a structured collection of data stored in a computer. The term structured implies that each record in the database is stored in a certain format.

A ***Relational Database*** organizes data in *tables*. A *table* has *rows* (or *records*) and *columns* (or *fields*). Tables are *related* based on common columns to eliminate *data redundancy* and ensure *data integrity*.

There are different Real world applications of Databases as follow-

- Library catalogues
- Airline Bookings
- Telephone Directories
- Train Timetable

Database Management System

Database management system is a software which is used to manage the database which facilitates the processes of *defining, constructing, manipulating, maintaining and sharing* databases among various users and applications.

It provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more along with protection and security to the database.

Why we need a DBMS??

- Enhance Data independence and efficient access.
- Data integrity and security.
- Uniform administration of data .
- Provide concurrent access to database and recovery from crashes.
- Control Data redundancy.
- Avoid Data inconsistency.

Vendors of Database Management System (DBMS) Software-

1. Commercial Database

- Oracle
- IBM DB2
- Microsoft SQL Server
- Microsoft Access
- Splunk

2. Open Source Database

- MySQL
- PostgreSQL
- MongoDB
- Elasticsearch
- Redis

Structure Query Language (SQL)

SQL is a ***domain-specific language*** *(**NOT Programming language**)* designed for interacting with the relational databases.

Types of SQL Commands			
DDL	**DML**	**DCL**	**TCL**
CREATE	SELECT	GRANT	COMMIT
ALTER	INSERT	REVOKE	ROLLBACK
DROP	UPDATE		SAVEPOINT
TRUNCATE	DELETE		
RENAME	MERGE		

Examples of some SQL commands

1. CREATE

- This statement is used to create a Database called **Employee**.

```
CREATE DATABASE Employee;
```

- This statement is used to create a Table called **Employee_Info**.

```
CREATE TABLE Employee_Info
(
EmployeeID int,
EmployeeName varchar(255),
EmergencyContactName varchar(255),
PhoneNumber int,
City varchar(255),
);
```

EmployeeID	EmployeeName	EmergencyContactName	PhoneNumber	City

2. INSERT INTO

- This statement is used to insert new records into the table.

```
INSERT INTO Employee_Info(EmployeeID, EmployeeName, EmergencyContactName, PhoneNumber, City)
VALUES (‘01', 'Sanjana','Jagath', ‘0751111111’, ‘Colombo’);
```

- Can insert values without mentioning column names also as follows,

```
INSERT INTO Employee_Info
VALUES (‘02', 'Preeti’,’Rahul', ‘0761511522’, ‘Kandy’);
VALUES (‘03', ‘Sarath’,’Samantha', ‘0712347589’, ‘Negombo’);
VALUES (‘04', ‘Nimali’,’Ashan', ‘0751122221’, ‘Colombo’);
```

EmployeeID	EmployeeName	EmergencyContactName	PhoneNumber	City
01	Sanjana	Jagath	0751111111	Colombo
02	Preeti	Rahul	0761511522	Kandy
03	Sarath	Samantha	0712347589	Negombo
04	Nimali	Ashan	0751122221	Colombo

3. UPDATE

- This statement is used to modify the records already present in the table.

```
UPDATE Employee_Info
SET EmployeeName = ‘Ahana’, City= ‘Ambalangoda’
WHERE EmployeeID = ’01’;-
```

EmployeeID	EmployeeName	EmergencyContactName	PhoneNumber	City
01	Ahana	Jagath	0751111111	Ambalangoda
02	Preeti	Rahul	0761511522	Kandy
03	Sarath	Samantha	0712347589	Negombo
04	Nimali	Ashan	0751122221	Colombo

4. DELETE

- This statement is used to delete the existing records in a table.

```
DELETE FROM Employee_Info
WHERE EmployeeName='Preeti';
```

EmployeeID	EmployeeName	EmergencyContactName	PhoneNumber	City
01	Ahana	Jagath	0751111111	Ambalangoda
03	Sarath	Samantha	0712347589	Negombo
04	Nimali	Ashan	0751122221	Colombo

5. SELECT

- This statement is used to select data from a database and the data returned is stored in a result table, called the **result-set**.
- The following command select all the columns in the table.

```
SELECT * FROM Employee_Info;
```

EmployeeID	EmployeeName	EmergencyContactName	PhoneNumber	City
01	Ahana	Jagath	0751111111	Ambalangoda
03	Sarath	Samantha	0712347589	Negombo
04	Nimali	Ashan	0751122221	Colombo

- To select required columns of the table, execute the below command.

```
SELECT EmployeeID, EmployeeName
FROM Employee_Info;
```

EmployeeID	EmployeeName
01	Ahana
03	Sarath
04	Nimali

- To select required columns of the table based on a condition, execute the below command.

```
SELECT * FROM Employee_Info;
 WHERE City='Colombo';
```

EmployeeID	EmployeeName	EmergencyContactName	PhoneNumber	City
04	Nimali	Ashan	0751122221	Colombo

6. ALTER ADD/ DROP

- This command is used to delete, modify or add constraints or columns in an existing table.
- To ADD a new column to the existing **Employee_Info** table, execute the following command.

```
ALTER TABLE Employee_Info
ADD BloodGroup varchar(255);
```

EmployeeID	EmployeeName	EmergencyContact Name	PhoneNumber	City	BloodGroup
01	Ahana	Jagath	0751111111	Ambalangoda	
03	Sarath	Samantha	0712347589	Negombo	
04	Nimali	Ashan	0751122221	Colombo	

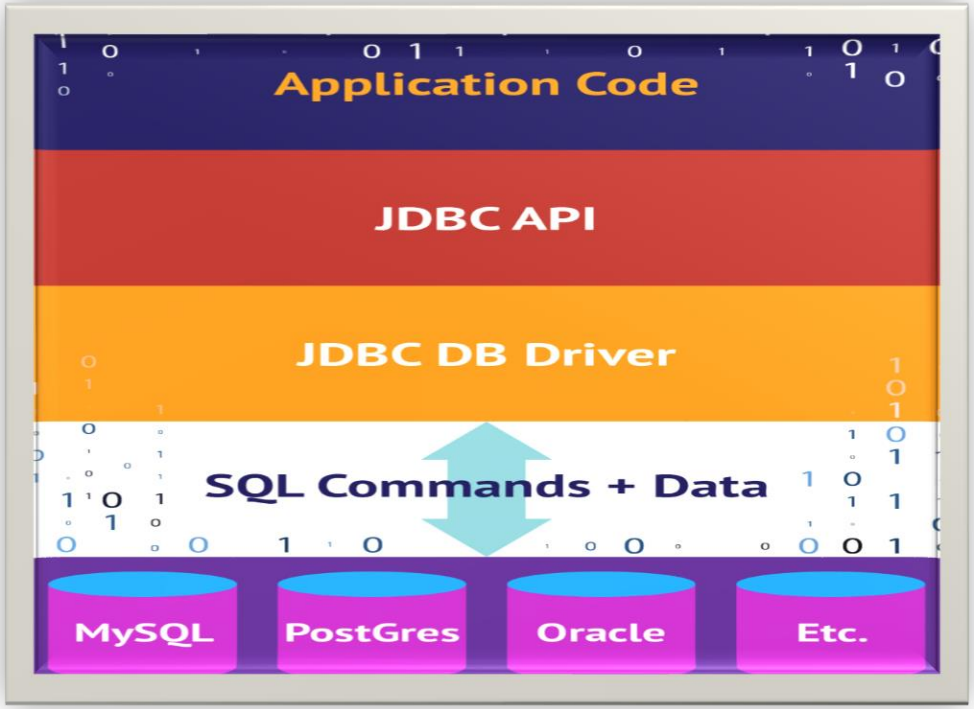
- To remove a column to the existing **Employee_Info** table, execute the following command.

```
ALTER TABLE Employee_Info
DROP COLUMN BloodGroup ;
```

EmployeeID	EmployeeName	EmergencyContact Name	PhoneNumber	City
01	Ahana	Jagath	0751111111	Ambalangoda
03	Sarath	Samantha	0712347589	Negombo
04	Nimali	Ashan	0751122221	Colombo

JAVA Database Connectivity

JDBC (Java Database Connectivity) is the Java API that manages connecting to a database, issuing queries and commands, and handling result sets obtained from the database.



The above image is Architectural overview of JDBC in the Java persistence layer.

Lets see how it works…

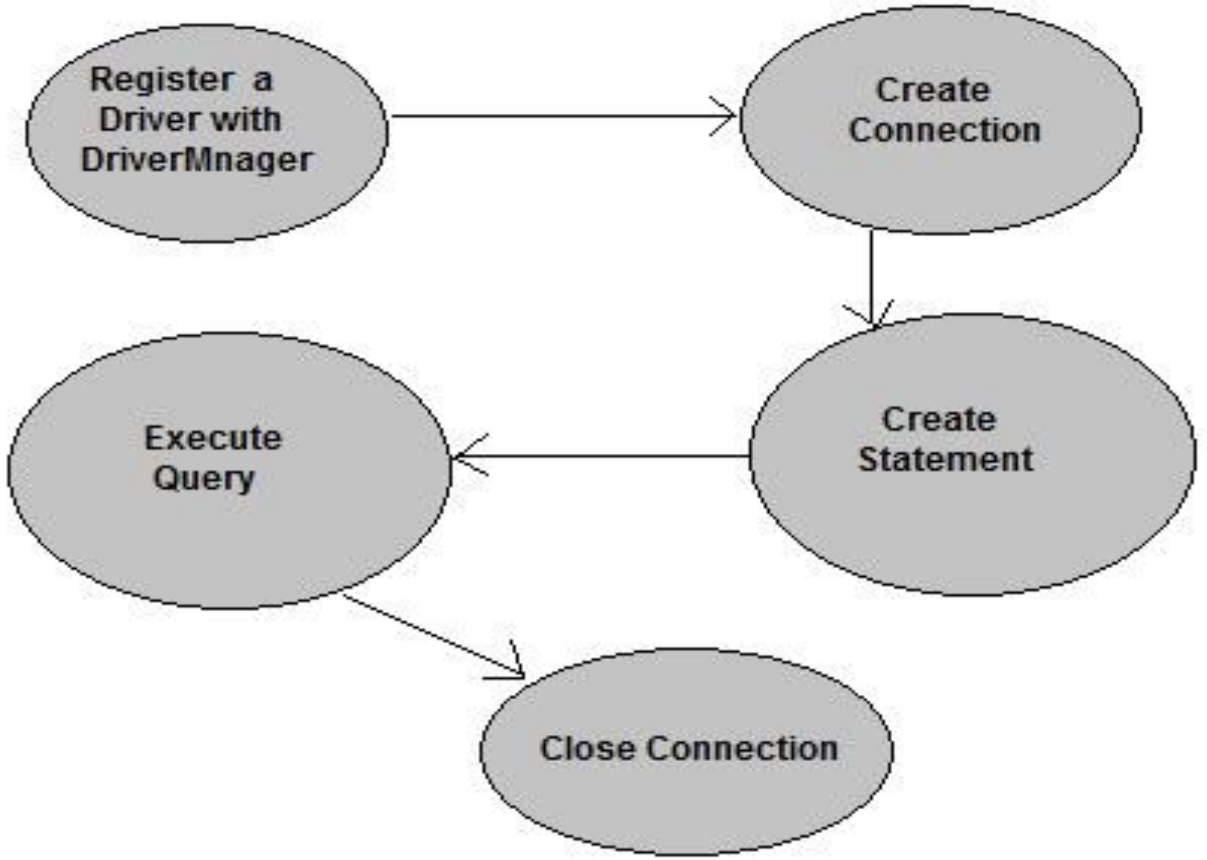
- JDBC developed as an alternative to the C-based ODBC (Open Database Connectivity) API, but JDBC offers a programming-level interface that handles the mechanics of Java applications communicating with a database or RDBMS.

- The JDBC interface consists of two layers:

    1. **JDBC API** supports communication between the Java application and the JDBC manager.
    2. **JDBC driver** supports communication between the JDBC manager and the database driver.

- JDBC is the common API that your application code interacts with. Beneath that is the JDBC-compliant driver for the database you are using.

Connecting JAVA application with any Database using JDBC



1. Register the Driver

In this step, the driver is registered and it causes the JVM to load the favorable driver implementation into the memory to make it able to fulfill the user's JDBC requests.

`Class.forName()` is used to dynamically load the driver's class file explicitly into the memory .

Syntax- `public static void forName(String className) throws ClassNotFoundException`

2. Create the connection object

Establish a connection with the database via the connection object by the help of the `DriverManager.getConnection()` method.

The most widely used form of getConnection() generally needs the user to pass a database ***URL***, a ***username***, and a ***password***.

The details of ***URL***, ***username***, and ***password*** as follow,

- ***URL*** : Your Database path
- ***Username*** : Database Login user name
- ***password*** : Database Login password

Syntax :

```
public static Connection getConnection(String URL, String Username , String password)  throws SQLException ;
```

3. Create the Statement object

The `createStatement()` method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax :

```
public Statement createStatement()throws SQLException;
```

4. Execute SQL Statement

`executeQuery()` method of **Statement** interface is used to execute SQL statements. This method returns the object of `ResultSet` that can be used to get all the records of a table.

Syntax :

```
public ResultSet executeQuery(String sql)throws SQLException;
```

5. Closing the connection

After executing SQL statement you need to close the connection and release the session. The `close()` method of **Connection** interface is used to close the connection.

Syntax :

```
public void close()throws SQLException ;
```

JDBC Driver Download for any Database

In order for Java applications working with a database engine via *Java Database Connectivity (JDBC)*, an appropriate JDBC driver library is required to be available in the application's class path.

Database	JDBC Driver Provider	JAR file name	Download
MySQL	*Oracle Corporation*	`mysql-connector-java-VERSION.jar`	Download JDBC Driver for MySQL
SQL Server	*Microsoft Corporation*	`sqljdbc41.jar, sqljdbc42.jar`	Download JDBC Driver for SQL Server
Oracle	*Oracle Corporation*	`ojdbc6.jar, ojdbc7.jar, ojdbc8.jar`	Download JDBC Driver for Oracle (login required)
PostgreSQL	*The PostgreSQL Global Development Group*	`postgresql-VERSION.jar`	Download JDBC Driver for PostgreSQL
Apache Derby	*Apache Software Foundation*	`derby.jar, derbyclient.jar`	Download JDBC Driver for Apache Derby
SQLite	*Xerial.org*	`sqlite-jdbc-VERSION.jar`	Download JDBC Driver for SQLite

HINT:

- Some drivers come as JAR files (Oracle, PostgreSQL), so you can add the JAR files directly to your application's class path.

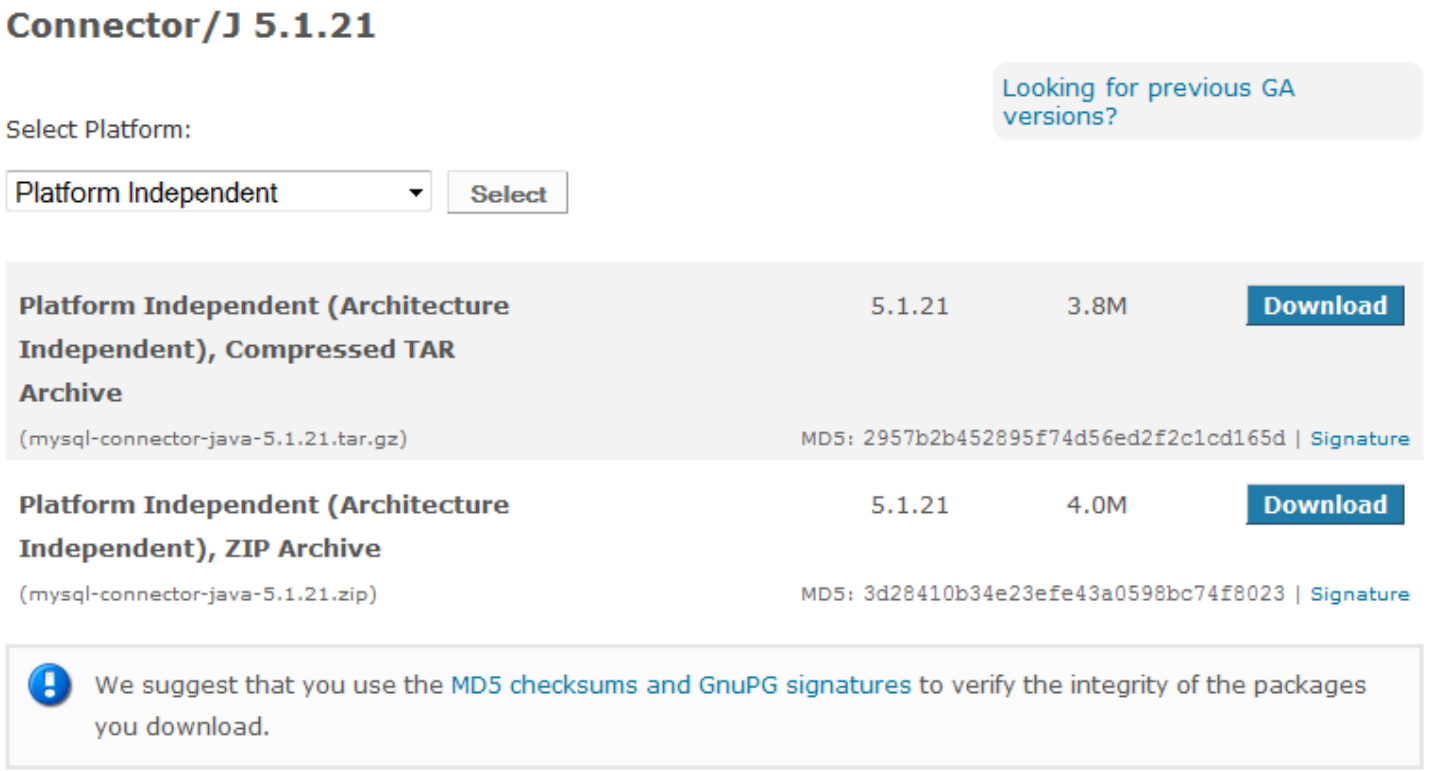
- Some drivers come as zipped bundles (MySQL, SQL Server), so you have to extract the bundles and copy the appropriate JAR file (as specified in the above table) to your application's class path.

- There is a distribution of Apache Derby comes with JDK 7 called **JavaDB** So if you are using JDK 7, you can use the jar files directly from *JDK_HOME\db\lib* directory without downloading Apache Derby. However, Java DB is removed from JDK since Java 8.

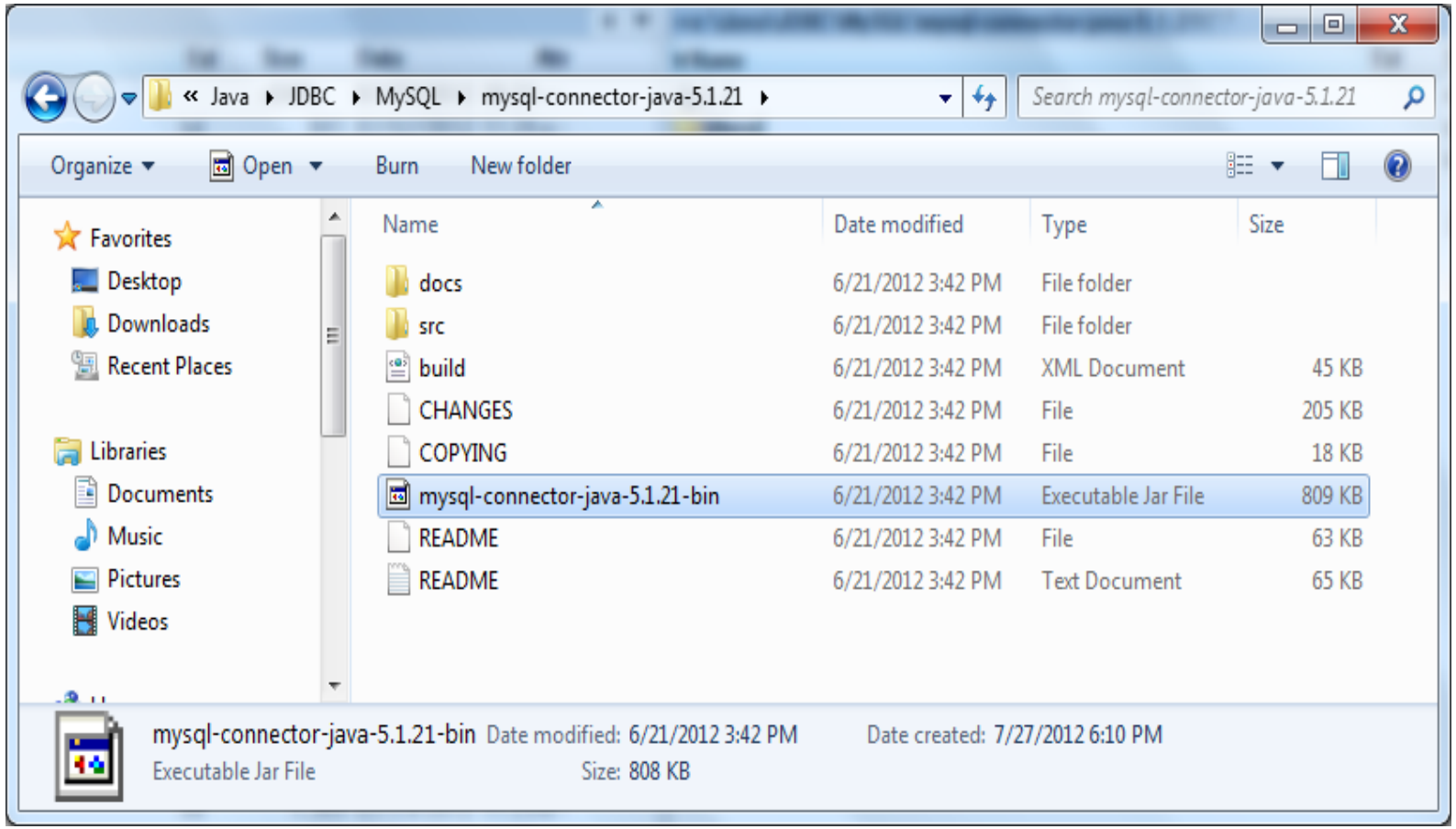
Java connect to MySQL database with JDBC

First, in order to have Java program working with MySQL, you need a JDBC Driver for MySQL.

From http://dev.mysql.com/downloads/connector/j/ you can download the latest version of Connector/J the JDBC Driver for MySQL. The latest version 8.0 supports JDBC 4.2 and JDK 8 or higher.

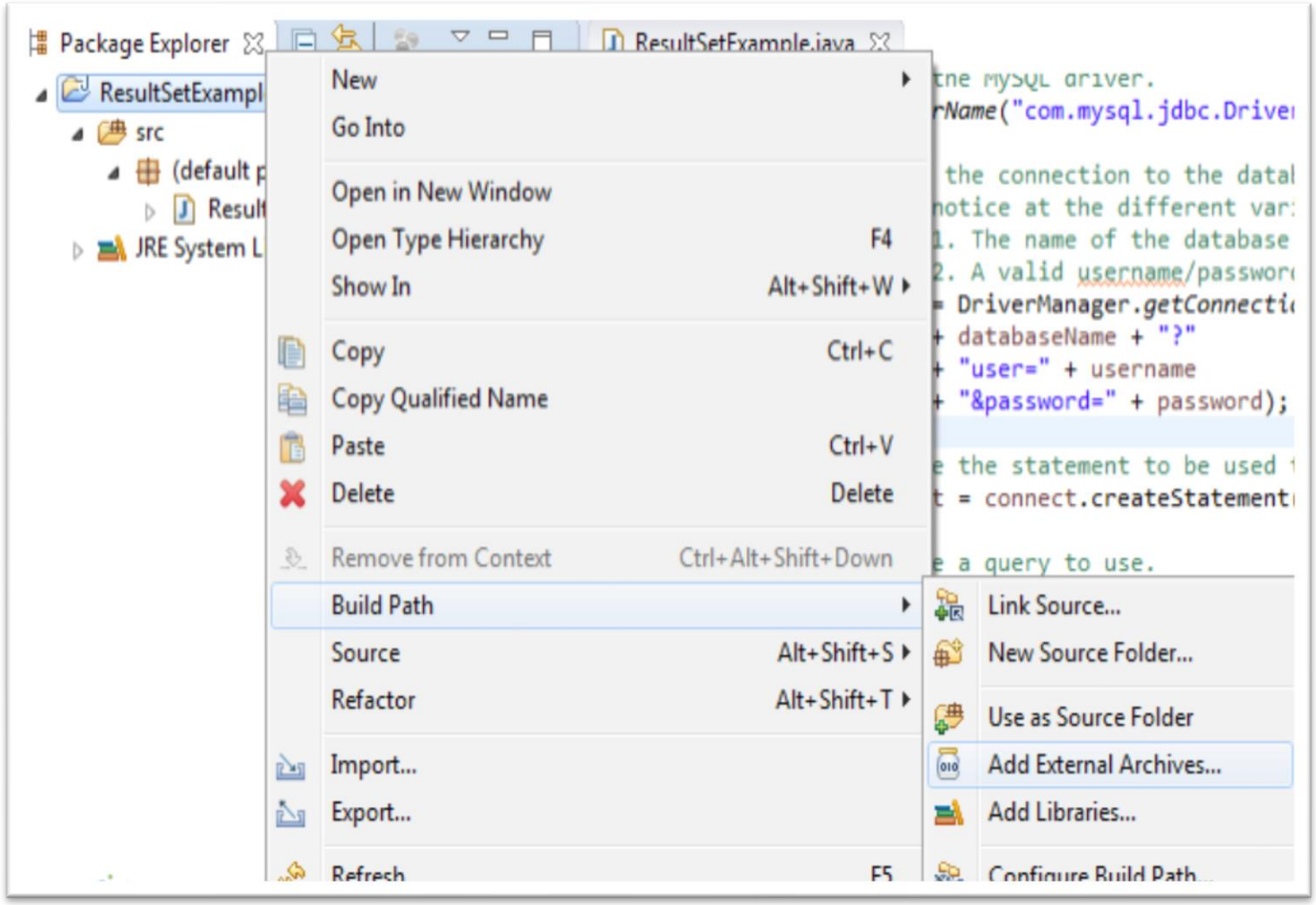


Select the **Platform Independent** option, and download the zip file and extract it into your desired location in the PC



The distribution includes a binary JAR file, source code, documentation and license files. But only one file we need is the JAR file mysql-connector-java-VERSION.jar. Copy this file into your project and make it available in your program's class path.

Copy this file into your project and make it available in your program's class path. You can add the connector to the build path, by **right-clicking on the project -> Build Path -> Add External Archives**, as shown in the image below:



NOTE- No need to load MySQL driver class explicitly

- The Connector/J version 8.0 library comes with a JDBC driver class: com.mysql.cj.jdbc.Driver. Before Java 6, we have to load the driver explicitly by this statement:

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

- However that statement is no longer needed, thanks to new update in JDBC 4.0 comes from Java 6. As long as you put the MySQL JDBC driver JAR file into your program's classpath, the driver manager can find and load the driver.

Lets apply the above steps in to a simple java program using MySQL with JDBC

```
//Import required packages
import java.sql.*;
public class JDBCExample {
      // JDBC driver name and database URL
      static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
      static final String DB_URL = "jdbc:mysql://localhost/";
      // Database credentials
      static final String USER = "username";
      static final String PASS = "password";

      public static void main(String[] args) {
      Connection conn = null;
      Statement stmt = null; try{

      //STEP 1: Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");
```

```
//STEP 2:Create a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL, USER, PASS);

//STEP 3 & 4:Create Statement & Execute a query
      System.out.println("Creating database...");
      stmt = conn.createStatement();

      String sql = "CREATE DATABASE STUDENTS";
      stmt.executeUpdate(sql);
      System.out.println("Database created successfully...");
}catch(SQLException se){

      //Handle errors for JDBC
      se.printStackTrace(); }catch(Exception e){

      //Handle errors for Class.forName
      e.printStackTrace();
}finally{
```

```
//finally block used to close resources
      try{
            if(stmt!=null)
                  stmt.close();
      }catch(SQLException se2){
      }// nothing we can do

      try{
            if(conn!=null)
            conn.close();
      }catch(SQLException se){
            se.printStackTrace();
      }//end finally try

}//end try
System.out.println("Goodbye!");
}//end main }
//end JDBCExample
```

Compile and Run the above Java code for the output

```
C:\>java JDBCExample
Connecting to database...
Creating database...
Database created successfully...
Goodbye!
C:\>
```

Demo

This Demo explains the MySQL database connection of a real world java project written in NetBeans through a JDBC connecter.

This consists of 4 steps as follows,

1. Install XAMMP for MySQL
2. Create User Database and User Table
3. Generate connection String in NetBeans Java project
4. Steps to insert records in table using JDBC

1. Install XAMMP for MySQL

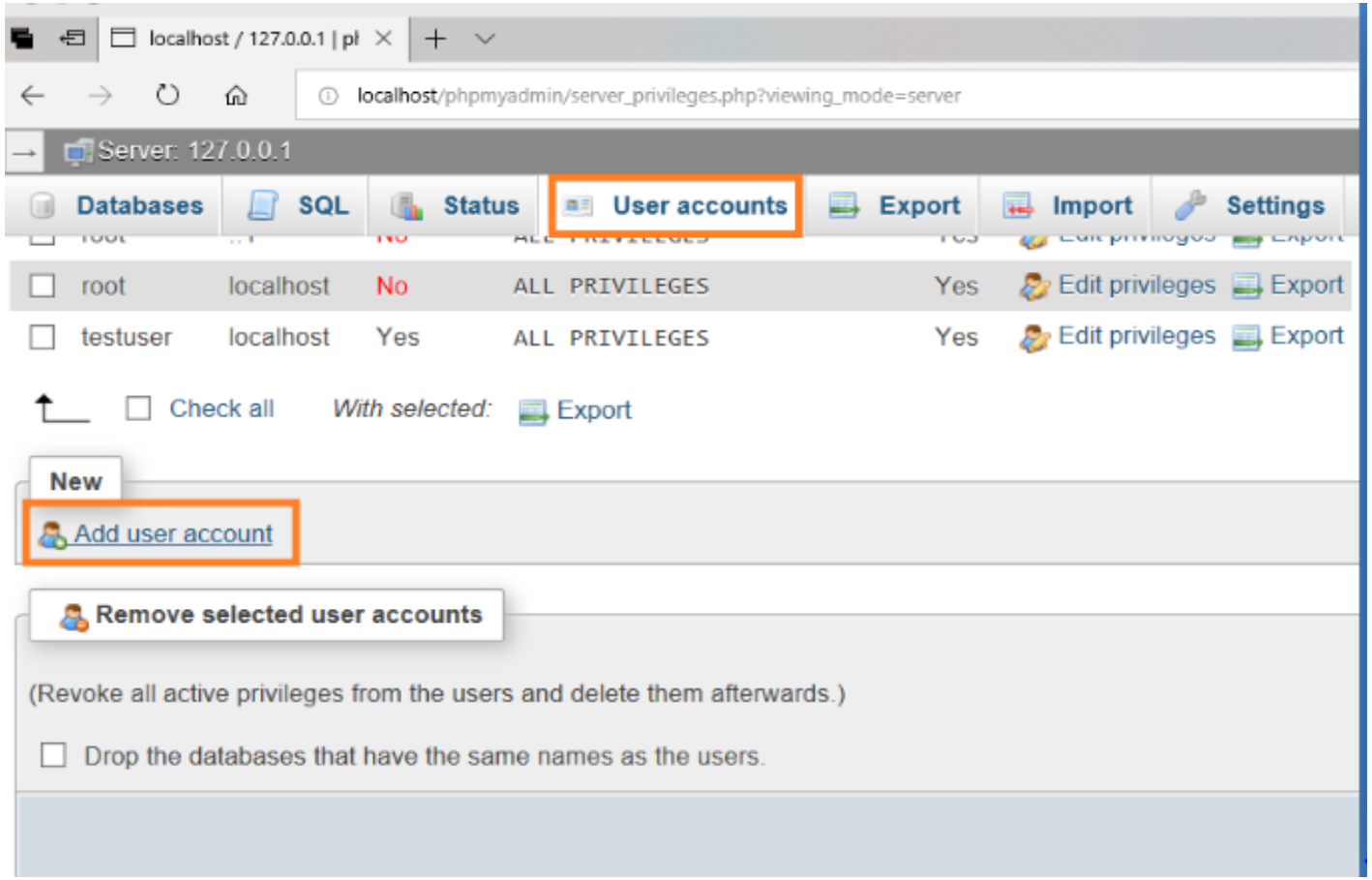
Its an open source software by Apache Friends, it contains Apache distributions for Apache server, MariaDB, H and Perl. It act as a local host or a local server.

So first download XAMMP and complete all installation wizards and run it. Then open XAMMP control panel and start apache and MySQL services as shown on the image.

2. Create User Database and User Table

To create a new database in MySQL,

open browser >> type localhost >> phpMyAdmin link >> click on new users tab >>create new user and database



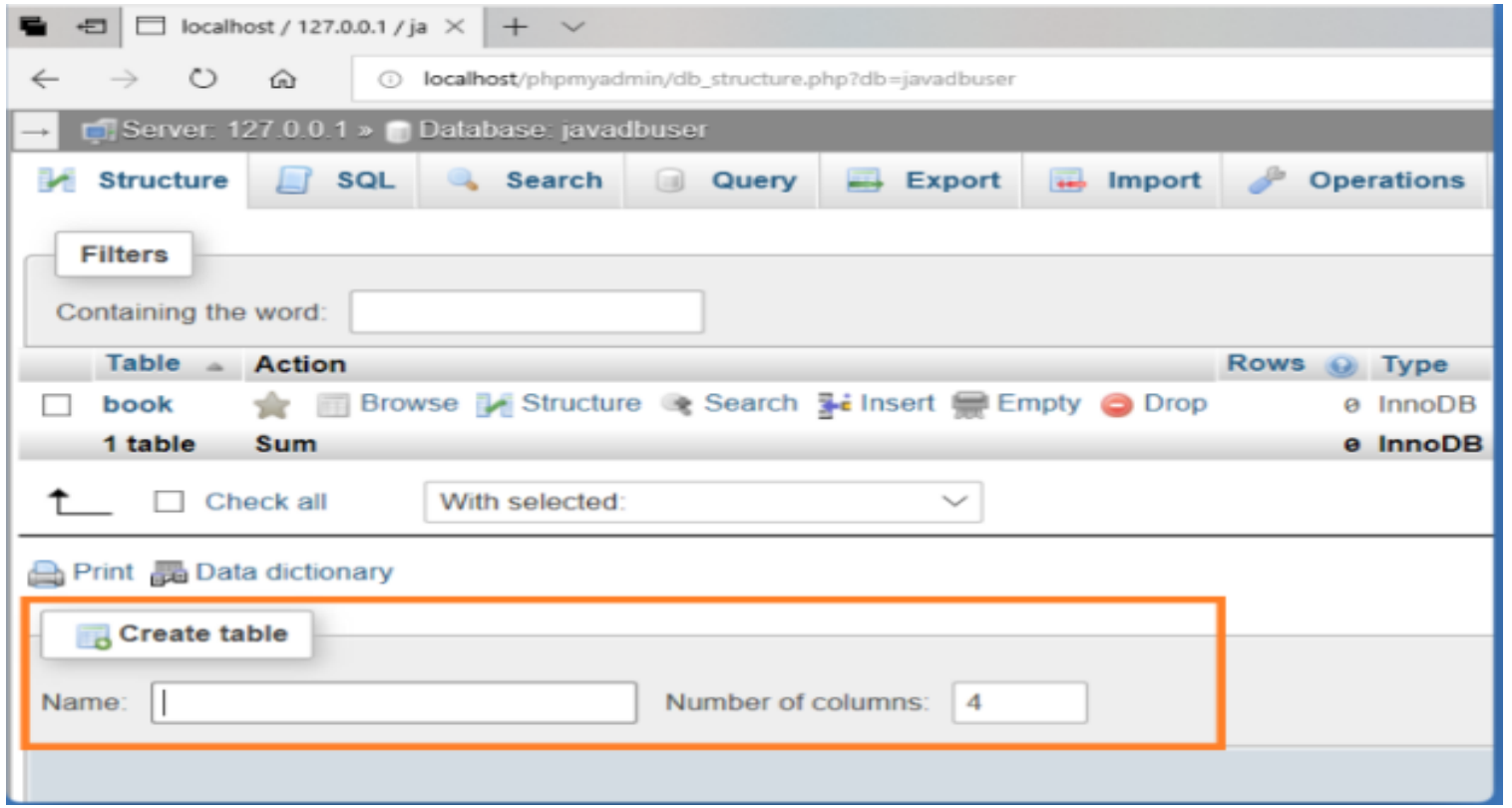
Click on Add new user >> Enter the username, password and host-name localhost

Note- Make sure you have selected the check box saying

Create database with same name and grant all privileges and all global privileges >> Click GO button

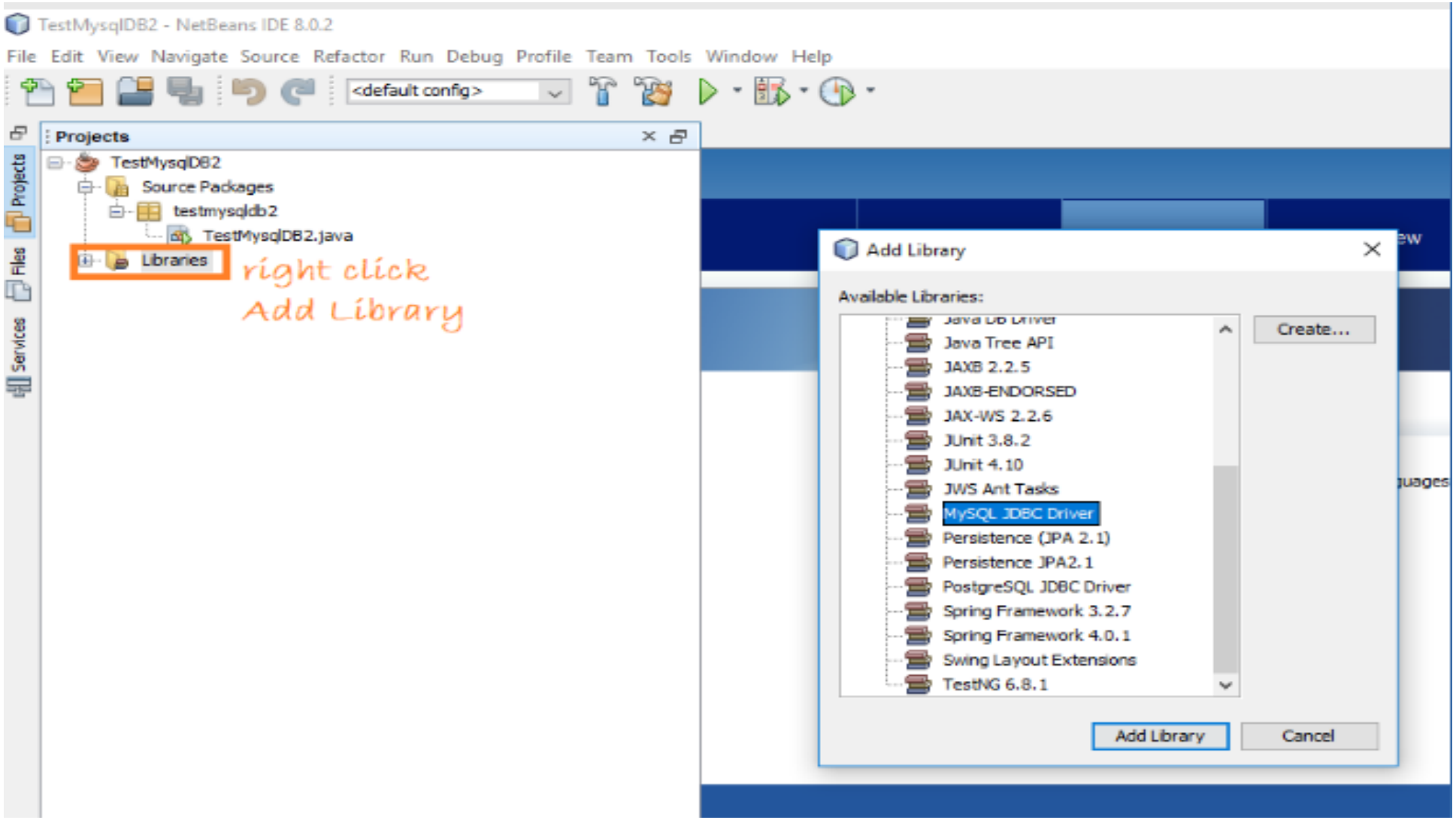
To Create Table

Click on Database Tab >>Database Name >> Enter Table Name >> Go



3. Generate connection String in NetBeans Java project

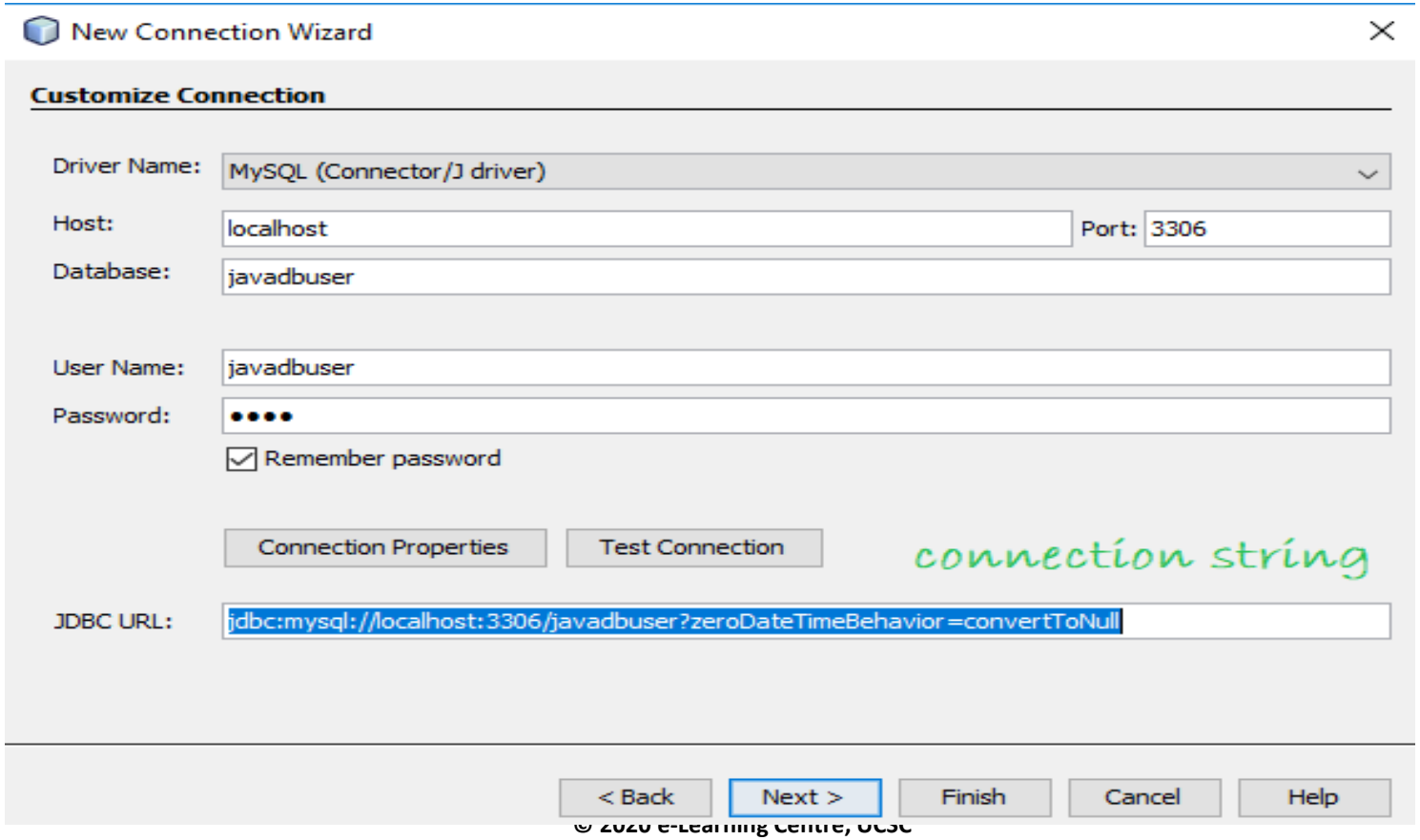
Open NetBeans and Create your project and Add library as shown



Create New Database Connection

Services Tab >> Click on Database>>New Connection>>Driver Mysql (Connector/ J Driver) >> Enter Username & Password of the database created.

You can check the connection by *Test Connection* Button. Then copy the JDBC URL which is the ***connection string***.



4. Steps to insert records in table using JDBC

These are the same steps we discussed on the sample java code in Slide Number 24. Lets look at another example of it.

1. Load Driver Manager

```
//step-1
Class.forName("com.mysql.jdbc.Driver");
```

You can skip this step for JDBC 4.0 API because this load driver instruction is integrated in getConnection() Method

2. Create Connection Object

```
//step-2                              //connection string, username, password
Connection con=DriverManager.getConnection(connectString,"javadbuser","1234");
```

3. Create Object Statement

```
//step-3        //Table fields: bookid,booknm,bookauthor,price
String q="insert into book values(111,'Java','Mr.X',700)";
```

Likewise, you can add INSER, SELECT, UPDATE & DELETE statements.

4. Execute Query on Database

```
//Step-4 //execute query on database
st.executeUpdate(q);
```

5. Close Connection

```
//Step-5 //Close Connection.
con.close();
```