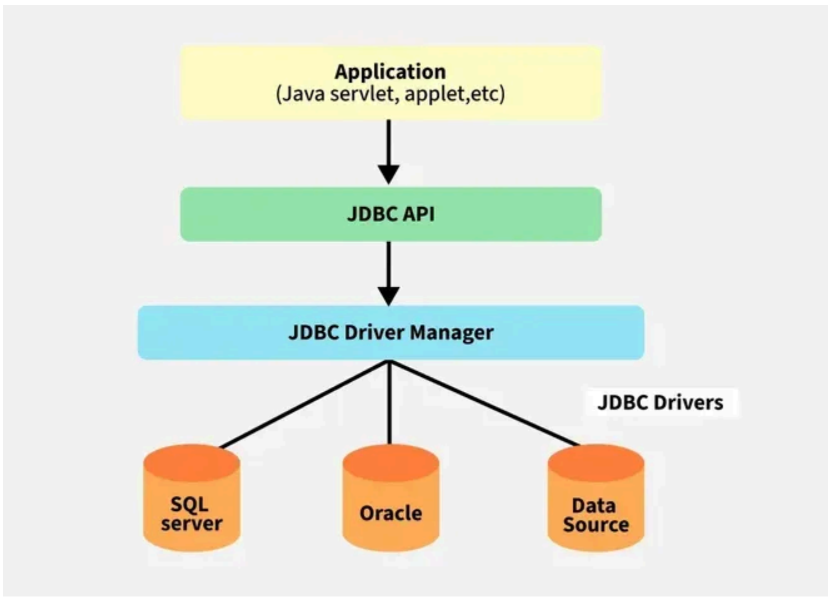
JDBC (Java Database Connectivity)

Last Updated : 18 Jan, 2025

JDBC (Java Database Connectivity) is an API in Java that enables applications to interact with databases. It allows **a** Java program to **connect to a database**, **execute queries,** and r**etrieve and manipulate data.** By providing a standard interface, JDBC ensures that Java applications can work with different relational databases like MySQL, Oracle, PostgreSQL, and more.

JDBC Architecture



Explanation:

- **Application:** It is a Java applet or a servlet that communicates with a data source.
- **The JDBC API:** It allows Java programs to execute SQL queries and retrieve results. Key interfaces include Driver, ResultSet, RowSet, PreparedStatement, and Connection. Important classes include DriverManager, Types, Blob, and Clob.

- **DriverManager:** It plays an important role in the JDBC architecture. It uses some database-specific drivers to effectively connect enterprise applications to databases.
- **JDBC drivers:** These drivers handle interactions between the application and the database.

The JDBC architecture consists of two-tier and three-tier processing models to access a database. They are as described below:

1. Two-Tier Architecture

A Java Application communicates directly with the database using a JDBC driver. Queries are sent to the database, and results are returned directly to the application. In a client/server setup, the user's machine (client) communicates with a remote database server.

Structure:

> *Client Application (Java) -> JDBC Driver -> Database*

2. Three-Tier Architecture

In this, user queries are sent to a middle-tier services, which interacts with the database. The database results are processed by the middle tier and then sent back to the user.

Structure:

> *Client Application -> Application Server -> JDBC Driver -> Database*

JDBC Components

There are generally **4 main components of JDBC** through which it can interact with a database. They are as mentioned below:

1. JDBC API

It provides various methods and interfaces for easy communication with the database. It includes two key packages

- **java.sql**: This package, is the part of **Java Standard Edition (Java SE) ,** which contains the core interfaces and classes for accessing and processing data in relational databases. It also provides essential functionalities like establishing connections, executing queries, and handling result sets

- **javax.sql**: This package is the part of **Java Enterprise Edition (Java EE) ,** which extends the capabilities of `java.sql` by offering additional features like connection pooling, statement pooling, and data source management.

It also provides a standard to connect a database to a client application.

2. JDBC Driver Manager

Driver manager is responsible for loading the correct database-specific driver to establish a connection with the database. It manages the available drivers and ensures the right one is used to process user requests and interact with the database.

3. JDBC Test Suite

It is used to test the operation(such as insertion, deletion, updating) being performed by JDBC Drivers.

4. JDBC Drivers

JDBC drivers are client-side adapters (installed on the client machine, not on the server) that convert requests from Java programs to a protocol that the DBMS can understand. There are 4 types of JDBC drivers:

1. Type-1 driver or JDBC-ODBC bridge driver
2. Type-2 driver or Native-API driver (partially java driver)
3. Type-3 driver or Network Protocol driver (fully java driver)

4. Type-4 driver or Thin driver (fully java driver) – It is deprecated and no longer supported since Java 8. Instead modern drivers like the Type – 4 driver are widely used.

JDBC Classes and Interfaces

Class/Interfaces	Description
DriverManager	Manages JDBC drivers and establishes database connections.
Connection	Represents a session with a specific database.
Statement	Used to execute static SQL queries.
PreparedStatement	Precompiled SQL statement, used for dynamic queries with parameters.
CallableStatement	Used to execute stored procedures in the database.
ResultSet	Represents the result set of a query, allowing navigation through the rows.
SQLException	Handles SQL-related exceptions during database operations.

Steps to Connect to MySQL Database Using JDBC

Step 1: Load the JDBC Driver

Class.forName(“com.mysql.cj.jdbc.Driver”);

Step 2: Establish a Connection

Connection connection = DriverManager.getConnection(

“jdbc:mysql://localhost:3306/your_database”,

“your_username”,

“your_password”

);

Step 3: Create a Statement

Statement statement = connection.createStatement();

Step 4: Execute a Query

String query = “INSERT INTO students (id, name) VALUES (101, ‘John Doe’)”;

int rowsAffected = statement.executeUpdate(query);

System.out.println(“Rows affected: ” + rowsAffected);

Step 5: Close the Connection

statement.close();

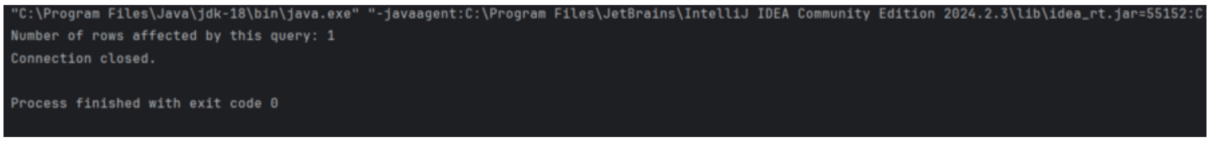
connection.close();

Create a Simple JDBC Application

The below Java program demonstrates *how to establish a MYSQL database connection using JDBC and execute a query.*

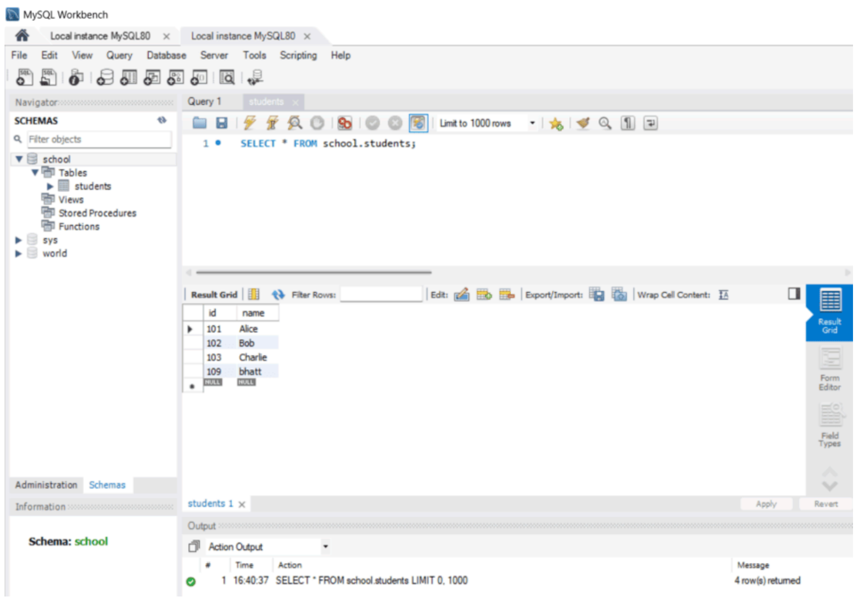
```java
// Java program to implement a simple JDBC application
import java.sql.*;

public class Geeks {
    public static void main(String[] args)
    {
        // Database URL, username, and password

        // Replace with your database name
        String url
            = "jdbc:mysql://localhost:3306/your_database";

        // Replace with your MySQL username
        String username = "your_username";

        // Replace with your MySQL password
        String password = "your_password";

        // Updated query syntax for modern databases
        String query
            = "INSERT INTO students (id, name) VALUES (109, 'bhatt')";

        // Establish JDBC Connection
        try {

            // Load Type-4 Driver
            // MySQL Type-4 driver class
            Class.forName("com.mysql.cj.jdbc.Driver");

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

            // Create a statement
            Statement st = c.createStatement();

            // Execute the query
            int count = st.executeUpdate(query);
            System.out.println(
                "Number of rows affected by this query: "
                + count);

            // Close the connection
            st.close();
            c.close();
            System.out.println("Connection closed.");
        }
        catch (ClassNotFoundException e) {
            System.err.println("JDBC Driver not found: "
                               + e.getMessage());
        }
        catch (SQLException e) {
            System.err.println("SQL Error: "
                               + e.getMessage());
        }
    }
}
```

Output:



```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.3\lib\idea_rt.jar=55152:C
Number of rows affected by this query: 1
Connection closed.

Process finished with exit code 0
```

Note: When the program runs successfully, a new record is added to the `students` table as shown below:



```
SELECT * FROM school.students;
```

id	name
101	Alice
102	Bob
103	Charlie
109	bhatt
NULL	NULL

#	Time	Action	Message
1	16:40:37	SELECT * FROM school.students LIMIT 0, 1000	4 row(s) returned

Key Features

- **Platform Independence**: It enables database operations across different platforms.
- **Standard API**: It provides a uniform interface for various databases.
- **Support for Multiple Databases**: It works with popular databases like MySQL, PostgreSQL, Oracle, etc.
- **Extensibility**: It offers features like batch processing, connection pooling, and transaction management.

Comment More info

Next Article