Database Connectivity with Java

Prativa Nyaupane

Recap

- JDBC Architecture

Outline

- JDBC Architecture
- JDBC Driver Types and Configuration
- Managing Connections and Statements
- Result Sets and Exception Handling
- DDL and DML Operations
- SQL Injection and Prepared Statements
- Row Sets and Transactions
- SQL Escapes

JDBC Driver Types

- JDBC driver implementations differ because they support various operating systems and hardware platforms running on Java.
- The implementation types are distributed in four categories:
  - Type 1 Driver/ JDBC-ODBC Bridge Driver
  - Type 2 Driver/ JDBC-Native API
  - Type 3 Driver/ JDBC-Net pure Java
  - Type 4 Driver/ 100% Pure Java

Type 1 Driver/JDBC-ODBC Bridge Driver

- A type 1 driver translates JDBC to ODBC(Open Database Connectivity) and relies on an ODBC driver to communicate with the database. ODBC is a standard API for accessing DBMS.
- Requires installation and configuration of the ODBC driver.

- While JDBC is specific to the Java programming language, ODBC is not tied to any specific programming language. It was originally developed by Microsoft and is widely used on Windows platform.
- Java 8 no longer provide the JDBC/ODBC bridge.

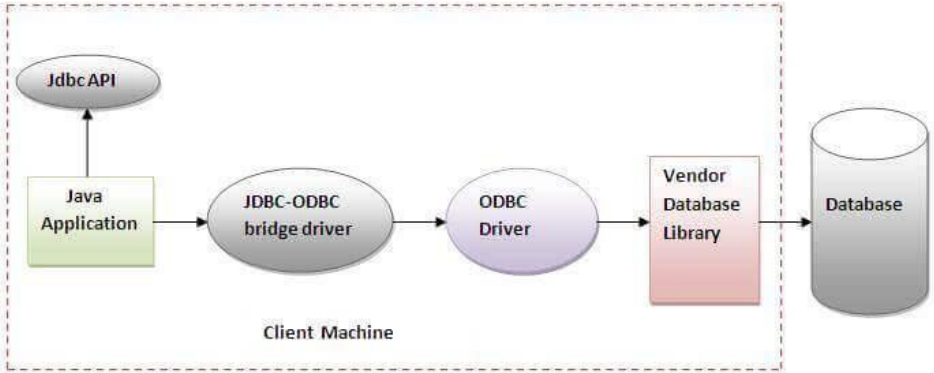

Figure- JDBC-ODBC Bridge Driver

Type 2 Driver/ JDBC-Native API

- A type 2 driver is written partly in Java and partly in native code(C/C++), it communicates with the client API of a database.
- The driver converts JDBC method calls into native calls of the database API.
- Faster than Type 1 drivers as it eliminates the overhead of ODBC.
- Requires installation and configuration of native binary code.

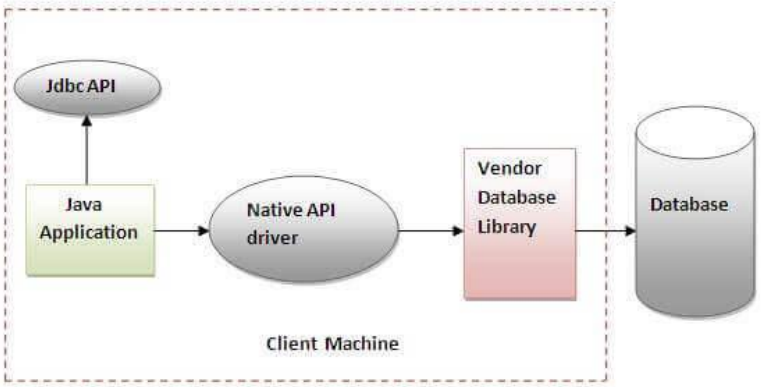

Figure- Native API Driver

Type 3 Driver/JDBC-Net Pure Java

- A type 3 driver is a pure Java client library that uses a database-independent protocol to communicate database requests to a server component, which then translates the request into a database-specific protocol.

- This simplifies deployment because the platform-specific code is located only on the server.
- No client installation is required.
- Server translates the protocol to DBMS function calls.

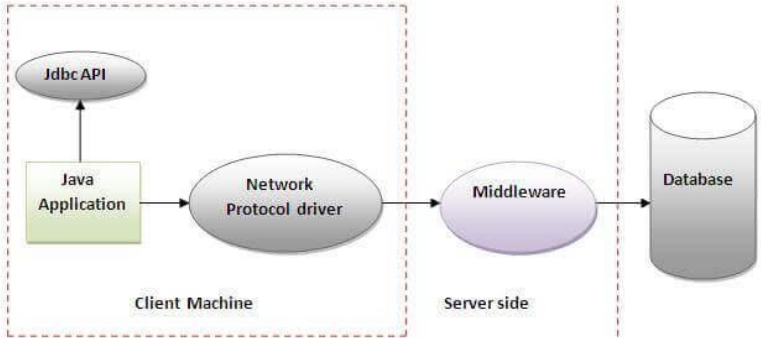

Figure- Network Protocol Driver

Type 4 Driver/100% Pure Java

- A pure java-based driver communicates directly with the vendor's database in the Type 4 driver through socket connection.
- It translates JDBC request directly to vendor-specific database protocol.

- It is a pure Java driver, no client installation or configuration is required.
- MySQL's connector/J driver is a Type 4 driver.

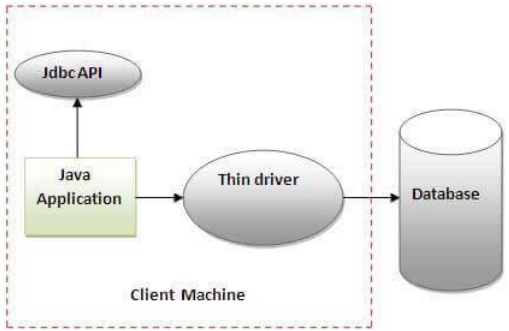

Figure- Thin Driver

Selection of Driver

- If we are accessing Oracle, SQL Server or MySQL, the preferred driver is type 4.
- If multiple types of databases are accessed at the same time by the Java application, then Type 3 is the preferred driver.
- If Type 3 or Type 4 drivers are not available for the database, then Type 2 drivers are useful in this situation.
- The Type 1 driver is used for development and testing purpose only. It is not considered to be deployment-level driver.

Intermediate Database Access Server

- An Intermediate Database Access Server is a server application that provides an interface between Java applications and multiple databases.
- It acts as a middle layer, handling database connection management, query execution, and result retrieval.
- The server simplifies database access by providing a unified API for connecting to and interacting with various databases.

Intermediate Database Access Server (contd...)

- The server consolidates database connections, manages pooling and caching, and provides load balancing and failover capabilities.
- It allows applications to connect to multiple databases using a single connection and provides an interface for database access.

JDBC API

- The JDBC API (Application Programming Interface) is a set of Java classes and interfaces that define the standard methods and behaviors for database connectivity.
- It allows Java applications to execute SQL statements, retrieve and manipulate data, and manage database connections.
- The JDBC API provides a common interface regardless of the specific database being used, enabling portability and flexibility.

Managing Connections and Statements

- Connection Management:
  - JDBC connections represent a connection to a database.
  - Use DriverManager to establish a connection.
  - Proper connection handling is crucial to avoid resource leaks.
  - Connections should be closed using the close() method.
- Statement Management:
  - Statements are used to execute SQL queries.
  - Types of statements include Statement, PreparedStatement, and CallableStatement.
  - Statement is a general-purpose interface.
  - PreparedStatement is precompiled and allows parameterized queries.
  - CallableStatement is used for calling stored procedures.

Result Sets and Exception Handling

- In JDBC, a ResultSet represents the data retrieved from a database after executing a SQL query.
- It provides methods to navigate, retrieve, and update data.
- Types of result sets include TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE.
- JDBC operations can throw exceptions, such as SQLException, which must be handled.
- Common exceptions include SQL syntax errors, connection issues, and data type mismatches.
- Proper exception handling ensures graceful error recovery and logging for troubleshooting.

Making a JDBC Application

- Making a JDBC application involves writing Java code to connect to a database, execute SQL queries or updates, and process the results.
- It typically includes steps such as loading the appropriate database driver, establishing a connection to the database, creating and executing SQL statements, and handling the query results or updates.

Example JDBC Application

```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/java_jdbc", "root", "root");

Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);

ResultSet rs = stmt.executeQuery("SELECT * FROM Persons");

System.out.println(rs.absolute(1));
```

Importing the Mysql JDBC driver using Reflection in Java

Creating Database Connection

Creating a Statement to execute queries

Execute the statement

GET first row