DB connectivity

1) JDBC
2) Drivers
3) Important classes
4) Steps to connect to DB

What is JDBC?

JDBC is an acronym for Java Database Connectivity. It's an advancement for ODBC (Open Database Connectivity). JDBC is a standard API specification developed in order to move data from frontend to the backend. This API consists of classes and interfaces written in Java. It basically acts as an interface (not the one we use in Java) or channel between your Java program and databases i.e it establishes a link between the two so that a programmer could send data from Java code and store it in the database for future use.

Drivers

JDBC Driver is a software component that enables java application to interact with the database

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. This is now discouraged because of thin driver.

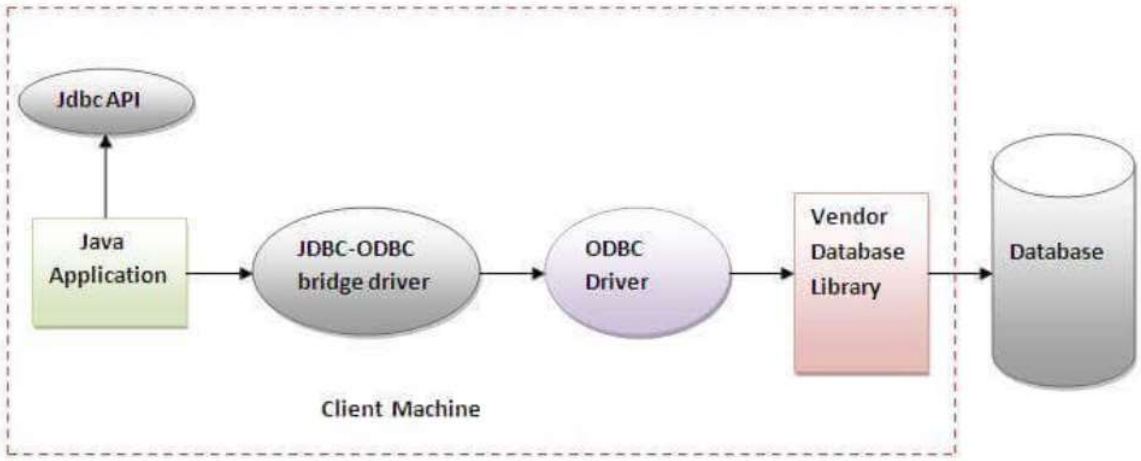


Figure- JDBC-ODBC Bridge Driver

2. **Native Driver**
   The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.
   Advantage:

performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

The Native driver needs to be installed on the each client machine.

3. The Vendor client library needs to be installed on client machine.

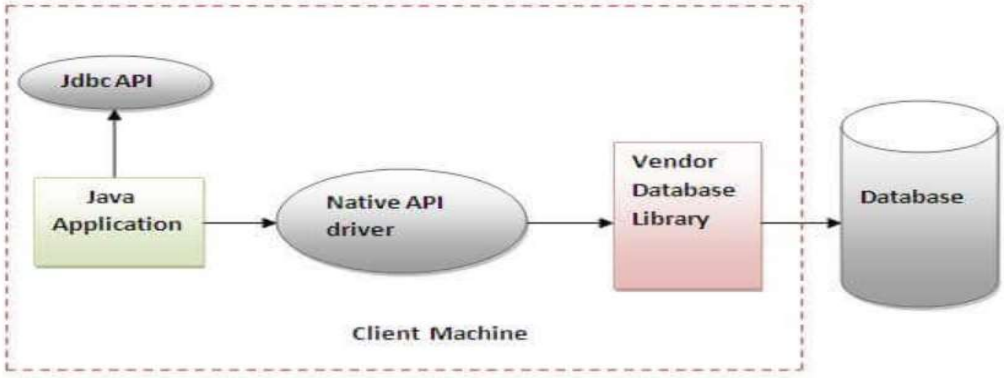


Figure- Native API Driver

4. **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.

Advantage:

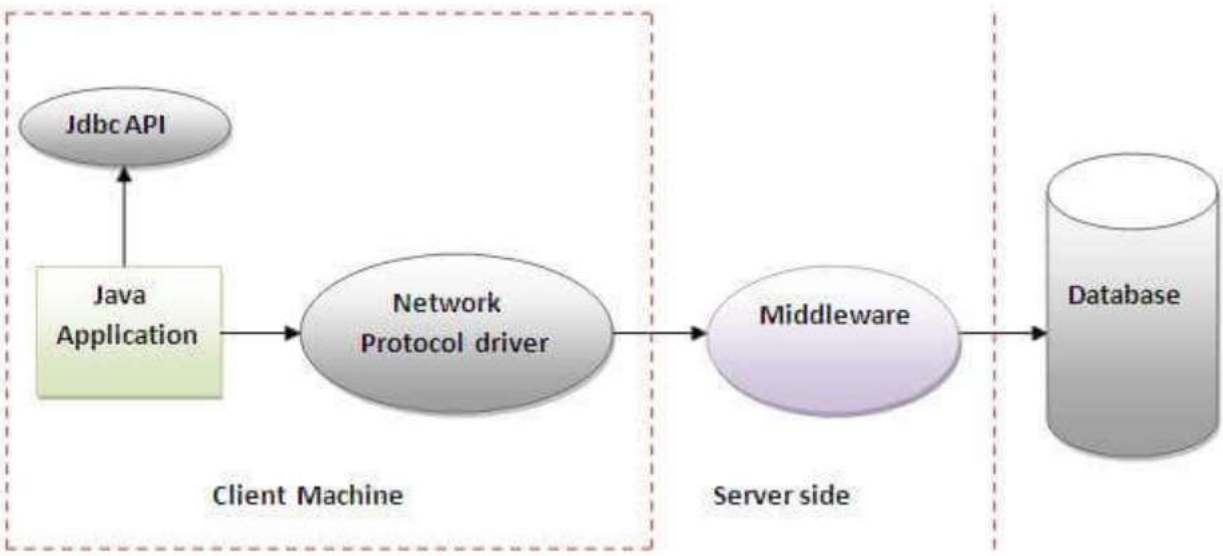
No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

Network support is required on client machine.

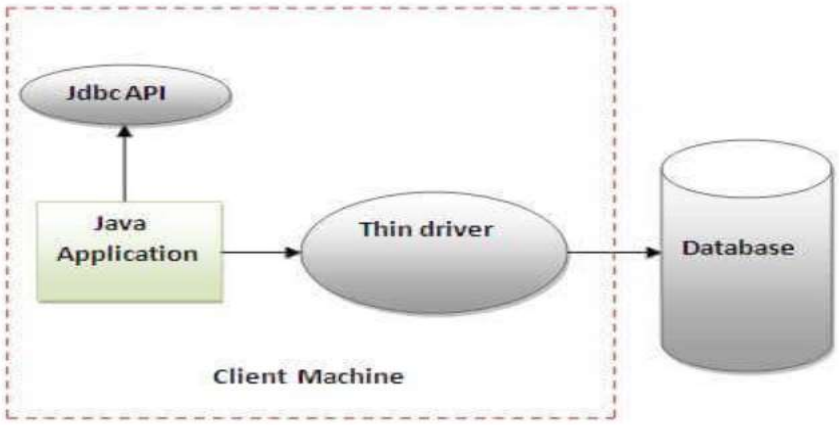
Requires database-specific coding to be done in the middle tier.

Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.



5. **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.



Advantage:
Better performance than all other drivers.
No software is required at client side or server side.

Disadvantage:
Drivers depend on the Database.

A list of popular interfaces of JDBC API are given below:

Driver interface
Connection interface
Statement interface
PreparedStatement interface
CallableStatement interface
ResultSet interface
ResultSetMetaData interface : To get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData
DatabaseMetaData interface
RowSet interface

A list of popular classes of JDBC API are given below:

DriverManager class
Blob class
Clob class
Types class

DB Connectivity Steps

1. Import the Packages
2. Load the drivers using the *forName() method*
3. Register the drivers *using DriverManager*
4. Establish a connection *using the Connection class object*
5. Create a statement
6. Execute the query
7. Close the connections

1) Import Packages

2) Loading Drivers :

load the driver's class file into memory at the runtime. No need of using new or create objects

```
Class.forName()
```

Example : Class.forName(“oracle.jdbc.driver.OracleDriver”);

3) Register Driver

DriverManager is a Java inbuilt class with a static member register. Here we call the constructor of the driver class at compile time

DriverManager.registerDriver()

Example : DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

4) Establish Connections

Connection con = DriverManager.getConnection(url,user,password)

getConnection Methods
1) public static Connection getConnection(String url)throws SQLException
2) public static Connection getConnection(String url,String name,String password) throws SQLException

user: Username from which your SQL command prompt can be accessed.
password: password from which the SQL command prompt can be accessed.
con: It is a reference to the Connection interface.
Url: Uniform Resource Locator which is created as shown below:

String url = “ jdbc:oracle:thin:@localhost:1521:xe”

5) Create a Statement

interact with the database

1. JDBCStatement,
2. CallableStatement
3. PreparedStatement

JDBC Statement
The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet

Example : `Statement st = con.createStatement();`

`Execution methods e`

`1) public ResultSet executeQuery(String sql): is used to execute SELECT`

```
query. It returns the object of ResultSet.

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

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

4) public int[] executeBatch(): is used to execute batch of commands.
```

Callable Statement

CallableStatement interface is used to call the stored procedures and functions.

```
CallableStatement stmt=con.prepareCall("{call insertR(?,?)}");
stmt.setInt(1,1011);
stmt.setString(2,"Amit");
stmt.execute();
```

Prepared Statement

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

Example : String sql="insert into emp values(?,?,?)";

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

update / delete Example

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

PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");

stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");

int i=stmt.executeUpdate();
```

Select example

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
```

```
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

6) Execute the query

public ResultSet executeQuery(String sql)throws SQLException

```
ResultSet rs=stmt.executeQuery("select * from emp");

while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

MySQL Connectivity Example

```
import java.sql.*;

class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

https://sqliteonline.com/

https://sqliteonline.com/#sharedb=s%3Aters:ter