JAVA DATABASE CONNECTIVITY

Pruthvikanth AC August 16, 2023

Java Database Connectivity (JDBC):

Java programs can communicate with relational databases using the Java Database Connectivity (JDBC) API. It gives Java programs a standardized way to connect to databases, send SQL queries, and work with data. Without having to deal with the complexity of database-specific communication protocols, developers can design database-driven apps by using JDBC.

JDBC Drivers

JDBC drivers are software components that enable Java applications to connect to specific database management systems (DBMS). They serve as intermediaries between the Java application and the database, translating Java calls into the appropriate database-specific protocol.

There are four main types of JDBC drivers:

1. **Type 1 Driver (JDBC-ODBC Bridge Driver):**
    - This driver acts as a bridge between Java and databases that support ODBC (Open Database Connectivity).
    - Java application → Type 1 Driver → ODBC → Database
    - Requires the ODBC driver to be installed on the client machine.
    - Not commonly used due to its limitations and reliance on platform-specific ODBC drivers.
2. **Type 2 Driver (Native-API Driver):**
    - This driver uses native libraries provided by the database vendor to connect to the database.
    - Java application → Type 2 Driver → Native API → Database
    - Offers better performance compared to Type 1 but still has some platform dependency.
    - More efficient than Type 1 but not as portable.

3. **Type 3 Driver (Network Protocol Driver):**
    - This driver communicates with a middle-tier server that converts JDBC calls into the database-specific protocol.
    - Java application → Type 3 Driver → Middle-tier Server → Database
    - Platform-independent and provides better security and performance.
    - Requires the installation and configuration of the middle-tier server.
4. **Type 4 Driver (Thin Driver or Direct Protocol Driver):**
    - This driver communicates directly with the database using the native protocol of the DBMS.
    - Java application → Type 4 Driver → Database
    - Highly efficient, platform-independent, and commonly used for modern applications.
    - No need for a middle-tier server.

Using JDBC in Java:

Here's a step-by-step overview of how you might use JDBC to connect to a database using the Type 4 driver:

1. **Load the Driver:** Load the appropriate JDBC driver class using `Class.forName()` to register the driver with the DriverManager. Some modern JDBC drivers automatically register themselves, so you might not need this step.
2. **Establish Connection:** Construct a connection URL that specifies the database location, and then use `DriverManager.getConnection()` to establish a connection to the database.
3. **Execute SQL Statements:** Use the `Connection` object to create Statement or `PreparedStatement` objects and execute SQL queries or updates on the database.
4. **Process Results:** If executing a query, retrieve the results using the `ResultSet` object, and process the data as needed.
5. **Close Resources:** Properly close the `ResultSet`, `Statement`, and `Connection` objects to release resources and maintain database connections.

Sample code:

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

public class JdbcExample {
    public static void main(String[] args) {
        try {
            // Load the JDBC driver (may not be needed for some drivers)
            Class.forName("com.mysql.jdbc.Driver");

            // Establish a connection to the database
            String url = "jdbc:mysql://localhost:3306/mydb";
            String user = "username";
            String password = "password";
            Connection connection = DriverManager.getConnection(url, user, password)
    ;

            // Execute a query
            String query = "SELECT * FROM employees";
            PreparedStatement statement = connection.prepareStatement(query);
            ResultSet resultSet = statement.executeQuery();

            // Process the results
            while (resultSet.next()) {
                String employeeName = resultSet.getString("name");
                int employeeId = resultSet.getInt("id");
                System.out.println("ID: " + employeeId + ", Name: " + employeeName);
            }

            // Close resources
            resultSet.close();
            statement.close();
            connection.close();
        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
        }
    }
}
```