Chapter 5

Database Connectivity(12M)

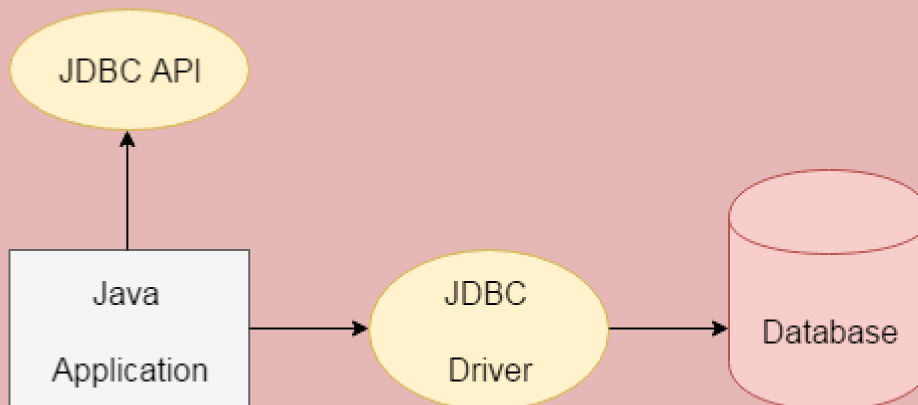
JDBC

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. There are **four types of JDBC drivers:**

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The **java.sql** package contains classes and interfaces for JDBC API. 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
- DatabaseMetaData interface
- RowSet interface

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

- DriverManager class
- Blob class
- Clob class
- Types class

Why Should We Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses **ODBC driver which is written in C language** (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses **JDBC drivers (written in Java language).**

We can use JDBC API to handle database using Java program and can perform the following activities:

1. **Connect to the database**
2. **Execute queries and update statements to the database**
3. **Retrieve the result received from the database.**

There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver(type 1)
2. Native-API driver (partially java driver) (type 2)
3. Network Protocol driver (fully java driver) (type 3)
4. Thin driver (fully java driver) (type 4)

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.

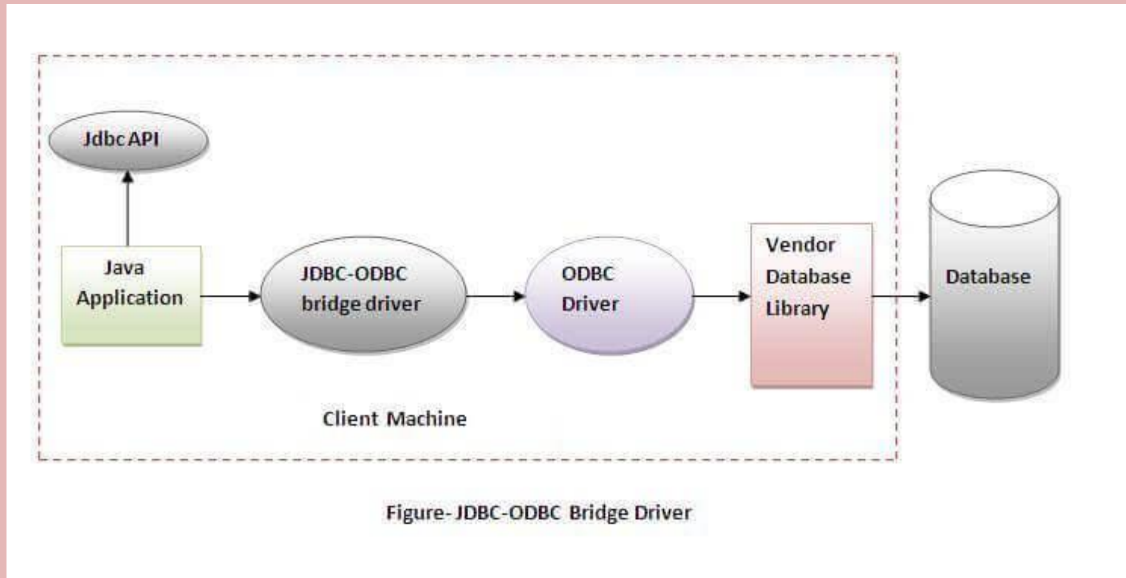


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- 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 converts **JDBC method calls into native calls of the database API**. It is not written **entirely in java**.

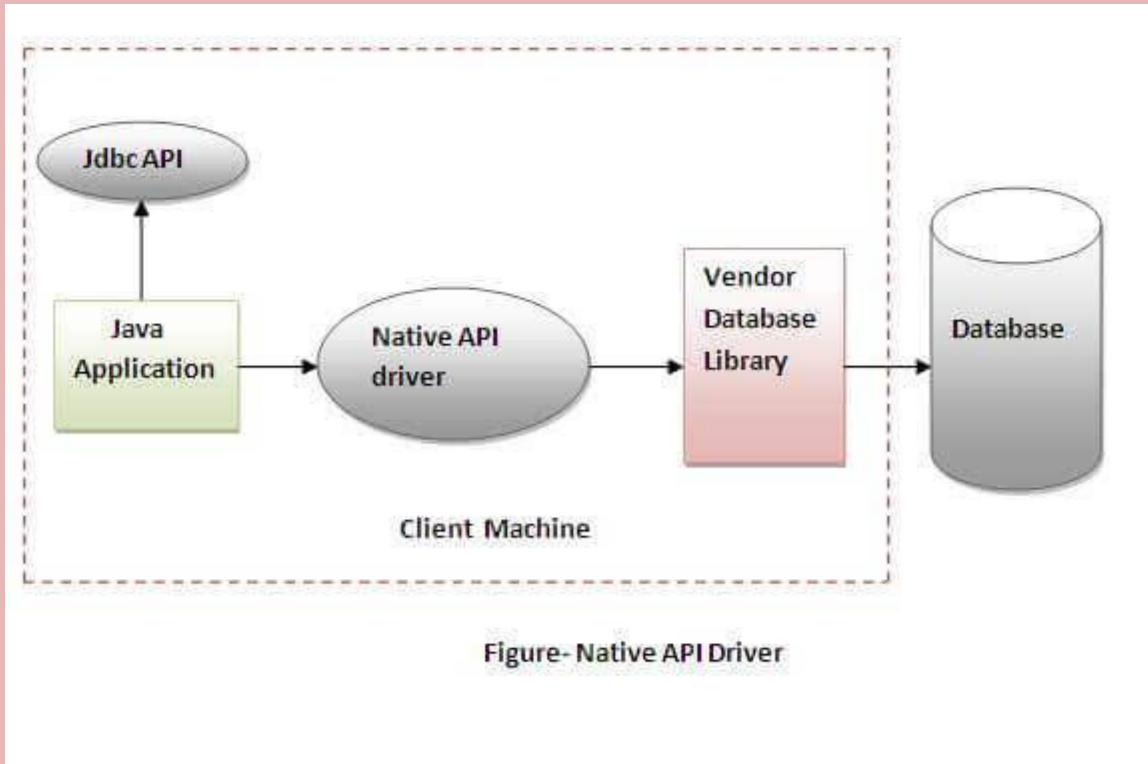


Figure- Native API Driver

Advantage:

- performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

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

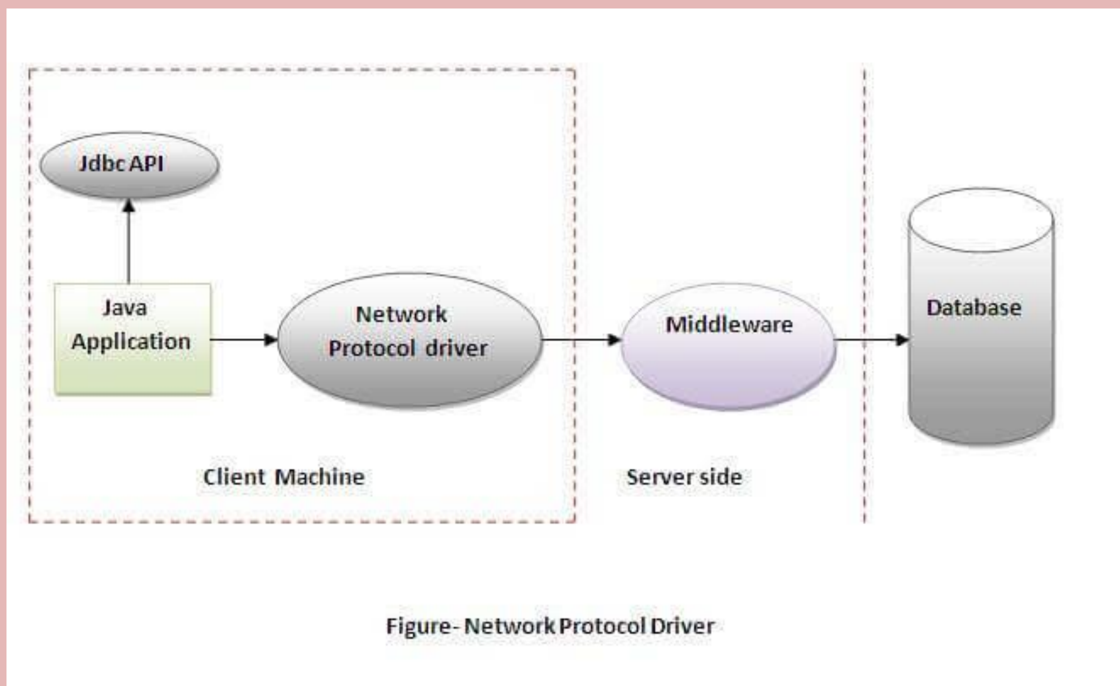


Figure- Network Protocol Driver

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.**

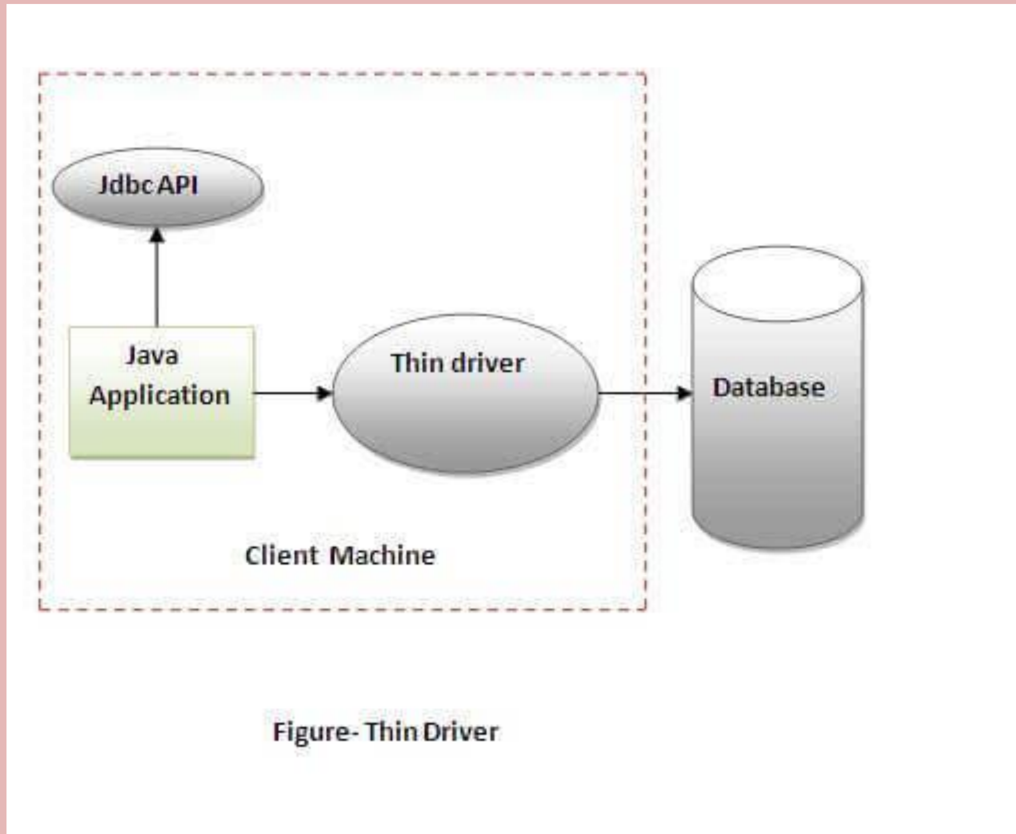


Figure- Thin Driver

Advantage:

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

Disadvantage:

- Drivers depend on the Database.

Java Database Connectivity with 5 Steps

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

1) Register the driver class

The **forName()** method of Class class **is used to register the driver class**. This method is used to dynamically load the driver class.

Syntax of forName() method

1. public static void forName(String className)throws ClassNotFoundException

for eg.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

2) Create the connection object

The **getConnection()** method of DriverManager class is used **to establish connection** with the database.

Syntax of getConnection() method

- public static Connection getConnection(String url,String name,String password)
throws SQLException

```
Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/emp","root","''");
```

3) Create the Statement object

The **createStatement() method of Connection interface is used to create statement**. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

1. public Statement createStatement()throws SQLException

Example to create the statement object

1. Statement stmt=con.createStatement();

4) Execute the query

The **executeQuery()** method of Statement interface is used to execute queries(**Select * from tablename**) to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

The **executeUpdate()** is used to execute DML commands(**create, drop, insert, update, delete**).

Syntax of executeQuery() method

```
1.  public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
1.  ResultSet rs=stmt.executeQuery("select * from emp");
2.
3.  while(rs.next()){
4.  System.out.println(rs.getInt(1)+" "+rs.getString(2));
5.  }
```

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. **The close() method of Connection interface is used to close the connection.**

Syntax of close() method

```
1.      public void close()throws SQLException
```

For eg.

```
con.close();
```

Program for Database Connectivity

```
import java.sql.*;

class Myconnectivity

{

public static void main(String args[])

{

try{

Class.forName("com.mysql.jdbc.Driver");

Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/emp","root","");

//here emp is database name, root is username and blank password

Statement stmt=con.createStatement();


String i="insert into emptable values(3,'abd',100,'Teacher')";

stmt.executeUpdate(i);


String d="delete from emptable where id=3";

stmt.executeUpdate(d);


String u="Update emptable set name='rohan' where id=2";

stmt.executeUpdate(u);
```

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

while(rs.next())

System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getInt(3)+"  "+rs.getString(4));


con.close();

}catch(Exception e){ System.out.println(e);}

}

}
```

DriverManager class

The **DriverManager class acts as an interface between user and drivers**. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The **DriverManager class maintains a list of Driver classes** that have registered themselves by calling the method DriverManager.registerDriver().

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.

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.

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

Commonly used methods of ResultSet interface

1) public boolean **next():**	is used to move the cursor to the one row next from the current position.
2) public boolean **previous()**:	is used to move the cursor to the one row previous from the current position.
3) public boolean **first()**:	is used to move the cursor to the first row in result set object.
4) public boolean **last()**:	is used to move the cursor to the last row in result set object.
5) public boolean **absolute**(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean **relative**(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.

PreparedStatement interface

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

Let's see the example of parameterized query:

1. String sql="insert into emp values(?,?,?)";

As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

```
import java.sql.*;

public class Pstatement

{

public static void main(String args[])
```

```
{
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("jdbc:odbc:data");
Statement st=con.createStatement();

String a="Update emp set name=? where id=?";

PreparedStatement ps=con.prepareStatement(a);
ps.setString(1,"shyam");
ps.setInt(2,2);
ps.executeUpdate();

ps.setString(1,"Rohan");
ps.setInt(2,3);
ps.executeUpdate();


ResultSet rs=st.executeQuery("select * from emp");
System.out.println("  id"+"  salary"+"  name");

while(rs.next())
```

```
{
System.out.println("  "+rs.getInt("id")+
"   "+rs.getString("salary")+"     "+rs.getString("name1"));
}
con.close();
}
catch(SQLException e)
{ }
catch(Exception e)
{
}
}}
```

Why use PreparedStatement?

Improves performance: The **performance of the application will be faster** if you use PreparedStatement interface because **query is compiled only once**.

How to get the instance of PreparedStatement?

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement. Syntax:

1. public PreparedStatement prepareStatement(String query)throws SQLException{}

Methods of PreparedStatement interface

The important methods of PreparedStatement interface are given below:

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.

ResultSetMetaData Interface

The metadata means data about data i.e. we can get further information from the data.

If you have to get metadata of a table **like total number of column, column name, column type** etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

DatabaseMetaData interface

DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

Java CallableStatement Interface

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

We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.