MODULE - V

Java Database Connectivity (JDBC)

MODULE 5

Java Database Connectivity (JDBC)

Module Description

Java Database Connectivity (JDBC) is an Application Programming Interface (API) used to connect Java application with Database. JDBC is used to interact with the various type of Database such as Oracle, MS Access, My SQL and SQL Server and it can be stated as the platform-independent interface between a relational database and Java programming.

This JDBC chapter is designed for Java programmers who would like to understand the JDBC framework in detail along with its architecture and actual usage.

iNurture's **Programming in Java** course is designed to serve as a stepping stone for you to build a career in Information Technology.

Chapter Table of Contents

Chapter 5.1

Introduction to JDBC

Aim

To provide students with a basics concepts of JDBC

Instructional Objectives

After completing this chapter, you should be able to:

- Explain database connectivity
- Demonstrate JDBC architecture*
- Explain different types of JDBC drivers
- Illustrate JDBC API
- Explain the basic steps to create a JDBC application

Learning Outcomes

At the end of this chapter, you are expected to:

- Describe JDBC architecture
- Identify the types of JDBC driver
- Describe the types of statements in JDBC
- Differentiate prepared statement and statement
- Develop a java program and show the database connectivity using JDBC

5.1.1 Database Connectivity

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. It is part of the Java Standard Edition platform, from Oracle Corporation. It provided methods to query and update data in a database and related towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.

The JDBC library combines APIs for each of the tasks discussed below that are associated with database usage.

1. Create a connection to a database.
2. Creating SQL or MySQL statements.
3. Executing SQL or MySQL queries in the database.
4. Viewing & modifying the resulting records.

JDBC is a specification that provides a comprehensive set of interfaces that allows for portable access to an underlying database. Java can be used to write different types of executable, such as:

1. Java Applications
2. Java Applets
3. Java Servlets
4. Java Server Pages (JSPs)
5. Enterprise JavaBeans (EJBs).

All of these different executable can use a JDBC driver to access a database and take advantage of the stored data. JDBC provides the same capabilities as ODBC, allowing Java programs to contain database-independent code. JDBC connection supports to create and execute statements. These statements are an updated statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT.

Statement are termed as a report which is sent to the database server all time. Prepared Statements are statements which are cached and the execution path is pre-determined on the database server. Callable Statements are used for executing stored methods.

Update statements such as INSERT, UPDATE and DELETE return indicates how many rows were affected in the database.

These statements rather do not return information.

Query statements come to a JDBC row result set. Separate columns in a row are retrieved by name or by its number. It may have any number of rows in the result set.

Using JDBC

1. **To execute a statement against a database, the following flow is observed**
   - Load the driver (Only performed once)
   - Obtain a Connection to the database (Save for later use)
   - Get a Statement object from the Connection
   - Use the Statement object to execute SQL. Updates, inserts and deletes return Boolean. Selects return a ResultSet
   - Navigate ResultSet, using data as required
   - Close ResultSet
   - Close Statement
2. **DO NOT close the connection**
   - The same connection object can be used to create further statements.
   - A Connection may only have one active Statement at a time. Do not forget to close the statement when it is no longer needed.
   - Close the connection when you no longer need to access the database.
3. **Using a connection**

The Connection interface defines many methods for managing and using a connection to the database. Few methods are:

public Statement createStatement( )

public PreparedStatement prepareStatement(String sql)

public void setAutoCommit(boolean)

public void commit( )

public void rollback( )

public void close( )

The most commonly used method is createStatement( ).These statements are an updated statements such as SQL's CREATE, INSERT, UPDATE and DELETE, or they may be query statements such as SELECT. Statements are termed as a report which is sent to the database server all time. Prepared Statements are statements which are cached and the execution path is pre-determined on the database server. When an SQL statement issues against the database, a Statement object must be created through the Connection.

Types of Connections are made to Database Servers

1. Direct connections

Having a direct relationship means you directly connect from the client to the database without an average service. To get a direct connection to a database, a programmer must have the correct direct connect drivers installed on the connecting client.

When attached to a database server and the databases stored on it, you must use a Windows-authenticated login. It is a process of identifying an individual user with credentials supplied by the Windows operating system of the user's computer. Hence, the login with which you log in to the client computer is the login that uses the connection. When connecting to a remote database server make use of a domain login. When connecting to a local database server, use either a local or a domain login. If you use a domain login while connecting to a local database server, you may not be able to log into the database server.

2. Local vs. remote connections

When connecting to database servers which reside on the same computer as the connecting client application, make use of a local or domain account to log in.

For example, if your local login is mymachine\myuser, creating a login with the same name on the remote machine, your computer, results in this login: yourmachine\myuser. These are essentially two different login names.

Self-assessment Questions

1) Identify which of the following is NOT a component/class of JDBC API?
   a) Statement
   b) ResultSet
   c) SQLException
   d) ConnectionPool

2) Choose one of the following steps to establish a connection with a database?
   a) Import packages containing the JDBC classes needed for database programming.
   b) Register the JDBC driver, so that you can open a communications channel with the database.
   c) Open a connection using the DriverManager. getConnection( ) method.
   d) Execute a query using an object of type Statement.

3) Methods for contacting a database is Connection.
   a) True
   b) False

4) JDBC architecture decouples an abstraction from its implementation and follows a bridge design pattern.
   a) True
   b) False

5) Is there any possibility for implementing an interface in JSP?
   a) Yes
   b) No
   c) None
   d) Can't say

5.1.2 JDBC Architecture

The JDBC API support both two-tier and three-tier processing models for database access but in usually, JDBC Architecture consists of two layers:

1. **JDBC Driver API:** JDBC Driver API supports the JDBC Manager-to-Driver Connection.

2. **JDBC API:** This provides the application-to-JDBC Manager connection.

The JDBC API applies a database-specific drivers and driver manager to provide good connectivity to create heterogeneous databases.

The JDBC driver manager secures that the correct driver is used to obtain each data source. The driver manager is able to support multiple concurrent drivers connected to multiple heterogeneous databases.

Following is the JDBC architectural diagram, which shows the location of the driver manager on the JDBC drivers and the Java application:

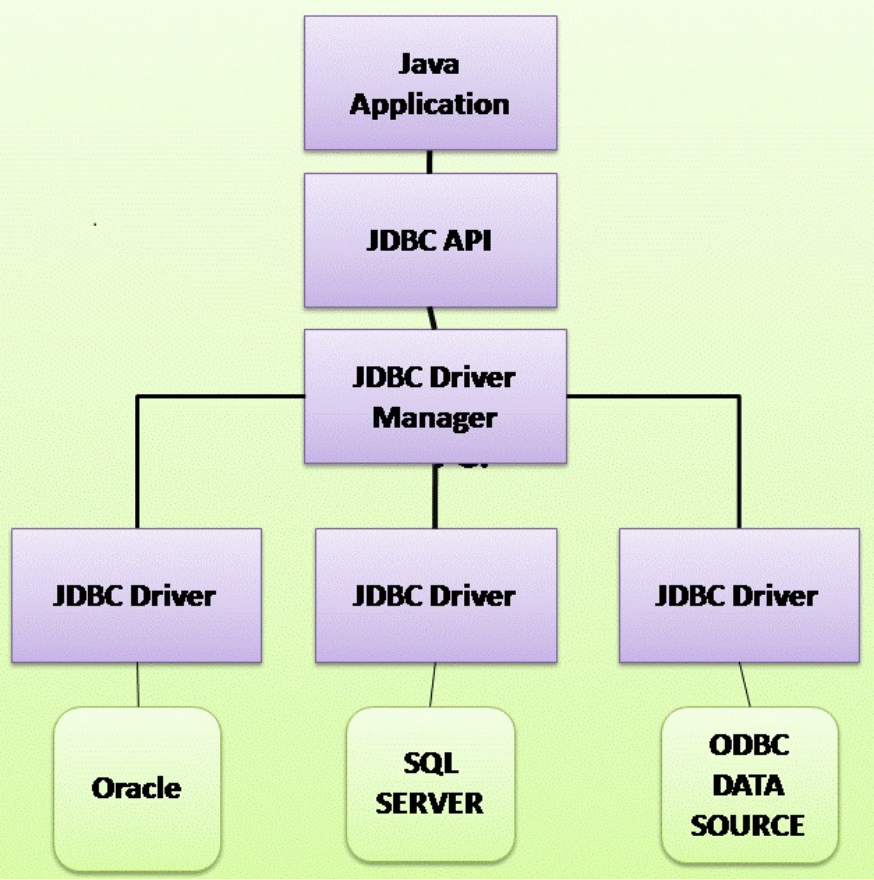

Figure 5.1.1: JDBC Architecture

Common JDBC Component:

These are the following interfaces and classes provided by JDBC API:

- **Driver Manager:** This class manages a list of database drivers. Matches connection requests from the Java application with the proper database driver using communication sub protocol. The first driver that recognises a certain sub protocol under JDBC will be used to establish a database Connection.

- **Driver:** This interface handles the connections with the database server. You will interact directly with Driver objects very rarely. Instead, JDBC uses Driver Manager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.

- **Connection:** This interface with all methods for contacting a database. The connection object represents communication context, *i.e.,* all communication with the database is through connection object only.

- **Statement:** You use objects created to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.

- **ResultSet:** These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.

- **SQLException:** This class handles any errors that occur in a database application.

Different types of JDBC drivers are explained in the following topics and the statements of JDBC is defined with examples. The following are the steps to create a JDBC application.

The necessary steps to create a JDBC application.

There are following six steps involved in building a JDBC application:

Step 1: Import the packages: Needs that coder include the packages including the JDBC classes needed for database programming. Often, we will use "using import java.sql.*;" as suffice.

Step 2: Register the JDBC driver: Requires that you initialise a driver so you can open a communication channel with the database.

Step 3: Open a connection: Requires using the DriverManager.getConnection( ) method to create a Connection object, which represents a physical connection with the database.

Step 4: Execute a query: Requires using an object of type Statement for building and submitting an SQL statement to the database.

Step 5: Extract data from result set: Requires that you use the appropriateResultSet.get( ) method to retrieve the data from the result set.

Step 6: Clean up the environment: Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

For example,

```
//Step 1:   Import required packages
import java.sql.*;
public class JdbcCon
{
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.sql.jdbc.Driver";
   static final String DB_URL = "jdbc:sql://localhost/EMPE";
   // Database credentials
   static final String UNAME = "username";
   static final String UPASS = "password";
   public static void main(String[] args)
   {
      Connection con = null;
      Statement stmtt = null;
      try
      {
         //Step 2: Register JDBC driver
         Class.forName("com.mysql.jdbc.Driver");
         //Step 3:   Open a connection
         System.out.println("Connecting to database...");
         con = DriverManager.getConnection(DB_URL,UNAME ,UPASS);
         //Step 4:   Execute a query
         System.out.println("Creating statement...");
         stmtt = conn.createStatement();
         String mysql;
         sql = "SELECT id, firstname, lastname, age FROM student";
         ResultSet rts = stmtt.executeQuery(sql);
         //Step 5:  Extract data from result set
         while(rts.next())
         {
            //Retrieve by column name
            int id = rts.getInt("id");
            int age = rts.getInt("yage");
            String first = rts.getString("first");
```

```
            String last = rts.getString("last");
            //Display values
            System.out.print("ID: " + id);
            System.out.print(", YourAge: " + age);
            System.out.print(", First: " + firstname);
            System.out.println(", Last: " + lastname);
         }
         //Step 6:   Clean-up environment
         rts.close();
         stmtt.close();
         con.close();
      }
      catch(SQLException se)
      {
         //Handle errors for JDBC
         se.printStackTrace();
      }
      catch(Exception e)
      {
         //Handle errors for Class.forName
         e.printStackTrace();
      }
      Finally
      {
         //finally block used to close resources
         try
         {
            if(stmtt!=null)
            stmtt.close();
         }
         catch(SQLException se2)
         { }// nothing we can do
         try{
         if(con!=null)
         con.close();
      }
      catch(SQLException se)
      {
         se.printStackTrace();
      }
      //end finally try
      }
      //end try
      System.out.println("Goodbye!");
   }
   //end main
}
//end FirstExample
```

Now we are going to compile the above example as follows:

C:\>javac FirstExample.java

C:\>

When you run FirstExample, it produces the following result:

C:\>java FirstExample

Connecting to database...

Creating statement...

ID: 10, YourAge: 18, First: Zara, Last: Khan

ID: 11, YourAge : 25, First: Prabha, Last: karan

ID: 12, YourAge: 30, First: Zaid, Last: Ali

ID: 13, YourAge: 28, First: Sunil, Last: Mittal

C:\>

Self-assessment Questions

6) How many layers JDBC architecture consists of?

a) 1 b) 3

c) 2 d) 5

7) JDBC API support_____ processing models for database access.

a) Only two-tier b) Only three-tier

c) Both a and b

5.1.3 JDBC Drivers

JDBC driver is used for interacting with the database server which implements the stated interfaces in the JDBC API.

For example, JDBC drivers interact with the database to open database connections by sending SQL or database commands and receive the results with Java:

- Any proprietary APIs are implemented by a JDBC driver.

Types of JDBC Drivers

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

1. **Type 1:** JDBC - ODBC bridge driver
2. **Type 2:** Java - Native code driver
3. **Type 3:** Network - Protocol driver (Middleware driver)
4. **Type 4:** Database - Protocol driver (Pure Java driver)

(i) JDBC – ODBC Bridge

ODBC driver is a type 1 driver installed on every client machine. It can be accessible by the JDBC Bridge which supports ODBC accessibility but at present this type is recommended only for experimental purposes.

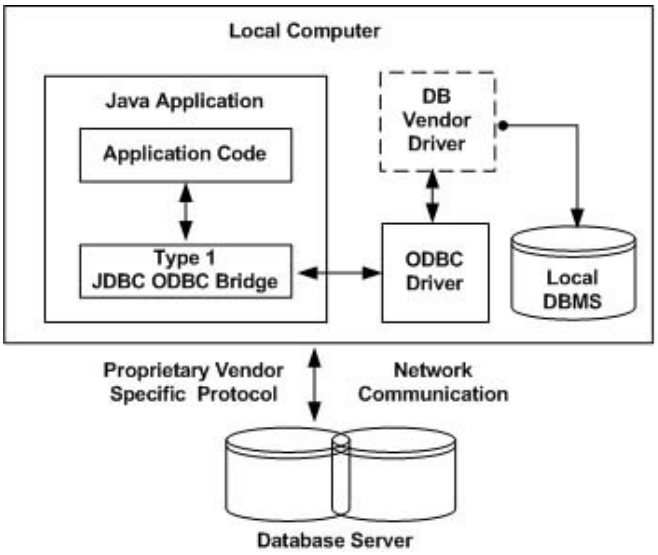

Figure 5.1.2: JDBC- ODBC Bridge

In a Type 1 driver (JDBC – ODBC Bridge), a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC requires configuring your system a Data Source Name (DSN) that represents the mark database. When Java first came out, JDBC – ODBC Bridge was a valuable driver because most databases only supported ODBC access but now this type of driver is approved only for experimental use or when no another alternative is available.

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

Advantages:

- Easy to use.
- Can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC calls converts into ODBC function calls.
- The ODBC driver requires installation on the client machine.

(ii) Native – API Driver

JDBC Native API is unique to the database because these API calls are converted into native C/C++ API calls. They are provided by the database vendors and used in a similar manner as the JDBC-ODBC Bridge.

The vendor-specific driver must be installed on each client machine and if the programmer wants to change the database, then the programmer has to modify the native API too.

JDBC-Native API can perform faster when compared to Type 1 driver because it eliminates ODBC's Overhead.

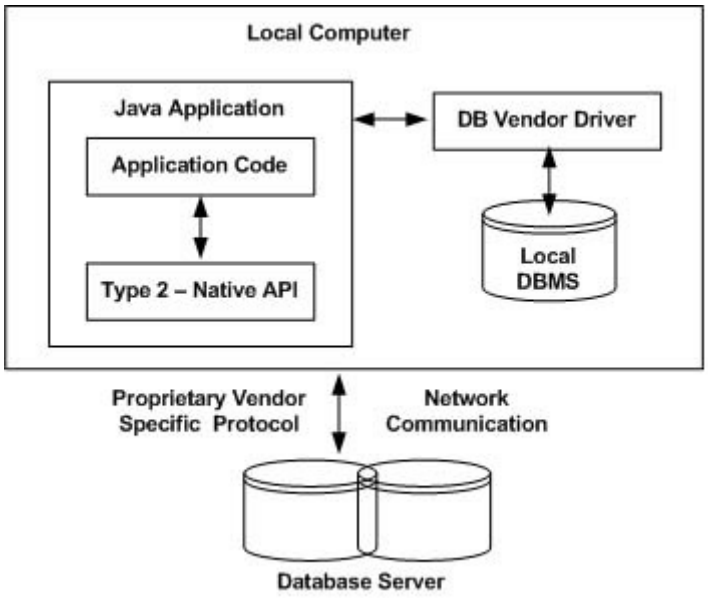

Figure 5.1.3: Native API Driver

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

Advantage:

- Performance upgrades than JDBC-ODBC bridge driver.

Disadvantages:

- The Native driver requires installation on the each client machine.
- The Vendor client library requires installation on client machine.

(iii) Network – Protocol Driver (Middleware Driver)

Network-Protocol driver is also called as middleware driver. It has a three-tier approach to access databases. It communicates with the middleware application server through a standard network socket. Then the device information is translated by the middleware application server as a call format required by the DBMS and forwarded to the database server.

Middleware drivers are extremely flexible since no code is installed on the client side. Even a single driver can provide access to multiple databases.

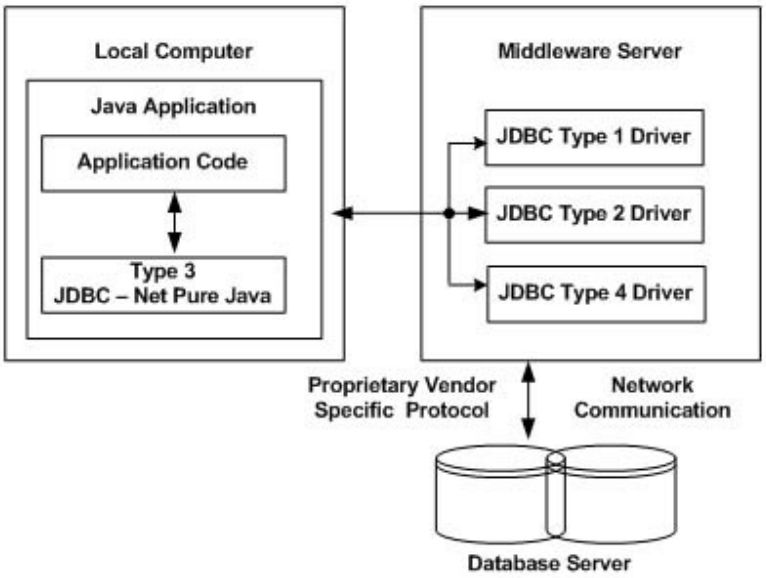

Figure 5.1.4: Middleware Driver

You can analyse the application server as a JDBC "proxy," meaning that it creates calls to the client application. As a result, you need some information about the application server's configuration to effectively use this driver type. Your application server might use a Type 1, 2, or 4 drivers to communicate with the database, understanding the nuances will prove helpful.

Advantage:

- No client-side library needs the application server that can perform many tasks like auditing, load balancing, logging, etc.

Disadvantages:

- Network support is necessary on the client machine.
- It requires database-specific coding for the middle tier.
- Maintenance of Network Protocol driver is costly because it requires database-specific coding in the middle tier.

(iv) Database – Protocol Driver (Pure Java Driver)

Database- Protocol driver is a type 4 driver called as a pure Java driver. It communicates directly with the vendor's database through socket information. It acts as the highest performance driver available for the database which is provided by the supplier itself.

Pure Java driver does not require any special software to install on the client or server side rather these drivers can be downloaded dynamically.

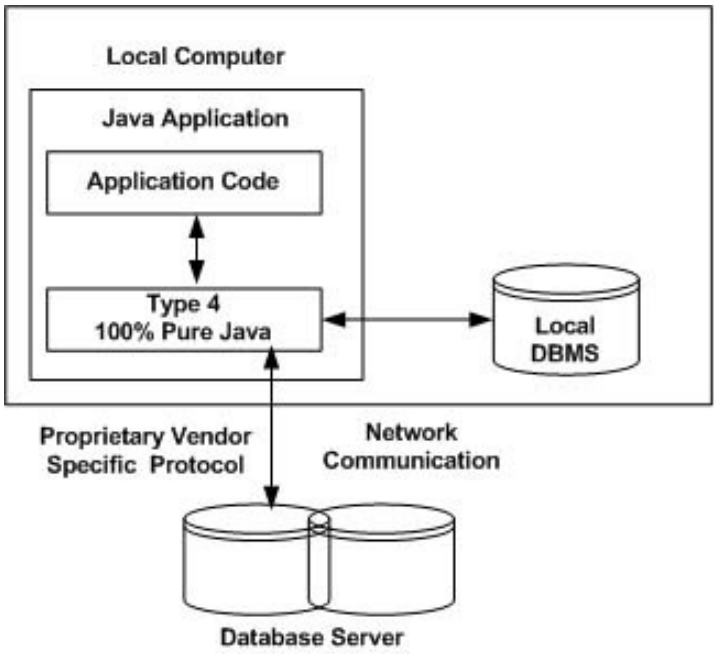

Figure 5.1.5: 100% Pure Java Driver

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

Advantages:

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

Disadvantage:

- Drivers depends on the Database.

Self-assessment Questions

7) Which is the type of JDBC driver, calls native code of the locally available ODBC 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

8) Which of the following type of JDBC driver, is also called Type 3 JDBC driver?
   a) JDBC-ODBC Bridge plus ODBC driver
   b) Nativ e-API, partly Java driver
   c) JDBC-Net, pure Java driver
   d) Native-protocol, pure Java driver

9) A native-protocol driver allows a direct call from the client machine to the DBMS server.
   a) True b) False

10) JDBC - ODBC Bridge provides JDBC API access via ODBC drivers.
   a) True b) False

5.1.4 JDBC API

Java programming language provides a universal data access to the Java Database Connectivity (JDBC) API. The application of JDBC API is that you can access virtually any data source, relational databases to spreadsheets and files. It provides a common base on which tools and alternate interfaces evolve.

The JDBC API has two packages:

1. java.sql
2. javax.sql

Java Database Connectivity (JDBC) API:

- General API (Application programming interface) for Java client code to connect to a database
- Database providers (IBM, Oracle, etc.) provide drivers
- **Driver:** specific implementation of the API for interacting with a particular database
- Support for
    a) Statement
    b) Prepared Statement
    c) Callable Statement (stored procedures)
    d) Common Java data types (Integer, Double, java.sql.Date)
- The javax.sql and java.sql are the primary packages for JDBC 4.0. It is the modern JDBC version at the time of writing this tutorial. It offers the leading (main) classes for interacting with your data sources. The new features in these packages include changes in the following areas:
    - Automatic database driver loading.
    - Exception handling improvements.
    - Enhanced BLOB/CLOB functionality.
    - Connection and statement interface enhancements.
    - National character set support.
    - SQL ROWID access.

- SQL 2003 XML data type support.
- Annotations.

Self-assessment Questions

11) The application of JDBC API is that you cannot access virtually any data source, relational databases to spreadsheets and files.

a) True b) False

12) The abbreviation of API

a) Application programming interface
b) Anti-programming interface
c) Abdicate programming interface
d) Abscising programming interface

13) Which processing models supports database access?

a) Only two-tier b) Only three-tier
c) Both two-tier and three-tier d) None of these

14) The latest version of JDBC 4.0 API is divided into two packages, these are?

1. java.sql
2. javax.sql
3. javac.sql
4. java.lang.sql

a) 2 and 3 b) 1 and 2
c) 1 and 3 d) 1 and 4

15) Select the API that provides the application-to-JDBC Manager connection?

a) JDBC API b) JDBC Driver API
c) Both JDBC API JDBC Driver API d) None of these

16) The JDBC API provide facility the facility for accessing the _____ from the java programming language.

a) Database b) Relational database
c) Irrational database d) None of these

Summary

- Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database.
- It is part of the Java Standard Edition platform, from Oracle Corporation. It provided methods to query and update data in a database and related towards relational databases.
- A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the Java virtual machine (JVM) host environment.
- In Java database connectivity JDBC API is a Java API that can access any tabular data, especially data stored in a Relational Database.
- JDBC works with Java on a variety of platforms, such as Windows, Mac OS and the various versions of UNIX.
- JDBC driver is used for interacting with the database server which implements the stated interfaces in the JDBC API.
- The JDBC driver manager secures that the correct driver is used to obtain each data source. The driver manager is able to support multiple concurrent drivers connected to multiple heterogeneous databases
- Having a direct relationship means you directly connect from the client to the database without an average service. To get a direct connection to a database, a programmer must have the correct direct connect drivers installed on the connecting client.

Terminal Questions

1. Explain database connectivity
2. Demonstrate JDBC architecture
3. Explain different types of JDBC drivers
4. Illustrate JDBC API
5. Explain the basic steps to create a JDBC application

Answer Keys

Self-assessment Questions	
Question No.	Answer
1	d
2	c
3	a
4	a
5	b
6	c
7	a
8	c
9	a
10	b
11	b
12	a
13	c
14	b
15	b
16	b

Activity

Activity Type: Offline **Duration:** 30 Minutes

Description:

Compare Type 1, Type 2, Type 3 and Type 4 JDBC drivers. Create a table which describe the features, advantages and disadvantages of these JDBC drivers.

Bibliography

External Resources

- The complete reference Java-2: Herbert Schildt-7th edition- McGraw Hill professional
- SAMS teach yourself Java-2- Rogers Cedenhead and Leura Lemay- 3rd edition – Pearson
- JDBC and Java database programming books, Laurence Vanhelsuwe

Video Links

Topic	Link
Architecture	https://docs.oracle.com/javase/tutorial/jdbc/overview/architecture.html
Types of Drive	http://tutorials.jenkov.com/jdbc/driver-types.html
Drive	https://www.youtube.com/watch?v=bo62AqckwHc
Types of Drive	http://tutorials.jenkov.com/jdbc/driver-types.html

Notes: