JDBC

JDBC stands for Java Database Connectivity.

JDBC is a Java API to connect and execute the query with the database.

It is a part of JavaSE (Java Standard Edition).

JDBC API uses JDBC drivers to connect with the database.

JDBC Drivers

1) JDBC-ODBC bridge driver:

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.

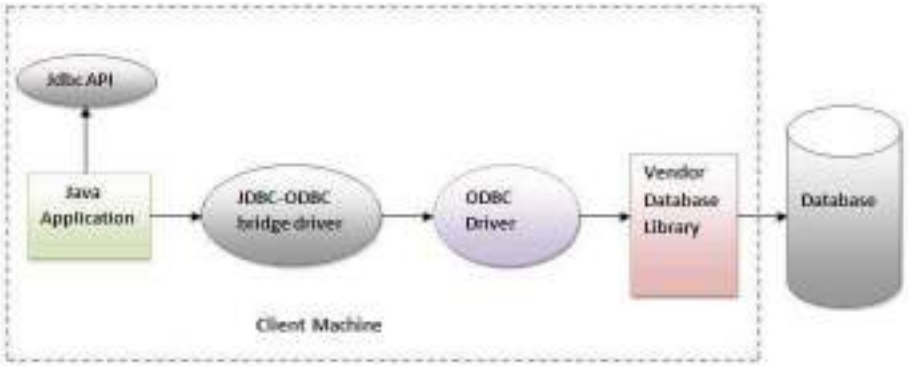

Figure- JDBC-ODBC Bridge Driver

In Java 8, the JDBC-ODBC Bridge has been removed.

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages: 1) easy to use. 2) can be easily connected to any database.

Disadvantages: 1) Performance degraded because JDBC method call is converted into the ODBC function calls. 2)The ODBC driver needs to be installed on the client machine.

2) Native-API driver: The Native API driver uses the client-side libraries of the database. The driver convertsJDBC method calls into native calls of the database API.
It is not written entirely in java.

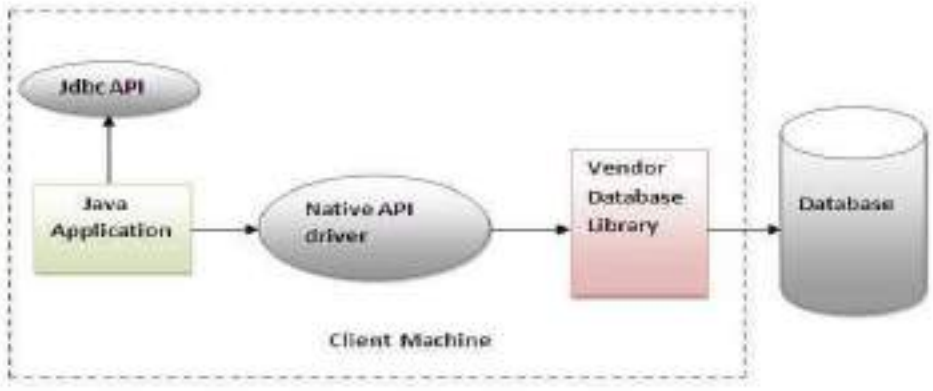

Figure- Native API Driver

Advantage:performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:1)The Native driver needs to be installed on the each client machine.

2)The Vendor client library needs to be installed on client machine.

3) Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

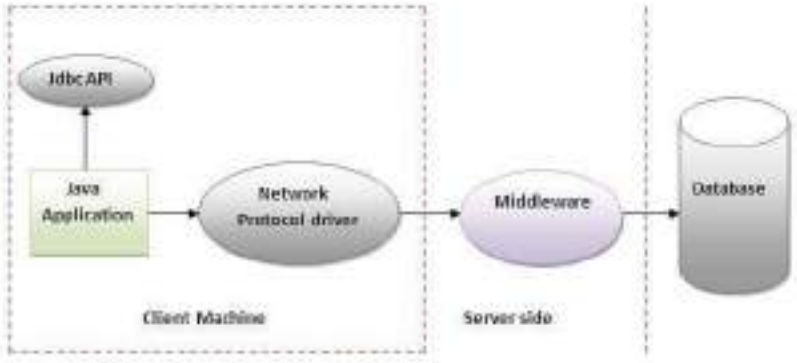

Figure- Network Protocol Driver

Advantage:No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

4) Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

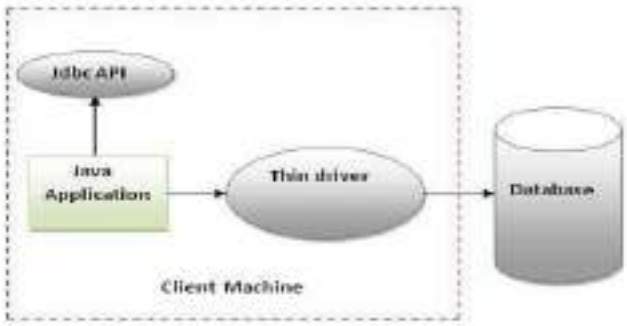

Figure- Thin Driver

Advantage: 1) Better performance than all other drivers.

2) No software is required at client side or server side.

Disadvantage:Drivers depend on the Database.

Connection Interface

A Connection is the session between java application and database.

The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData.

The Connection interface provide many methods for transaction management like commit(), rollback() etc.

Commonly used methods of Connection interface:

1) public StatementcreateStatement(): creates a statement object that can be used to execute SQL queries.

2) public voidsetAutoCommit(boolean status): is used to set the commit status.By default it is true.

3) public voidcommit(): saves the changes made since the previous commit/rollback permanent.

4) public voidrollback(): Drops all changes made since the previous commit/rollback.

5) public voidclose(): closes the connection and Releases a JDBC resources immediately.

Connecting to MYSQL Database:

```
Connection con;
Class.forName("com.mysql.jdbc.Driver");
        (Or)
DriverManager.registerDriver( newcom.mysql.jdbc.Driver());
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/kmit","root","shiva");
```

Connecting to Oracle Database:

```
Connection con;
Class.forName("oracle.jdbc.driver.OracleDriver");
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
```

Statement interface

The Statement interface provides methods to execute queries with the database.

The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Commonly used methods of Statement interface:

1) public ResultSetexecuteQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public intexecuteUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public booleanexecute(String sql): is used to execute queries that may return multiple results.

Types of Statement interfaces:

1. **Statement**.
2. **PreparedStatement**: Helps to pass runtime parameters to the SQL Query.
3. **CallableStatement**: Helps us to invoke stored procedures/functions from java application.

How to get Statement interface object:

Statement st= con.prepareStatement();

How to get PrepareStatement interface object:

PreparedStatementpst;

pst = con.prepareStatement("SQL Query with Runtime parameters");

Example:

Q)Write an sql query that returns the returns all the records from employee table whose employee ID is greater than the runtime values supplied.

Answer:

PreparedStatementpst = con.prepareStatement("select * from emp where empid>?");

pst.setInt(1,id); // where id is value of employee id given by user at runtime

ResultSetrs=pst.executeQuery(); // now rsconatins all the required records.

Database Queries

Inserting data into a Database Table:

int i=st.executeUpdate("insert into student(rollno,name) values('17BD1A1235','SHIVA')");

Updating data of the Database Table:

int i=st.executeUpdate("update emp set salary=salary+5000 where empid='263'");

Deleting data from the Database Table

int i=st.executeUpdate("deletefrom emp where empid>'263'");

Retrieving records from the Database Table:

```
ResultSetrs=st.executeQuery("select * from student");


while(rs.next())
{
System.out.print(rs.getInt("rno")+"    ");
System.out.print(rs.getString(“name”)+"  ");
System.out.println(rs.getString(“grade”)+"  ");
}
```