Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) is an application programming interface (API) that helps Java program to communicate with databases and manipulates their data. The JDBC API provides the methods that can be used to send SQL and PL/SQL statements to almost any relational database. The latest version of JDBC is 4.2 that comes along with Java SE 8.

Purpose of JDBC API

- To access tables and its data from relation database.
- To send queries and update statement to database.
- Obtain and modify the results to and from a JDBC application.
- Find the metadata of the table.
- Performing different operations on a database, like creating table, querying data, updating data, inserting data from a Java application.

Architecture of JDBC

The JDBC API supports both two-tier and three- tier architecture for database access.

Two-tier Architecture

Two-tier Architecture provides direct communication between Java applications to the database. It requires a **JDBC driver** that can help to communicate with the particular database.

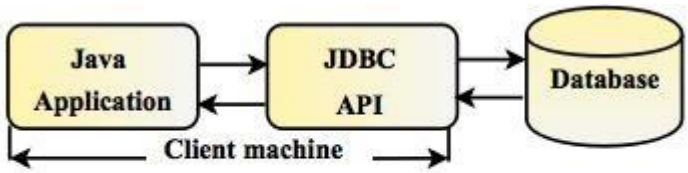


Fig: Two-tier Architecture of JDBC

Three-tier Architecture

In the three-tier model, commands are sent by the HTML browser to middle services i.e. Java application which can send the commands to the particular database. The middle tier has been written in **C or C++**languages. It can also provide better performance.

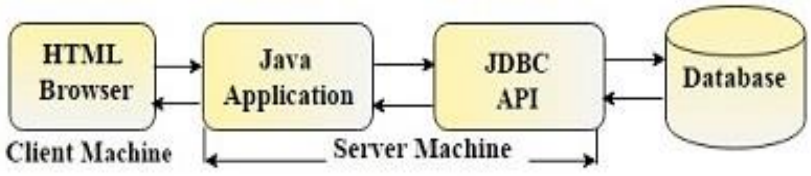


Fig: Three-tier Architecture of JDBC

JDBC Components

Following are the components of JDBC that help Java application connect with database.

JDBC API

The JDBC provides the various methods and interface for easy communication between Java application and database.

DriverManager
The DriverManager is a class that manages all database drivers. It loads the specific database drivers in an application to establish connection with database.

Connection
Connection is an interface that contains all methods for contacting with database.

JDBC Test Suite
The operation of every driver is different in Java applications. The JDBC test suit helps us to test the operation being performed by the JDBC drivers.

JDBC Driver
A JDBC driver is set of software components that help a Java application to interact with database. The JDBC driver implements lots of JDBC classes and interfaces that enable to open connection and interact with database server.
JDBC-ODBC Bridge
JDBC-ODBC Bridge provides a interface that helps to connect database drivers to the database.

JDBC Drivers Types

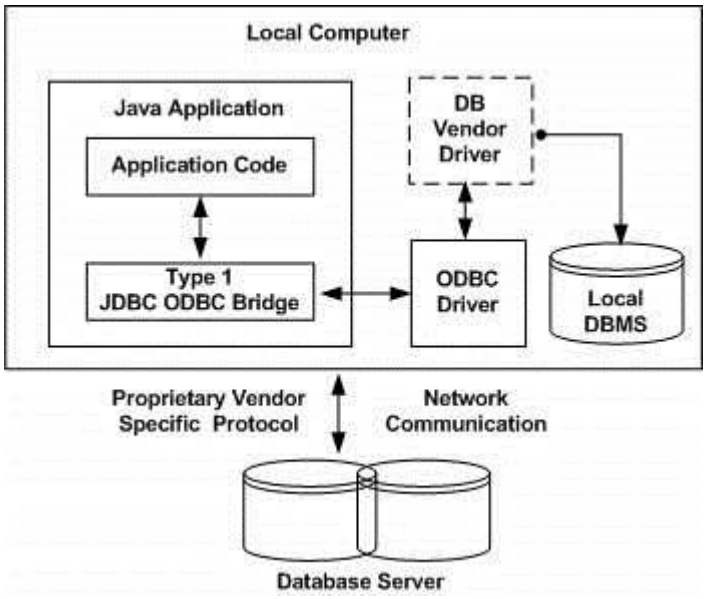
There are 4 types of JDBC drivers:
Type 1 : JDBC-ODBC bridge driver
Type 2 : Native API driver (Partial Java driver)
Type 3 : Network Protocol driver (Pure Java driver for database middleware)
Type 4 : Thin driver (Pure Java driver)

JDBC driver implementations vary because of the wide variety of operating systems and hardware platforms in which Java operates. Sun has divided the implementation types into four categories, Types 1, 2, 3, and 4, which is explained below −

Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

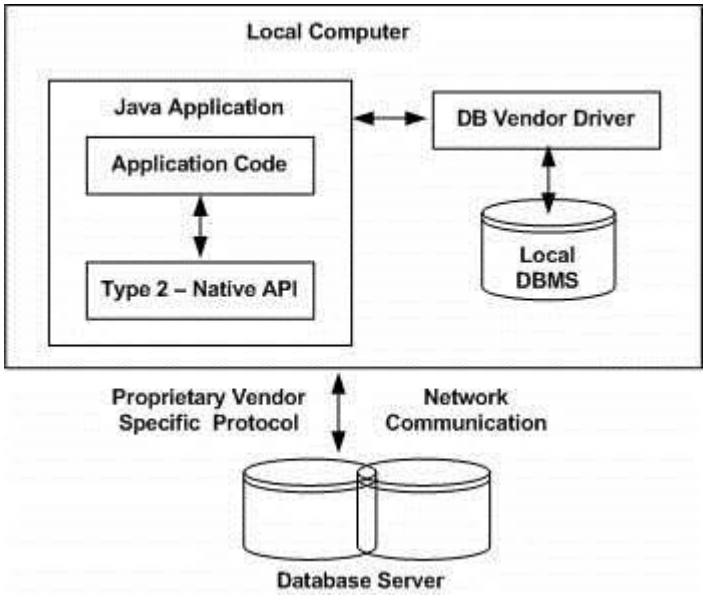


The JDBC-ODBC Bridge that comes with JDK 1.2 is a good example of this kind of driver.

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

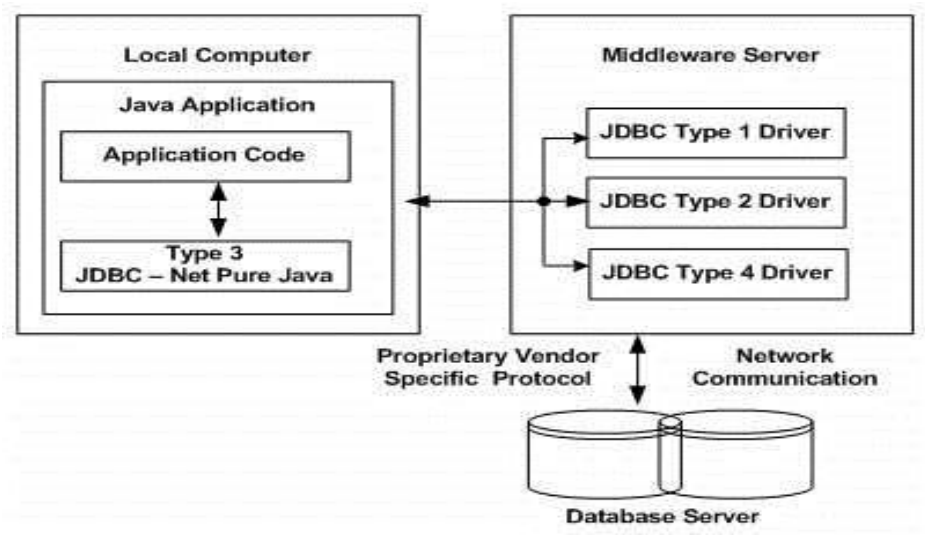


The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



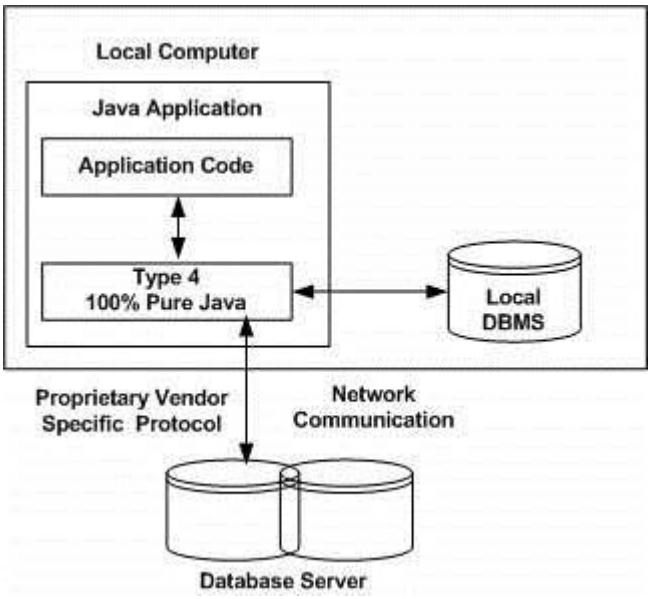
You can think of the application server as a JDBC "proxy," meaning that it makes calls for the client application. As a result, you need some knowledge of the application server's configuration in order to effectively use this driver type.

Your application server might use a Type 1, 2, or 4 driver to communicate with the database, understanding the nuances will prove helpful.

Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



MySQL's Connector/J driver is a Type 4 driver. Because of the proprietary nature of their network protocols, database vendors usually supply type 4 drivers.

Which Driver should be Used?

If you are accessing one type of database, such as Oracle, Sybase, or IBM, the preferred driver type is 4.

If your Java application is accessing multiple types of databases at the same time, type 3 is the preferred driver.

Type 2 drivers are useful in situations, where a type 3 or type 4 driver is not available yet for your database.

The type 1 driver is not considered a deployment-level driver, and is typically used for development and testing purposes only.

Steps to Connect a Java Application to Database
There are following five steps to create the database connection with Java application:

1. Register the Driver
2. Create Connection
3. Create SQL Statement
4. Execute SQL Queries
5. Close the Connection

1. Register the driver
The **Class.forName()** method is used to register the driver class dynamically.

For example:
```
Class.forName("oracle.jdbc.odbc.JdbcOdbcDriver");
```

2. Create the Connection Object

The **DriverManager** class provides the **getConnection()** method to establish connection object. It requires to pass a database url, username and password.

Syntax

```
getConnection(String url);
getConnection(String url, String username, String password);
getConnection(String url, Properties Info);
```

Example : Creating connection with oracle driver

```
Connection con = DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:XE","username","password");
```

3. Create SQL Statement

The **Connection interface** provides the **createStatement()** method to create SQL statement.

Syntax:

```
public Statement createStatement( ) throws SQLException
```

Example:

```
Statement stmt = con.createStatement();
```

4. Execute SQL Queries

The **Statement interface** provides the **executeQuery( )** method to execute SQL statements.

Syntax:

```
public ResultSet executeQuery(String sql) throw SQLException
```

Example

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

5. Closing the Connection

The **Connection interface** provides **close( )** method, used to close the connection. It is invoked to release the session after execution of SQL statement.

Syntax:

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

Example:

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

Note: We will discuss whole program in JDBC using oracle database with type 4 (Thin) driver.

Example : Connect the Java application with Oracle database

```
import java.sql.*;
class JDBCDemo
{
    public static void main(String args[])
    {
        try
        {
            //Load the driver
            Class.forName("oracle.jdbc.driver.OracleDriver");

            //Cretae the connection object
            Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott", "tiger");

            //Create the Statement Object
            Statement stmt = con.createStatement();

            //Excute the SQL query
            ResultSet rs = stmt.executeQuery("Select * from students");
            while (rs.next())
            {
                System.out.println (rs.getInt(1)+" "+rs.getString(2)+" "+rs.getFloat(3));
            }
            //Closing the connection object
            con.close();
            stmt.close();
            rs.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

DriverManager Class

The **DriverManager class** is responsible for managing the basic service to set of JDBC drivers. It acts as an interface between Java application and drivers. The DriverManager class will attempt to load the driver classes referenced in **"jdbc.drivers"** system property.

The DriverManager class loads the JDBC drivers to the system property.

DriverManager Class Methods

Methods	Description
getDriver(String url)	Helps to locate a driver that understands the given URL.
registerDriver(Driver driver)	Used to register the given driver with the DriverManager class.
static void deregisterDriver(Driver driver)	Removes the specified driver from the DriverManager class.
static Connection getConnection(String url)	It creates the connection with the given database URL.
static Connection getConnection(String url, String username, String password)	Establishes the connection with given database URL with username and password.

Statement Interface in JDBC

Statement Interface

The **Statement interface** provides the method to execute the database queries. After making a connection, Java application can interact with database. The Statement interface contains the ResultSet object.

Statement Interface Methods

The Statement interface provides the following important methods:

Sr No	Method Name	Description
1	public boolean execute(String sql)	It executes the given SQL query, which may return multiple results.
2	public intexecuteBatch( )	It submits the batch of commands to the database and returns an array of update counts.
3	public ResultSetexecuteQuery( )	Executes the given SQL queries which return the single ResultSet object.
4	publicintexecuteUpdate(String sql)	It performs the execution of DDL (insert, update or delete) statements.
5	public Connection getConnection( )	It retrieves the connection object that produced the statement object.

Example : Performing select operation with Statement Interface

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

```
import java.sql.Statement;
public class SelectTest
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("oracle.jdbc.OracleDriver");
        Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
        Statement st = con.createStatement();
        ResultSetrs = st.executeQuery("select * from student");
        while(rs.next()!=false)
        {
            System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+"
"+rs.getString(4));
        }
        rs.close();
        st.close();
        con.close();
    }
}
```

PreparedStatement Interface

- The **PreparedStatement interface** extends the Statement interface. It represents precompiled SQL statements and stores it in a PreparedStatement object.
- It increases the performance of the application because the query is compiled only once.
- The **PreparedStatement** is easy to reuse with new parameters.

Creating PreparedStatement Object

```
String sql = "Select * from Student where rollNo= ?";
PreparedStatementps = con.prepareStatement(sql);
```

Note: All the parameter are represented by **"?" symbol** and each parameter is referred to by its origin position.

Example : Insert operation with PreparedStatement Interface

```
import java.sql.*;
class PreparedStatDemo
{
    public static void main(String args[])
    {
       try
       {
          Class.forName("oracle.jdbc.driver.OracleDriver");
          Connection con =
```

```
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","username", "password");
            PreparedStatementps = con.prepareStatement("insert into Student values(?, ?, ?)");
            ps.setInt(1, 101);
            ps.setString(2, "Surendra");
            ps.setString(3, "MCA");
            ps.executeUpdate();
            con.close();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

CallableStatement Interface

The **CallableStatement interface** is used to execute the SQL stored procedure in a database. The JDBC API provides stored procedures to be called in a standard way for all RDBMS.

A stored procedure works like a function or method in a class. The stored procedure makes the performance better because these are precompiled queries.

Creating CallableStatement Interface
The instance of a CallableStatement is created by calling **prepareCall()** method on a Connection object.

For example:

```
CallableStatementcallableStatement = con.prepareCall("{call procedures(?,?)}");
```

Example : CallableStatement Interface using Stored procedure

Creating stored procedure

```
create or replace procedure "insertStudents"
(rollno IN NUMBER,
name IN VARCHAR2,
course IN VARCHAR2)
is
begin
insert into Students values(rollno, name, course);
end;
/
```

// ProcedureDemo.java

```
import java.sql.*;
classProcedureDemo
{
   public static void main(String args[])
   {
      try
      {
         Class.forName("oracle.jdbc.driver.OracleDriver");
         Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","tiger");
         CallableStatementstmt = con.prepareCall("{call insertStudents(?, ?) }");
         stmt.setInt(1, 101);
         stmt.setString(2, Vinod);
         stmt.setString(3, BE);
         stmt.execute();
         System.out.println("Record inserted successfully");
         con.close();
         stmt.close();
      }
      catch(Execption e)
      {
         e.printStackTrace();
      }
   }
}
```

Note: The **ProcedureDemo.java** file inserts the record in Students table in Oracle database by use of stored procedure.

ResultSet Interface

- The result of the query after execution of database statement is returned as table of data according to rows and columns. This data is accessed using the **ResultSet** interface.
- A default **ResultSet** object is not updatable and the cursor moves only in forward direction.

Creating ResultSet Interface

To execute a **Statement or PreparedStatement**, we create **ResultSet** object.

Example

```
Statement stmt = connection.createStatement();
ResultSet result = stmt.executeQuery("select * from Students");
```

Or

```
String sql = "select * from Students";
PreparedStatementstmt = con.prepareStatement(sql);
ResultSet result = stmt.executeQuery();
```

ResultSet Interface Methods

Methods	Description
public boolean absolute(int row)	Moves the cursor to the specified row in the ResultSet object.
public void beforeFirst( )	It moves the cursor just before the first row i.e. front of the ResultSet.
public void afterLast()	Moves the cursor to the end of the ResultSet object, just after the last row.
public boolean first()	Moves the cursor to first value of ResultSet object.
public boolean last( )	Moves the cursor to the last row of the ResultSet object.
public boolean previous ( )	Just moves the cursor to the previous row in the ResultSet object.
public boolean next( )	It moves the curser forward one row from its current position.
public intgetInt(intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.
public String getString(intcolIndex)	It retrieves the value of the column in current row as int in given ResultSet object.
public void relative(int rows)	It moves the cursor to a relative number of rows.

MCQ's on JDBC

1. Which of the following contains both date and time?
   a) java.io.date
   b) java.sql.date
   c) java.util.date
   d) java.util.dateTime
   Answer: d
   Explanation: java.util.date contains both date and time. Whereas, java.sql.date contains only date.
2. Which of the following is advantage of using JDBC connection pool?
   a) Slow performance
   b) Using more memory
   c) Using less memory
   d) Better performance
   Answer: d
   Explanation: Since the JDBC connection takes time to establish. Creating connection at the application start-up and reusing at the time of requirement, helps performance of the application.
3. Which of the following is advantage of using PreparedStatement in Java?
   a) Slow performance
   b) Encourages SQL injection
   c) Prevents SQL injection
   d) More memory usage
   Answer: c
   Explanation: PreparedStatement in Java improves performance and also prevents from SQL injection.
4. Which one of the following contains date information?
   a) java.sql.TimeStamp
   b) java.sql.Time
   c) java.io.Time
   d) java.io.TimeStamp
   Answer: a
   Explanation: java.sql.Time contains only time. Whereas, java.sql.TimeStamp contains both time and date.
5. Which of the following is used to limit the number of rows returned?
   a) setMaxRows(int i)
   b) setMinRows(int i)
   c) getMaxrows(int i)
   d) getMinRows(int i)
   Answer: a
   Explanation: setMaxRows(int i) method is used to limit the number of rows that the database returns from the query.

6. Which of the following is method of JDBC batch process?
   a) setBatch()
   b) deleteBatch()
   c) removeBatch()
   d) addBatch()
   Answer: d
   Explanation: addBatch() is a method of JDBC batch process. It is faster in processing than executing one statement at a time.
7. Which of the following is used to rollback a JDBC transaction?
   a) rollback()
   b) rollforward()
   c) deleteTransaction()
   d) RemoveTransaction()
   Answer: a
   Explanation: rollback() method is used to rollback the transaction. It will rollback all the changes made by the transaction.
8. Q 1 - Which of the following is correct about DriverManager class of JDBC?
   A - JDBC DriverManager is a class that manages a list of database drivers.
   B - It matches connection requests from the java application with the proper database driver using communication subprotocol.
   C - Both of the above.
   D - none of the above.
   Answer : C
   Explanation
   JDBC DriverManager is a class that manages a list of database drivers. It matches connection requests from the java application with the proper database driver using communication subprotocol.
9. Q 2 - Which of the following manages a list of database drivers in JDBC?
   A - DriverManager
   B - JDBC driver
   C - Connection
   D - Statement
   Answer : A
   Explanation
   DriverManager class manages a list of database drivers in JDBC.
10. Q 3 - Which of the following type of JDBC driver, is also called Type 1 JDBC driver?
    A - JDBC-ODBC Bridge plus ODBC driver
    B - Native-API, partly Java driver
    C - JDBC-Net, pure Java driver
    D - Native-protocol, pure Java driver
    Answer : A
    Explanation
    JDBC-ODBC Bridge plus ODBC driver, is also called Type 1 JDBC driver.

11. Which of the following is correct about JDBC?
    A - The JDBC API provides the abstraction andthe JDBC drivers provide the implementation.
    B - New drivers can be plugged-in to the JDBC API without changing the client code.
    C - Both of the above.
    D - None of the above.
    Answer : C
    Explanation
    The JDBC API provides the abstraction and the JDBC drivers provide the implementation. New drivers can be plugged-in to the JDBC API without changing the client code.
12. How many Result sets available with the JDBC 2.0 core API?
    a. 2
    b. 3
    c. 4
    d. 5
    ANSWER: 3
13. Which method is used to establish the connection with the specified url in a Driver Manager class?
    a. public static void registerDriver(Driver driver)
    b. public static void deregisterDriver(Driver driver)
    c. public static Connection getConnection(String url)
    d. public static Connection getConnection(String url,String userName,String password)
    ANSWER: public static Connection getConnection(String url)
14. JDBC RowSet is the wrapper of ResultSet,It holds tabular data like ResultSet but it is easy and flexible to use.
    a. True
    b. False
    ANSWER: True
15. The ResultSet.next method is used to move to the next row of the ResultSet, making it the current row.
    a. True
    b. False
    ANSWER: True
16. **What is used to execute parameterized query?**
    a. Statement interface
    b. PreparedStatement interface
    c. ResultSet interface
    d. None of the above
    ANSWER: PreparedStatement interface