JDBC Lab Manual

Ja-AAiT

Yared Y.

November 2023

Steps

1. Connection with MySQL DB
2. Register the Driver Class
3. Create Connection
4. Create Statement
5. Execute Queries
6. Close Connection

Download the MySQL connector Java driver in mvnrepository.com (or you can just use Maven build tool and the configure it in the pom.xml file)

Ja-AAiT

JDBC Connectivity Setup in Eclipse

Prerequisite to understand Java Database Connectivity with MySQL:-

1. Have MySQL on your System.

2. Have JDK on your System, Eclipse or other equivalent required.

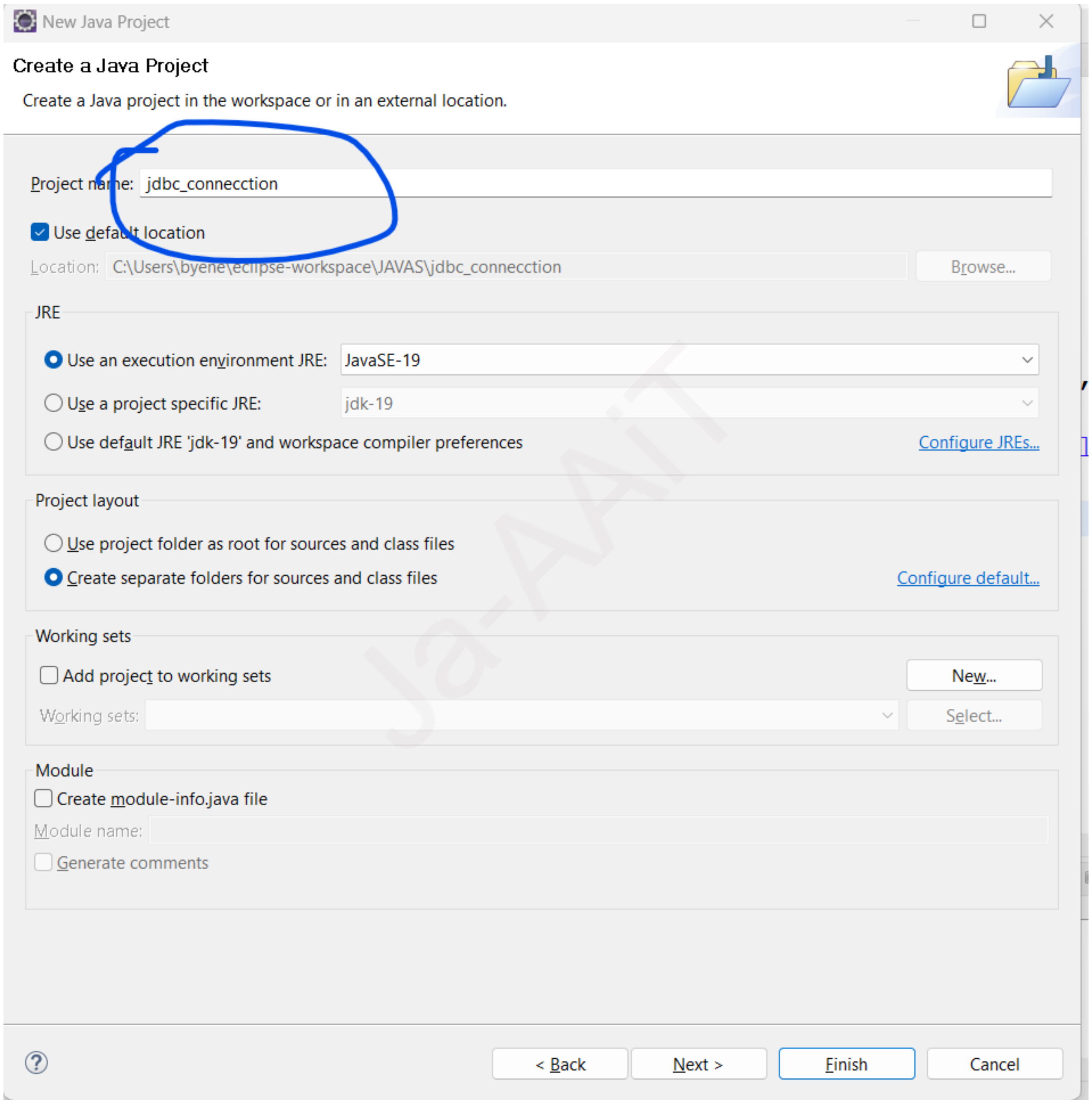
3. To set up the connectivity user should have MySQL Connector to the Java (JAR file), the ‘JAR’ file can be found in mvnrepository.com (or you can download it manually. That is optional.)

Have maven in your machine (Optional).

Ja-AAiT

Steps

1. Create Java project named **jdbc_connection**



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: jdbc_conneccton

[✓] Use default location

Location: C:\Users\byene\eclipse-workspace\JAVAS\jdbc_conneccton

JRE

(●) Use an execution environment JRE: JavaSE-19

( ) Use a project specific JRE: jdk-19

( ) Use default JRE 'jdk-19' and workspace compiler preferences

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

Working sets:

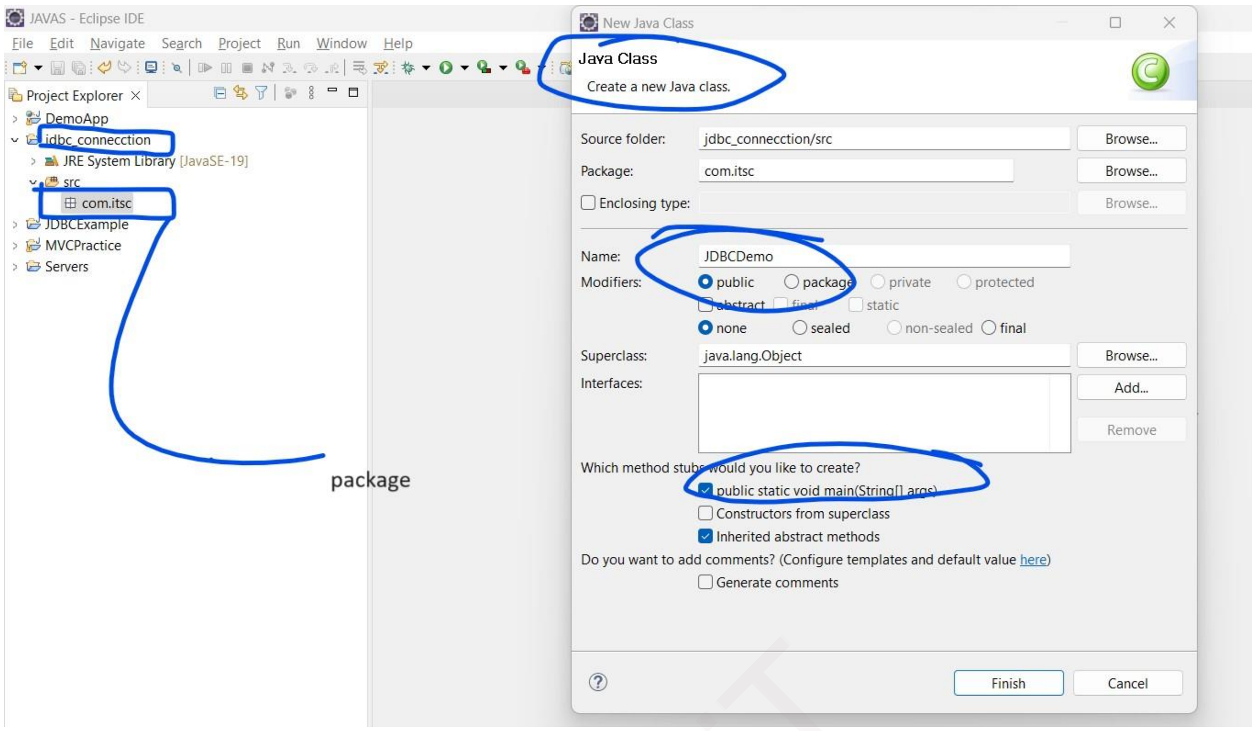
Module

[ ] Create module-info.java file

Module name:

[ ] Generate comments

Ja-AAiT



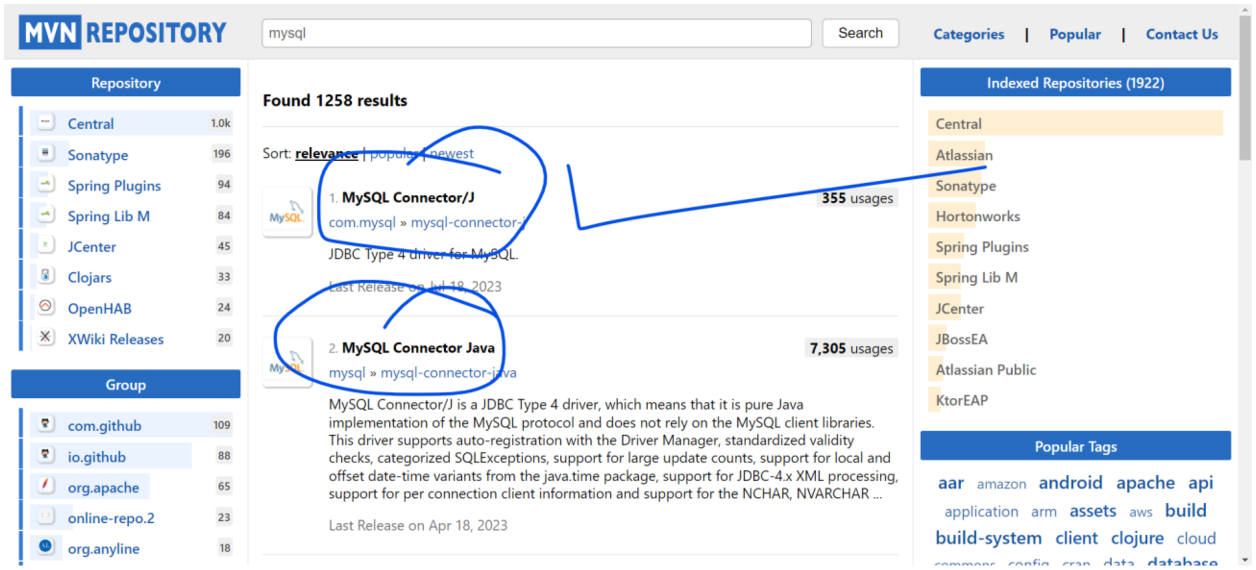
Java Class
Create a new Java class.

Source folder: jdbc_connecction/src
Package: com.itsc
Enclosing type: ☐
Name: JDBCDemo
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
[✓ none] sealed non-sealed final
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
☑ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

2. Add the JDBC Driver:

1. Download the MySQL Connector/J JDBC driver from the official MySQL website (https://dev.mysql.com/downloads/connector/j/). Or You can visit mvnrepository.com and search for MySQL driver.
2. Extract the downloaded zip file.
3. In Eclipse, right-click on your project in the "Package Explorer" and select "Properties."
4. In the project properties window, go to "Java Build Path."
5. Click on the "Libraries" tab.
6. Click "Add External JARs" and navigate to the location where you extracted the MySQL Connector/J JAR file.
7. Select the JAR file and click "Open."
8. Click "Apply and Close."


MVN REPOSITORY mysql Search Categories | Popular | Contact Us

Repository

Central	1.0k
Sonatype	196
Spring Plugins	94
Spring Lib M	84
JCenter	45
Clojars	33
OpenHAB	24
XWiki Releases	20

Group

com.github	109
io.github	88
org.apache	65
online-repo.2	23
org.anyline	18

Found 1258 results

Sort: **relevance** | popular | newest

1. **MySQL Connector/J** — **355** usages
com.mysql » mysql-connector-j
JDBC Type 4 driver for MySQL.
Last Release on Jul 18, 2023

2. **MySQL Connector Java** — **7,305** usages
mysql » mysql-connector-java
MySQL Connector/J is a JDBC Type 4 driver, which means that it is pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries. This driver supports auto-registration with the Driver Manager, standardized validity checks, categorized SQLExceptions, support for large update counts, support for local and offset date-time variants from the java.time package, support for JDBC-4.x XML processing, support for per connection client information and support for the NCHAR, NVARCHAR ...
Last Release on Apr 18, 2023

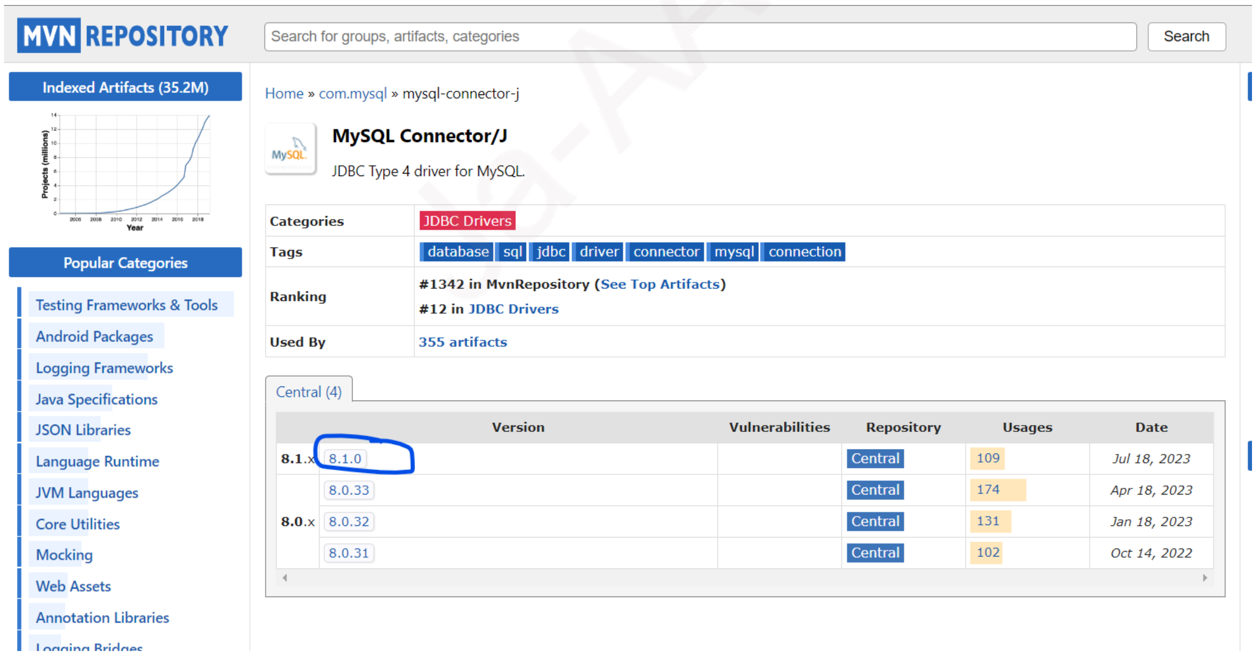
Indexed Repositories (1922)

Central
Atlassian
Sonatype
Hortonworks
Spring Plugins
Spring Lib M
JCenter
JBossEA
Atlassian Public
KtorEAP

Popular Tags

aar amazon android apache api application arm assets aws build build-system client clojure cloud

You can download the latest version:


MVN REPOSITORY Search for groups, artifacts, categories Search

Indexed Artifacts (35.2M)

Popular Categories

Testing Frameworks & Tools
Android Packages
Logging Frameworks
Java Specifications
JSON Libraries
Language Runtime
JVM Languages
Core Utilities
Mocking
Web Assets
Annotation Libraries

Home » com.mysql » mysql-connector-j

MySQL Connector/J

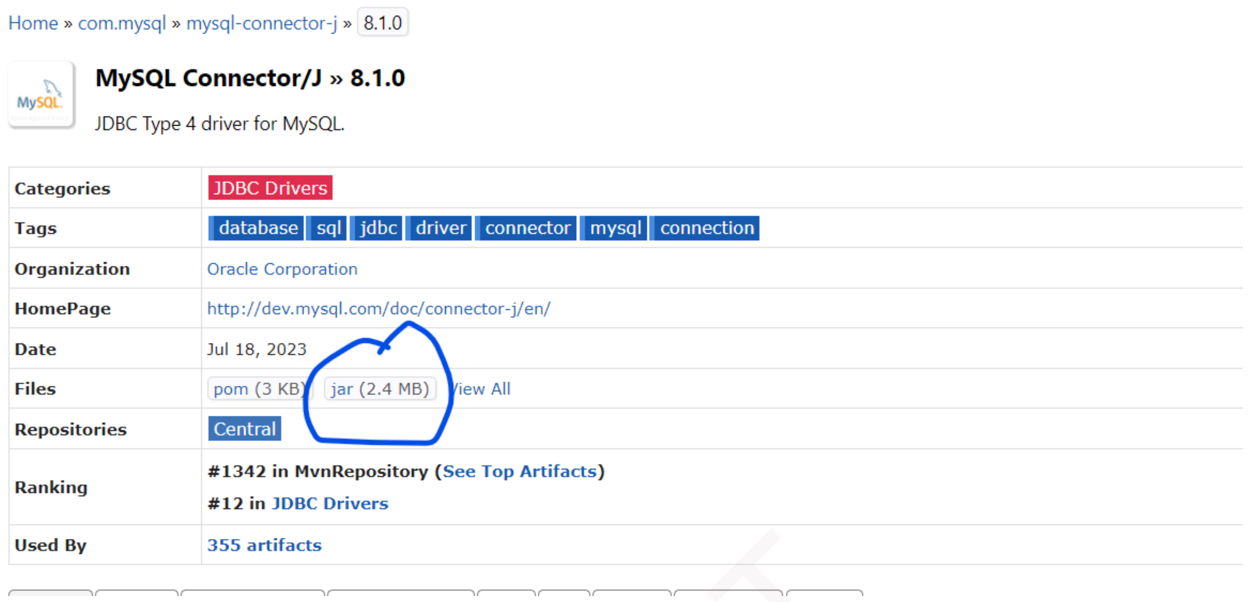
JDBC Type 4 driver for MySQL.

Categories	JDBC Drivers
Tags	database sql jdbc driver connector mysql connection
Ranking	#1342 in MvnRepository (See Top Artifacts) #12 in JDBC Drivers
Used By	355 artifacts

Central (4)

	Version	Vulnerabilities	Repository	Usages	Date
8.1.x	8.1.0		Central	109	Jul 18, 2023
8.0.x	8.0.33		Central	174	Apr 18, 2023
	8.0.32		Central	131	Jan 18, 2023
	8.0.31		Central	102	Oct 14, 2022

Download the JAR file:



Home » com.mysql » mysql-connector-j » 8.1.0

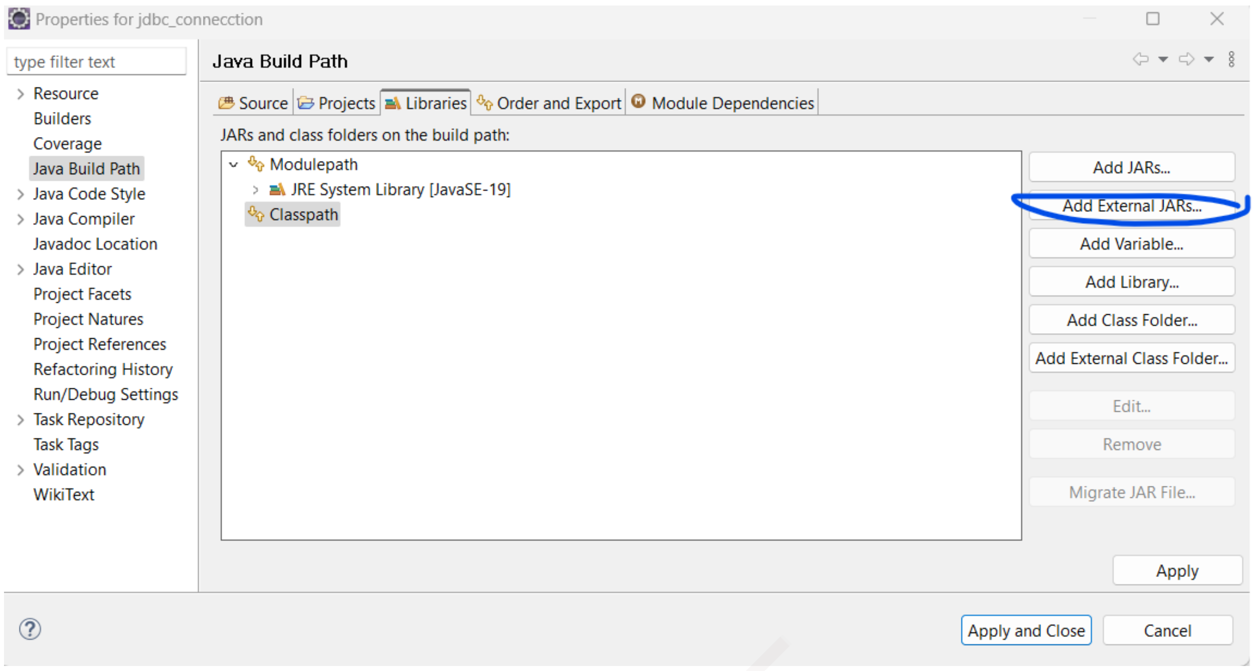
MySQL Connector/J » 8.1.0

JDBC Type 4 driver for MySQL.

Categories	JDBC Drivers
Tags	database sql jdbc driver connector mysql connection
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	Jul 18, 2023
Files	pom (3 KB) jar (2.4 MB) View All
Repositories	Central
Ranking	#1342 in MvnRepository (See Top Artifacts) #12 in JDBC Drivers
Used By	355 artifacts

Ja-AAiT

Right click on the project folder, select Build Path, and from the submenu select Add External Archives...



In the JAR Selection panel that comes up, navigate to the folder where the JAR file you downloaded exists.

Ja-AAiT

Note:
JDBC Drivers:

JDBC driver is a collection of classes which implements interfaces defined in the JDBC API for opening database connections, interacting with database and closing database connections.

Now to establish your connection, follow the steps:

Load the MySQL JDBC driver:

First step is to load or register the JDBC driver for the database.
Class class provides forName() method to dynamically load the driver class.

Syntax:

```
Class.forName("driverClassName");
```

com.mysql.cj.jdbc.Driver is the fully qualified class name of the MySQL JDBC driver's implementation class. This class is used to register the driver with the Java application, and it's necessary to load the driver into memory before you can establish a connection to a MySQL database.

Here's what each part of the class name means:

- `com.mysql.cj`: This is the package name where the MySQL Connector/J (JDBC driver for MySQL) classes are located.
- `jdbc`: This is a subpackage within `com.mysql.cj` that contains the JDBC-related classes.
- `Driver`: This is the name of the class that implements the JDBC driver for MySQL.

When you use **Class.forName("com.mysql.cj.jdbc.Driver")**, you are telling Java to load the class **com.mysql.cj.jdbc.Driver** into memory. This registration step is essential because it allows the Java application to recognize and use the MySQL JDBC driver to establish a database connection. Without this step, you won't be able to use the driver to connect to a MySQL database.

Class.forName()

The `Class.forName` method in JDBC (Java Database Connectivity) is used to dynamically load and register a JDBC driver class in your Java application. This is a crucial step when working with JDBC because it allows your Java application to establish a database connection using a specific database driver.

Here's how `Class.forName` is typically used in JDBC:

1. **Loading the JDBC Driver:** Before you can connect to a specific database using JDBC, you need to load the JDBC driver class that corresponds to that database. Different databases have their own JDBC driver classes. In the example provided earlier (`Class.forName("com.mysql.cj.jdbc.Driver")`), `"com.mysql.cj.jdbc.Driver"` is the fully qualified class name for the MySQL JDBC driver.

   By loading this class using `Class.forName`, you make it available for use in your application.

2. **Driver Registration:** When you call `Class.forName` with the appropriate driver class name, the driver class is loaded and registered with the `DriverManager`. The `DriverManager` is a part of the JDBC API and is responsible for managing a list of database drivers. Registering the driver allows the `DriverManager` to recognize it and use it for database connections.

3. **Establishing a Database Connection:** Once the driver is loaded and registered, you can use the `DriverManager` to establish a database connection. You typically provide the database URL, username, and password as parameters to the `DriverManager.getConnection` method, and it returns a connection to the database.

It's worth noting that starting with Java 6 and JDBC 4.0, the Class.forName method is no longer required to load the driver explicitly. The driver is typically loaded and registered automatically when you place the JDBC driver JAR file on the classpath. However, in older or non-standard scenarios, you may still encounter Class.forName being used to load the driver class.

Establish a connection to the database:

Before performing any database operation via JDBC, we have to open a database connection. To open a database connection we can call getConnection() method of DriverManager class.

Second step is to open a database connection. DriverManager class provides the facility to create a connection between a database and the appropriate driver.

To open a database connection we can call getConnection() method of DriverManager class.

Note: Create a database called "**StudentsDB**" using MySQL workbench or using the CLI of MySQL [if you're using other DBMS, follow the required steps to create your databases and configuration pahts.]

Ja-AAiT

Syntax:

```
Connection connection = DriverManager.getConnection(url, user, password)
```

```java
import java.sql.*;

public class JDBCDemo {

public static void main(String[] args) {

     try {

          String driver = "com.mysql.cj.jdbc.Driver";

          String url = "jdbc:mysql://localhost:3306/StudentsDB";

          String username = "root"; //your username

          String password = "root"; //your password

          Class.forName(driver);  //optional

          Connection conn = DriverManager.getConnection(url,
     username, password);

          System.out.println("Established Connection");

     }catch (Exception e) {

          e.printStackTrace();// TODO: handle exception

     }

}

}
```

If this goes fine, you have successfully connected.

Ja-AAiT

Notes:

Establishing a connection to a database is a critical step in working with JDBC (Java Database Connectivity). You need to provide the necessary connection information, including the database URL, username, and password, to create a connection object. Once the connection is established, you can use it to interact with the database. Here's more information about establishing a connection:

Database URL (Uniform Resource Locator): The database URL is a string that specifies the location and configuration of the database you want to connect to. The format of the URL depends on the database you are using. In the example provided earlier, the MySQL database URL looks like this:

```
String url = "jdbc:mysql://localhost:3306/your_database";
```

- `"jdbc:mysql://"`: This part indicates the JDBC protocol and the database type (MySQL).
- `"localhost:3306"`: This specifies the hostname (or IP address) and the port number of the database server.
- `"your_database"`: This is the name of the specific database you want to connect to.

For MySQL, the URL format is often `jdbc:mysql://hostname:port/database`. This URL provides information about the database server, the port it's listening on, and the specific database you want to access.

Username and Password: You typically need to provide a username and password to access the database. These credentials are used for authentication.
Establishing the Connection: The `DriverManager.getConnection` method is used to establish a connection to the database. It takes the database URL, username, and password as arguments and returns a **`Connection`** object.

Here's the code snippet again:

```
// Establish a database connection
String url = "jdbc:mysql://localhost:3306/your_database";
String username = "your_username";
String password = "your_password";
Connection connection = DriverManager.getConnection(url, username,
password);
```

After establishing the connection, you can use the `**connection**` object to perform various database operations, such as executing SQL queries, inserting, updating, or deleting records, and retrieving data from the database.

Once the driver is loaded, **DriverManager.getConnection** establishes a connection to the database server specified in the URL, using the provided username and password.

If the connection is successfully established, it returns a **Connection** object that can be used to interact with the database. This object allows you to create statements, execute queries, and perform various database operations.
The `Connection` object is your gateway to the database.
When you are done with the connection, it's essential to close it to release any associated resources:

```
// Close the connection when done
connection.close();
```

Failure to close the connection can lead to resource leaks and decreased performance, so it's essential to always close the connection when you no longer need it. Typically, this is done within a `finally` block to ensure that the connection is closed even if an exception is thrown during database operations.
Keep in mind that handling exceptions and errors properly, such as catching `SQLException`, is crucial when working with database connections to ensure that your application handles any potential issues gracefully.

Creating a database

To create a new database in MySQL using Java and JDBC, you need to execute a SQL query that creates the database. Here's an example of how to do it:

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;


public class CreateDatabaseExample {

    public static void main(String[] args) {

        String url = "jdbc:mysql://localhost:3306/";

        String username = "your_username";

        String password = "your_password";

        String databaseName = "new_database";

        try {

            // Establish a connection to the MySQL server

            Connection connection = DriverManager.getConnection
                                            (url, username, password);

            // Create a statement

            Statement statement = connection.createStatement();

            // Execute the SQL query to create the new database

            String createDatabaseSQL = "CREATE DATABASE " + databaseName;

            statement.executeUpdate(createDatabaseSQL);

            System.out.println("Database '" + databaseName + "' created
successfully.");

            // close resources

            statement.close();

            connection.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

This Java code demonstrates how to use JDBC (Java Database Connectivity) to create a new database using MySQL. I'll explain the important ones:
`**Connection** connection = **DriverManager**.**getConnection**("jdbc:mysql://localhost:3306/", "root", "root");`
   - This line establishes a database connection using the `DriverManager.getConnection` method. It connects to a MySQL database running on the local machine at port 3306 with the username "root" and password "root". The connection object is stored in the `connection` variable.

`**Statement** statement = connection.**createStatement**();`
   - This line creates a `**Statement**` object (`statement`) using the connection (`connection`). A `**Statement**` is used to execute SQL queries against the database.

`String query = "Create database database_name";`
   - This line defines an SQL query as a string. The query is intended to create a new database named "database_name."

`statement.execute(query);`
   - This line executes the SQL query stored in the `query` string using the `execute` method of the `Statement` object (`statement`). This query creates the "new_database" database.

CREATE TABLE

```
package com.itsc;

import java.sql.*;

public class StatementWithResultSet {

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

       String jdbcURL = "jdbc:mysql://localhost:3306/teachersdb";

      String username = "root";

        String pwd = "root";

        Connection connection = DriverManager.getConnection(jdbcURL,
username, pwd);

        Statement statement = connection.createStatement();

        String query = "Select * from teachers";

        String createTableSQL = "CREATE TABLE teacher1 (" +

                                "id int auto_increment primary key," +

                                "first_name varchar(255)," +

                                "last_name varchar(255)," +

                                "school varchar(255)," +

                                "hire_date date," +

                                "salary decimal(10, 2))";

        statement.executeUpdate(createTableSQL);

        System.out.println("Ttable ‘teachers’ created successfully.”);

    }

}
```

Ja-AAT

INSERT DATA INTO TABLE

To populate the "teachers" table with the provided set of data, you can use SQL `INSERT` statements. Here's a Java JDBC example to insert the data into the "teachers" table:

```
package com.itsc;

import java.sql.*;

public class StatementWithResultSet {

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

       String jdbcURL = "jdbc:mysql://localhost:3306/teachersdb";

      String username = "root";

        String pwd = "root";

        Connection connection = DriverManager.getConnection(jdbcURL,
username, pwd);

        Statement statement = connection.createStatement();

        String query = "Select * from teachers";

        String createTableSQL = "CREATE TABLE teacher1 (" +

                                "id int auto_increment primary key," +

                                "first_name varchar(255)," +

                                "last_name varchar(255)," +

                                "school varchar(255)," +

                                "hire_date date," +

                                "salary decimal(10, 2))";

        statement.executeUpdate(createTableSQL);

        System.out.println("Ttable ‘teachers’ created successfully.”);

    }

       // define SQL insert statements for each row of data

        String[] insertStatements = {
```

Ja-AAiT

```
                "insert into teacher1(first_name, last_name, school,
hire_date, salary) values('Aster', 'Nega', 'Yekatit 12', '2021-01-01',
8000)",

                "insert into teacher1(first_name, last_name, school,
hire_date, salary) values('Jemal', 'Edris', 'Bole', '2021-09-11', 20000)",

                "insert into teacher1(first_name, last_name, school,
hire_date, salary) values('Haile', 'Anaol', 'Saris', '2022-01-22', 15000)",

                "insert into teacher1(first_name, last_name, school,
hire_date, salary) values('Teddy', 'Anbesaw', 'Bole', '2021-01-01', 8000)"

 };

        for(String stmt : insertStatements) {

           statement.executeUpdate(stmt);

        }

        System.out.println("Data Inserted Successfullly.");
```

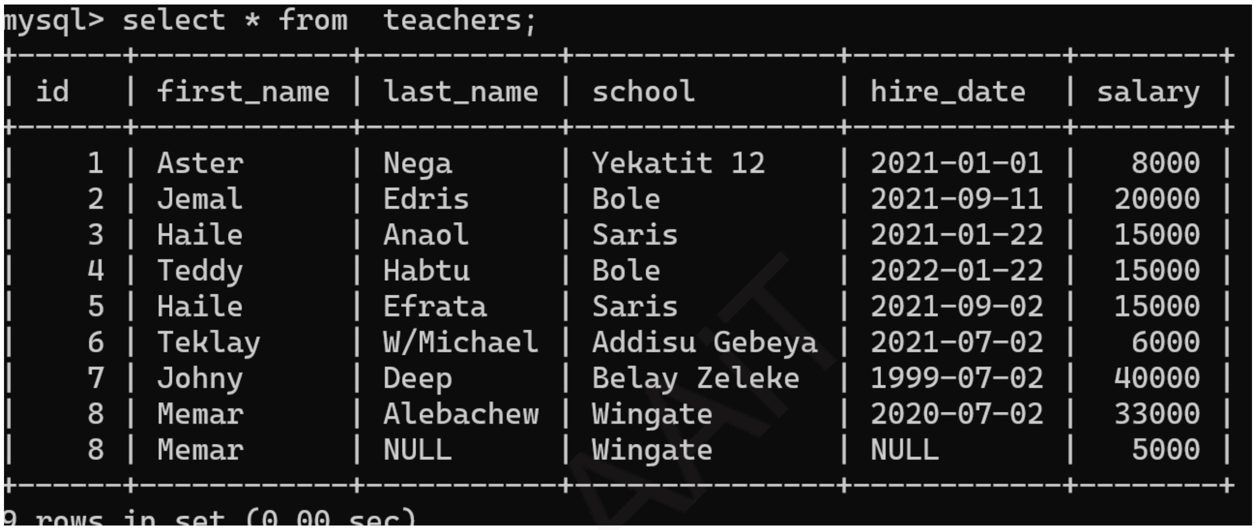
Ja-AAiT

SELECT FROM TABLE

STATEMENT

In Java, a `Statement` is an interface provided by the JDBC (Java Database Connectivity) API that allows you to execute SQL queries against a database.

Example: Given the following teachers table, write a java code to retrieve the data



```
mysql> select * from  teachers;
+------+------------+-----------+---------------+------------+--------+
| id   | first_name | last_name | school        | hire_date  | salary |
+------+------------+-----------+---------------+------------+--------+
|    1 | Aster      | Nega      | Yekatit 12    | 2021-01-01 |   8000 |
|    2 | Jemal      | Edris     | Bole          | 2021-09-11 |  20000 |
|    3 | Haile      | Anaol     | Saris         | 2021-01-22 |  15000 |
|    4 | Teddy      | Habtu     | Bole          | 2022-01-22 |  15000 |
|    5 | Haile      | Efrata    | Saris         | 2021-09-02 |  15000 |
|    6 | Teklay     | W/Michael | Addisu Gebeya | 2021-07-02 |   6000 |
|    7 | Johny      | Deep      | Belay Zeleke  | 1999-07-02 |  40000 |
|    8 | Memar      | Alebachew | Wingate       | 2020-07-02 |  33000 |
|    8 | Memar      | NULL      | Wingate       | NULL       |   5000 |
+------+------------+-----------+---------------+------------+--------+
9 rows in set (0.00 sec)
```

Ja-AAT

```java
package com.itsc;

import java.sql.*;

public class StatementWithResultSet {

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

       String jdbcURL = "jdbc:mysql://localhost:3306/teachersdb";

      String username = "root";

        String pwd = "root";

        Connection connection = DriverManager.getConnection(jdbcURL,
username, pwd);

        Statement statement = connection.createStatement();

        String query = "Select * from teachers";

        ResultSet res = statement.executeQuery(query);

      while(res.next()) {

            int id = res.getInt("id");

            String fname = res.getString("first_name");

//          String lname = res.getString("last_name");

//          String school = res.getString("school");

            Date hire_date = res.getDate("hire_date");

            double salary = res.getDouble("salary");

            System.out.println("Teacher ID:  " + id);

            System.out.println("Teacher Name:  " + fname);

            System.out.println("Teacher Hire Date:  " + hire_date);

            System.out.println("Teacher Salary:  " + salary);

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

        }

        res.close();        statement.close();        connection.close();

    }

}
```

Ja-AiT

In JDBC, the **`Statement`** interface provides several methods for executing SQL queries and commands. Here are some of the key methods related to `Statement`:

1. executeQuery(String sql):
   - Executes a SQL query that returns a `ResultSet`. Typically used for `SELECT` statements to retrieve data.

2. executeUpdate(String sql):
   - Executes an SQL statement (typically `INSERT`, `UPDATE`, or `DELETE`) and returns the number of affected rows. It's used for data manipulation operations.

3. execute(String sql):
   - Executes any SQL statement (either a query or a command) without returning a result. It's used for a wide range of SQL operations.

Ja-AAiT

Practice Time:

Certainly! Here's a series of tasks that involve connecting to a MySQL database, creating a database and table, populating the table, and performing various operations using Java:

Task 1: Connect to MySQL Database, Create Database, and Table

- Connect to a MySQL server with appropriate credentials.
- Create a database named `StudentsDB`.
- Create a table named `students` with the specified schema (id, firstname, lastname, grade).

Task 2: Insert Data

- Insert a single row into the `students` table as an example.
- Insert ten more rows with different data.

Task 3: Retrieve Data

- Retrieve and display five rows from the `students` table.

Task 4: Update Data

- Update the `firstname` of a student with a given `id`.

Task 5: Delete Data

- Delete a row from the `students` table with a given `id`.

Task 6: Calculate Average Grade

- Calculate and display the average grade of all students in the table.

Here's a sample Java program to help you get started with these tasks:

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
```

Ja-AAiT

```
import java.sql.ResultSet;
import java.sql.Statement;

public class StudentDatabaseOperations {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/StudentsDB";
        String username = "your_username";
        String password = "your_password";

        try {
            // Task 1: Connect to MySQL Database, Create Database, and Table
            Connection connection = DriverManager.getConnection(url, username,
password);
            Statement statement = connection.createStatement();
            statement.execute("CREATE TABLE IF NOT EXISTS students (id INT
PRIMARY KEY, firstname VARCHAR(255), lastname VARCHAR(255), grade
INT)");

            // Task 2: Insert Data
            insertSampleData(connection);

            // Task 3: Retrieve Data
            retrieveData(connection);

            // Task 4: Update Data
            updateStudentName(connection, 1, "UpdatedFirstName");

            // Task 5: Delete Data
            deleteStudent(connection, 2);

            // Task 6: Calculate Average Grade
            calculateAverageGrade(connection);

            // Close the resources
            statement.close();
            connection.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Ja-AAiT

```
    private static void insertSampleData(Connection connection) {
        try {
            // Insert a single row
            PreparedStatement insertSingle = connection.prepareStatement("INSERT
INTO students (id, firstname, lastname, grade) VALUES (?, ?, ?, ?)");
            insertSingle.setInt(1, 1);
            insertSingle.setString(2, "John");
            insertSingle.setString(3, "Doe");
            insertSingle.setInt(4, 90);
            insertSingle.executeUpdate();

            // Insert ten more rows (you can use a loop)
            // ...

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void retrieveData(Connection connection) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM students
LIMIT 5");

            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String firstname = resultSet.getString("firstname");
                String lastname = resultSet.getString("lastname");
                int grade = resultSet.getInt("grade");
                System.out.println("ID: " + id + ", Name: " + firstname + " " + lastname
+ ", Grade: " + grade);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    }

    private static void updateStudentName(Connection connection, int id, String
newFirstName) {
        try {
            PreparedStatement updateStatement =
connection.prepareStatement("UPDATE students SET firstname = ? WHERE id =
?");
            updateStatement.setString(1, newFirstName);
            updateStatement.setInt(2, id);
            updateStatement.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void deleteStudent(Connection connection, int id) {
        try {
            PreparedStatement deleteStatement =
connection.prepareStatement("DELETE FROM students WHERE id = ?");
            deleteStatement.setInt(1, id);
            deleteStatement.executeUpdate();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private static void calculateAverageGrade(Connection connection) {
        try {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT AVG(grade) AS
average_grade FROM students");

            while (resultSet.next()) {
                double averageGrade = resultSet.getDouble("average_grade");
                System.out.println("Average Grade: " + averageGrade);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ja-AiT

```

Please replace `"your_username"` and `"your_password"` with your actual MySQL credentials, and make sure your MySQL server is running and accessible. Also, adjust the data and table structure as needed for your specific use case.

Ja-AAiT
```