FREE Live Classes

Java

Theory

Coding

RedSysTech
-Love to Learn...

Core Java

by Karthik Ponnusamy

Course Contents:-

- Core Java Basic Concepts
- OOP Concepts
- Fundamentals of Java Programming
- Java Control Flow Statements
- Deep dive into Collection Framework
- Deep dive into Multithreading
- Java 1.8 Features

Batch Starts from Jan - 18 - 2022

Mon to Thurs - 10 PM to 11 PM CST /

9:30 AM to 10:30 AM IST

Day 05: Agenda

JDBC & MySQL Local Setups

JDBC - What and Why is it required?
JDBC Architecture - Application, The JDBC API, DriverManager,
JDBC drivers
Types of JDBC Architecture(2-tier and 3-tier)
Setup MySQL DB locally
Write a simple JDBC application

JDBC - What and Why is it required?

1. JDBC stands for Java Database Connectivity.
2. It is a standard Java API that provides a set of classes and interfaces to access and manipulate relational databases.
3. JDBC allows Java applications to connect to a database, execute SQL statements, retrieve results and handle errors.
4. JDBC uses a driver-based architecture to communicate with different databases.

A JDBC driver is a software component that provides an implementation of the JDBC API for a particular database.

There are four types of JDBC drivers:

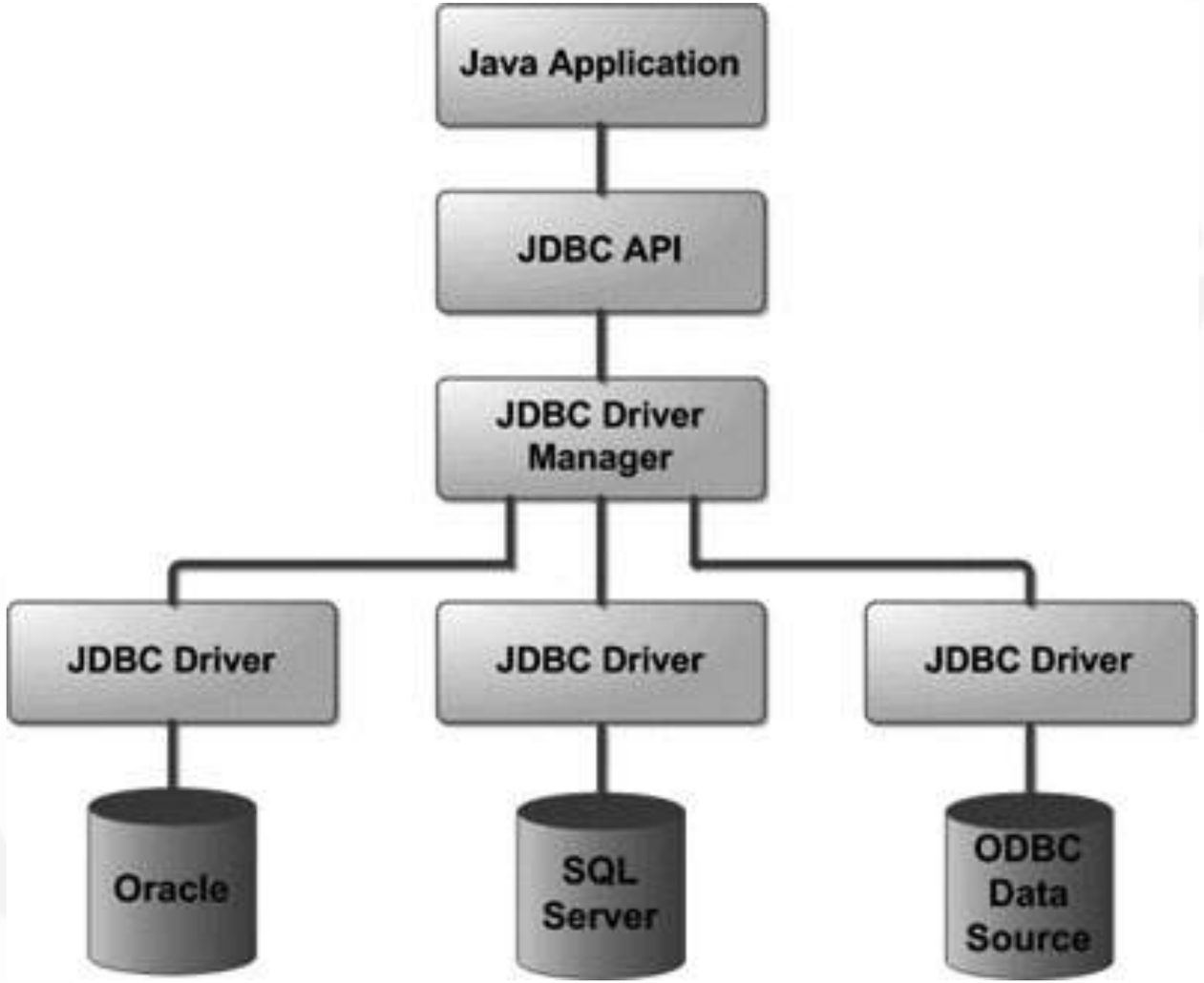
1. **JDBC-ODBC bridge driver:** uses an ODBC driver to connect to the database
2. **Native API driver:** uses a database-specific library to connect to the database
3. **Network protocol driver:** uses a middleware server to communicate with the database
4. **Thin driver:** a pure Java driver that communicates directly with the database over the network

To use JDBC in a Java application, you need to follow these steps:

1. Load the JDBC driver
2. Establish a connection to the database
3. Create a Statement object to execute SQL statements
4. Execute SQL statements and retrieve results
5. Handle any errors that may occur
6. Close the connection and release resources

JDBC is a powerful tool for developing database-driven applications in Java. It provides a standardized way to access and manipulate databases, which makes it easier to write portable and scalable applications.

JDBC Architecture



JDBC Architecture

JDBC (Java Database Connectivity) is a Java API that allows Java programs to connect to and interact with relational databases. The architecture of JDBC includes the following components:

Application: This is the program that uses the JDBC API to connect to the database, execute SQL statements, and process the results.

JDBC API: This is a set of classes and interfaces that define how Java programs interact with databases. It provides methods for connecting to a database, executing SQL statements, and processing the results.

JDBC Driver Manager: This component manages the JDBC drivers that are installed on the system. It provides methods for loading and unloading JDBC drivers and for creating connections to databases.

JDBC Driver: This is a software component that allows JDBC to communicate with a specific database. Each JDBC driver is designed to work with a specific database vendor and is responsible for translating the JDBC API calls into the appropriate database-specific commands.

Database: This is the relational database system that stores the data that the application interacts with.

When an application uses JDBC to connect to a database, it first loads the appropriate JDBC driver using the DriverManager. The driver is then responsible for establishing a connection to the database, executing SQL statements, and returning results to the application. The application uses the JDBC API to interact with the driver, which in turn interacts with the database to perform the necessary operations.

Types of JDBC Architecture

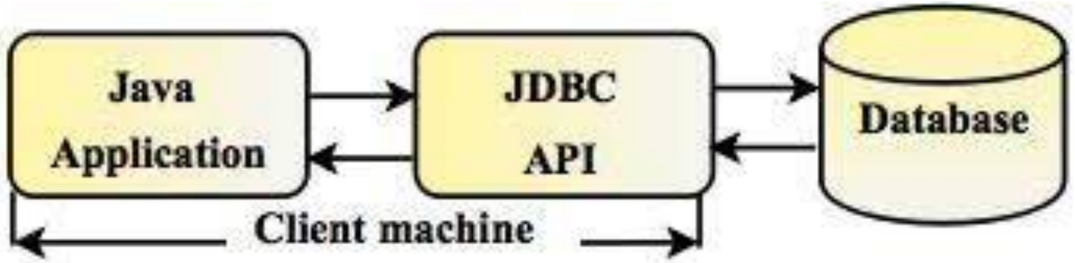


Fig: Two-tier Architecture of JDBC

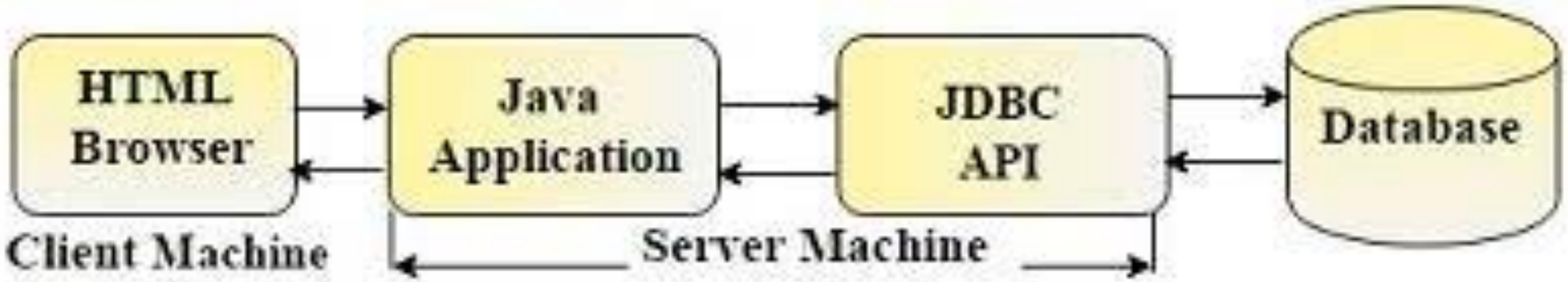


Fig: Three-tier Architecture of JDBC

Types of JDBC Architecture

There are two types of JDBC architecture:

Two-Tier Architecture:

The Two-Tier Architecture is also known as the Client/Server Architecture. It has two layers: client-side and database-side. In this architecture, the JDBC driver communicates directly with the database through a socket connection. The client-side consists of the Java application or applet, while the database-side consists of the database server.

Three-Tier Architecture:

The Three-Tier Architecture is also known as the Distributed Architecture. It has three layers: client-side, middleware, and database-side. In this architecture, the JDBC driver communicates with the middleware, which then communicates with the database. The client-side consists of the Java application or applet, the middleware is responsible for managing the communication between the client and the database, while the database-side consists of the database server. The advantage of this architecture is that it separates the presentation logic from the data storage and retrieval logic, making it easier to manage and maintain.

Setup MySQL DB locally

To set up MySQL on your local machine with MySQL Workbench, follow these steps:

Download and install MySQL:

You can download the MySQL Community Server from the official MySQL website. Follow the installation instructions for your operating system.

Download and install MySQL Workbench:

MySQL Workbench is a visual tool for managing MySQL databases. You can download it from the official MySQL website. Follow the installation instructions for your operating system.

Start MySQL:

After the installation is complete, start the MySQL server. On Windows, you can do this by opening the MySQL Command Line Client from the Start menu. On macOS or Linux, you can start the MySQL server using the terminal command mysql.server start.

Launch MySQL Workbench:

Launch MySQL Workbench and click on the "New Connection" button in the home screen.

Configure the connection:

In the "New Connection" window, enter a name for the connection and enter the connection details, including the hostname, port, username, and password. Click "Test Connection" to make sure that the connection is successful.

Create a database:

Once you are connected to MySQL, you can create a new database by clicking on the "Create a new Schema" button in the "Navigator" pane on the left-hand side of the screen.

Create tables:

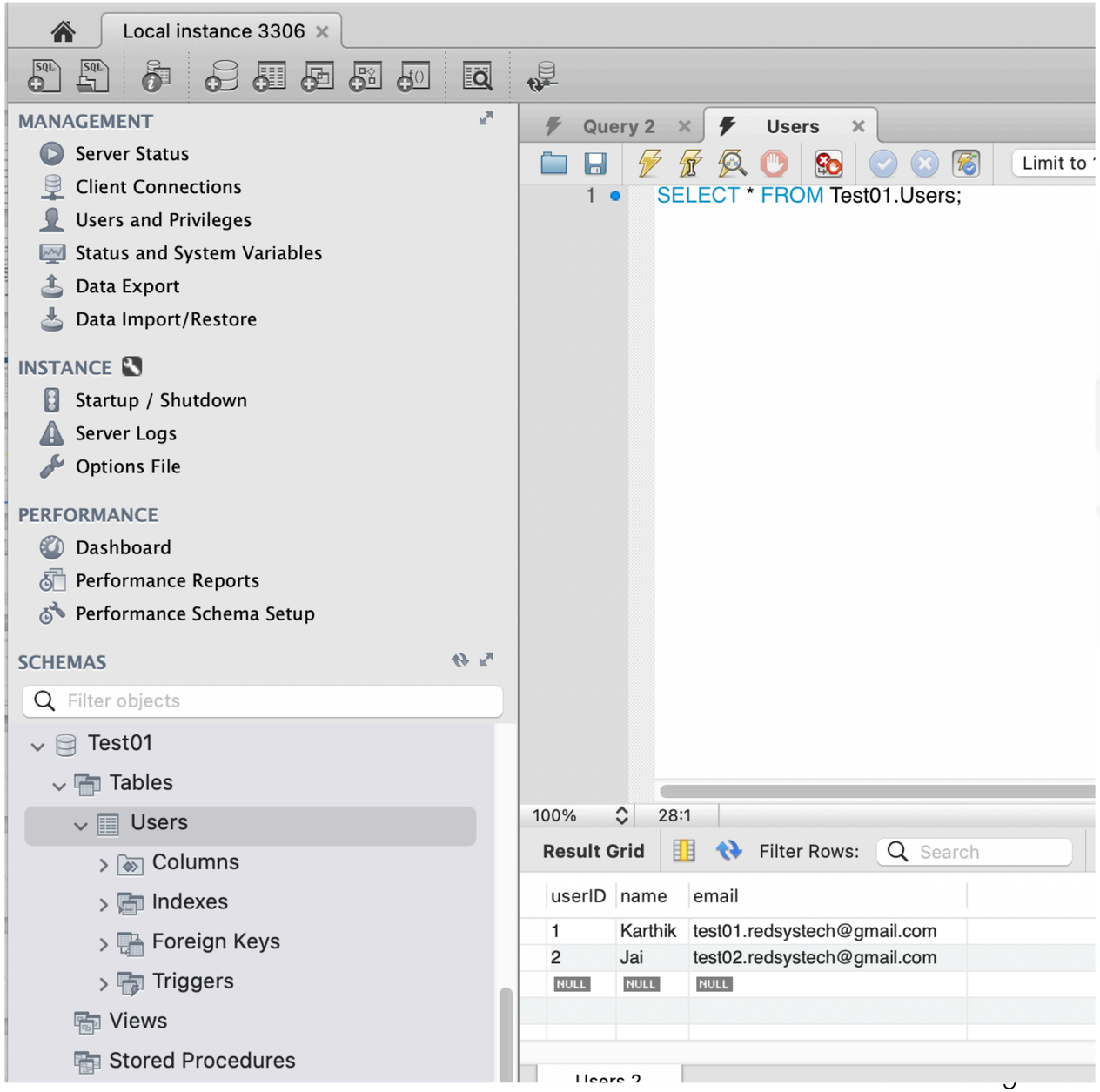
After creating the database, you can create tables by clicking on the "Create Table" button in the "Navigator" pane. In the "Create Table" window, enter the table name and the column details.

Insert data:

You can insert data into the table by clicking on the "Insert Rows" button in the "Navigator" pane. In the "Insert Rows" window, enter the data for each column and click "Apply" to save the changes.

Note: It's important to remember to secure your MySQL server by setting a strong password for the root user and disabling remote access, among other best practices.

Setup MySQL DB locally



Write a simple JDBC application

Here is an example of a simple JDBC application in Java that connects to a MySQL database and performs a SELECT query:

```
import java.sql.*;

public class JDBCTest {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydb";
        String username = "root";
        String password = "password";

        try {
            // Connect to the database
            Connection conn = DriverManager.getConnection(url, username, password);

            // Create a statement
            Statement stmt = conn.createStatement();

            // Execute a SELECT query
            ResultSet rs = stmt.executeQuery("SELECT * FROM users");

            // Iterate over the result set
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String email = rs.getString("email");
                System.out.println(id + ", " + name + ", " + email);
            }

            // Close the resources
            rs.close();
            stmt.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

In this example, we first define the connection URL, username, and password.

We then use the DriverManager.getConnection() method to create a connection to the database.

Next, we create a Statement object using the createStatement() method of the Connection object. We then execute a SELECT query using the executeQuery() method of the Statement object.

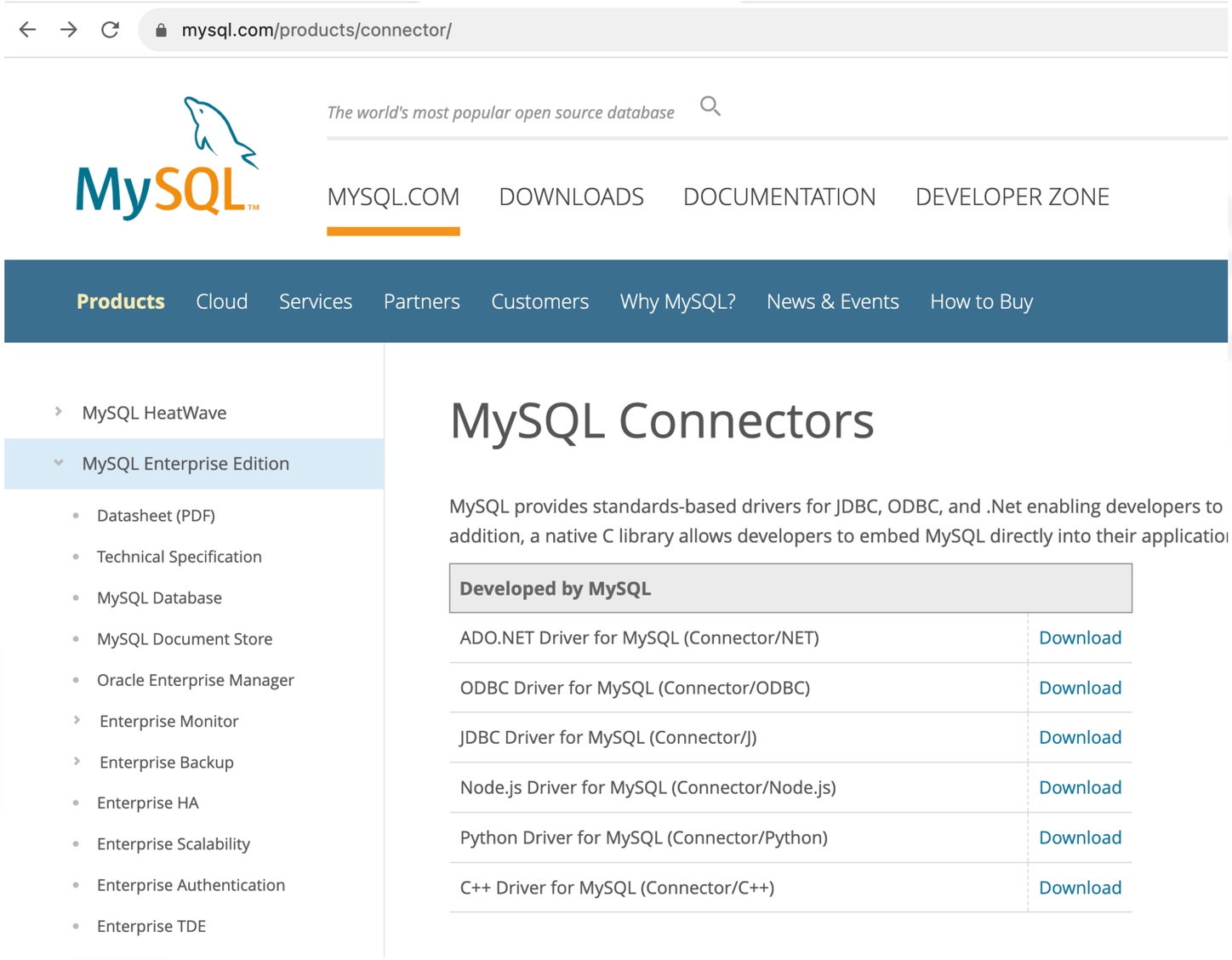
We iterate over the ResultSet using the next() method, and retrieve the values of the columns using the getInt() and getString() methods.

Finally, we close the resources in a finally block to ensure that they are properly released.

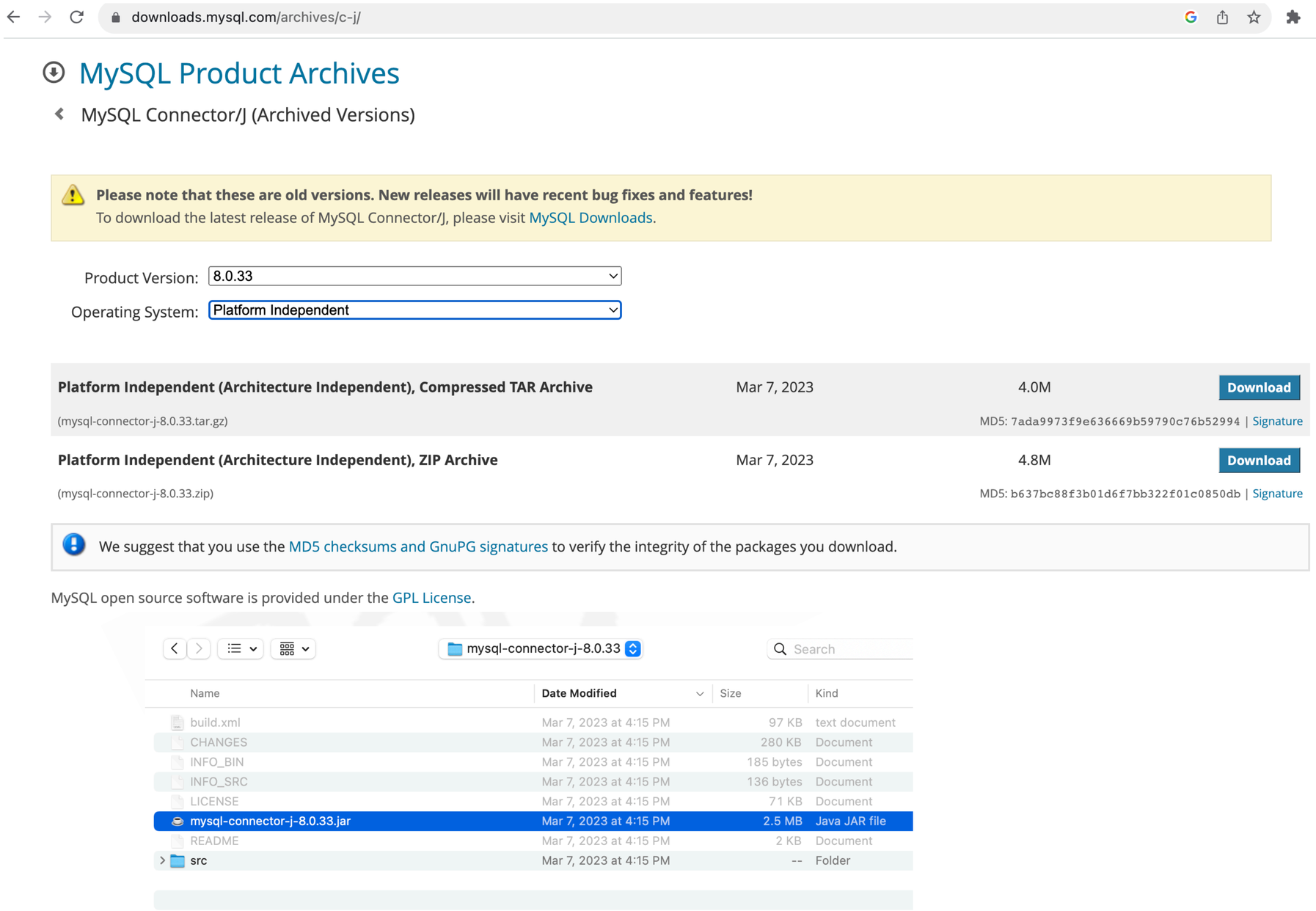
https://www.mysql.com/products/connector/

Write a simple JDBC application

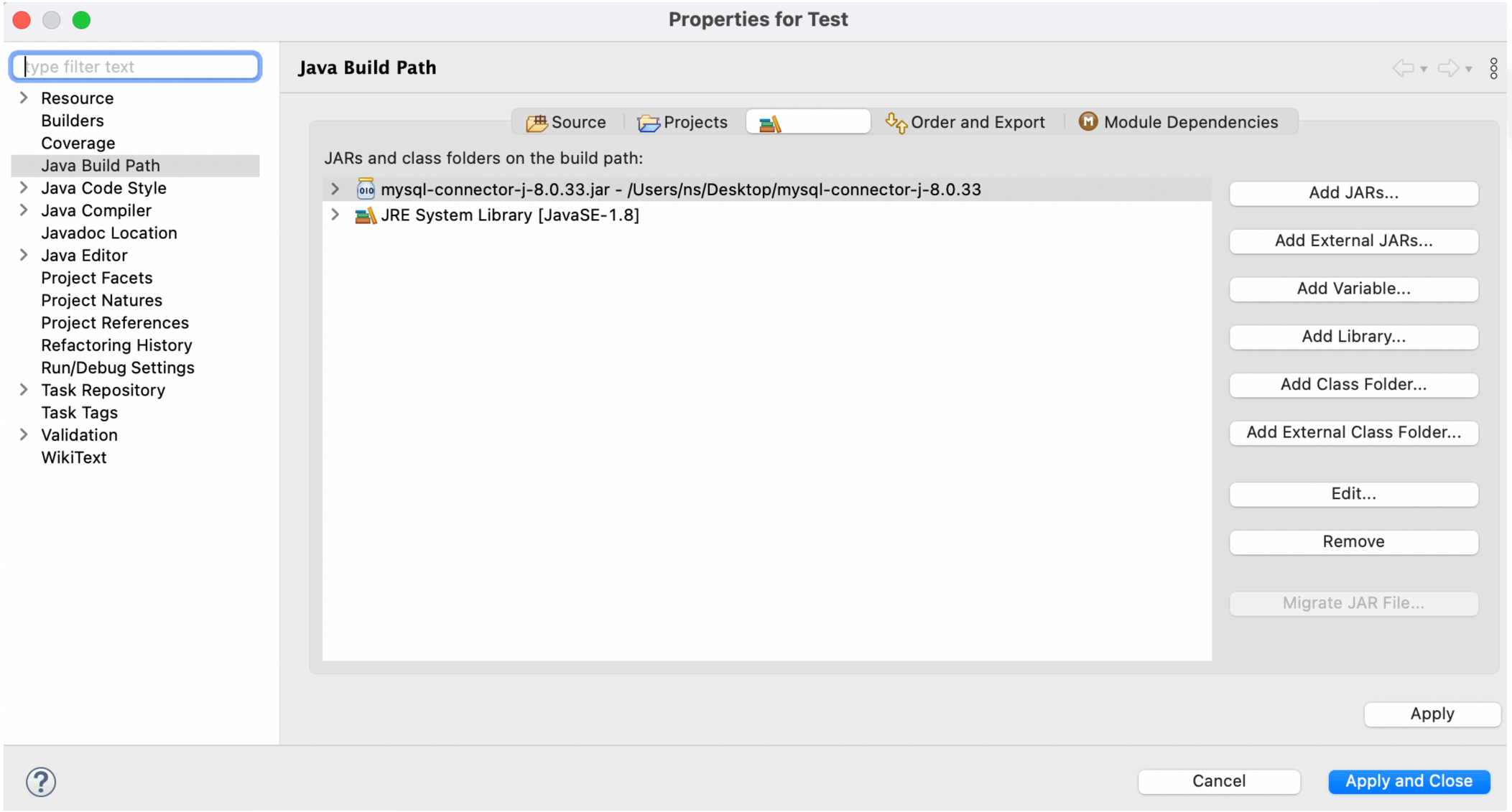
https://www.mysql.com/products/connector/



Write a simple JDBC application



Write a simple JDBC application



Write a simple JDBC application

JDBCTesting.java

```
public class JDBCTesting {

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/Test01";
        String username = "root";
        String password = "mysql";

        try {
            // Connect to the database
            Connection conn = DriverManager.getConnection(url, username, password);

            // Create a statement
            Statement stmt = conn.createStatement();

            // Execute a SELECT query
            ResultSet rs = stmt.executeQuery("SELECT * FROM Users");

            // Iterate over the result set
            while (rs.next()) {
                int id = rs.getInt("userID");
                String name = rs.getString("name");
                String email = rs.getString("email");
                System.out.println(id + ", " + name + ", " + email);
            }

            // Close the resources
            rs.close();
            stmt.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

```

Markers | Properties | Servers | Data Source Explorer | Snippets | Console

```
<terminated> JDBCTesting [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java  (Aug
1, Karthik, test01.redsystech@gmail.com
2, Jai, test02.redsystech@gmail.com
```

WHY?
WHAT?
WHERE?
WHEN?
WHO?
HOW?
WHERE?
HOW?
HOW?
WHAT? HOW? WHY?
WHO? WHERE? WHAT? HOW?
WHAT?
WHO?
WHERE?
HOW? WHO?
WHAT? HOW? WHY?
HOW?