UNIT 2 JAVA DATABASE CONNECTIVITY

Structure **Page Nos.**

2.0 INTRODUCTION

In previous blocks of this course we have learnt the basics of Java Servlets. In this UNIT we shall cover the database connectivity with Java using JDBC. JDBC (the Java Database Connectivity) is a standard SQL database access interface that provides uniform access to a wide range of relational databases like MS-Access, Oracle or Sybase. It also provides a common base on which higher-level tools and interfaces can be built. It includes ODBC Bridge. The Bridge is a library that implements JDBC in terms of the ODBC standard C API. In this unit we will first go through the different types of JDBC drivers and then JDBC API and its different objects like connection, statement and ResultSet. We shall also learn how to query and update the database using JDBC API.

2.1 OBJECTIVES

After going through this unit, you should be able to:

- understand the basics of JDBC and ODBC;
- understand the architecture of JDBC API and its objects;.
- understand the different types of statement objects and their usage;
- understand the different Types of JDBC drivers & their advantages and disadvantages;
- steps to connect a database;
- how to use JDBC to query a database and,
- understand, how to use JDBC to modify the database.

2.2 JDBC Vs. ODBC

Now, we shall study the comparison between JDBC and ODBC. The most widely used interface to access relational databases today is Micorsoft's ODBC API. ODBC stands for Open Database Connectivity, a standard database access method developed by the SQL Access group in 1992. Through ODBC it is possible to access any data from any application, regardless of which database management system (DBMS) is handling the data. ODBC manages this by inserting a middle layer, called a *database*

driver, between an application and the DBMS. The purpose of this layer is to translate the application's data queries into commands that the DBMS understands.

Microsoft ODBC API offers connectivity to almost all databases on almost all platforms and is the most widely used programming interface for accessing relational databases. But ODBC cannot be used directly with Java Programs due to various reasons described below.

1) ODBC cannot be used directly with Java because, it uses a C interface. This will have drawbacks in the security, implementation, and robustness.
2) ODBC makes use of Pointers, which have been removed from Java.
3) ODBC mixes simple and advanced features together and has complex structure.

Hence, JDBC came into existence. If you had done Database Programming with Visual Basic, then you will be familiar with ODBC. You can connect a VB Application to MS-Access Database or an Oracle Table directly via ODBC. Since Java is a product of Sun Microsystems, you have to make use of JDBC with ODBC in order to develop Java Database Applications.

JDBC is an API (Application Programming Interface) which consists of a set of Java classes, interfaces and exceptions With the help of JDBC programming interface, Java programmers can request a connection with a database, then send query statements using SQL and receive the results for processing.

According to Sun, specialised JDBC drivers are available for all major databases — including relational databases from Oracle Corp., IBM, Microsoft Corp., Informix Corp. and Sybase Inc. — as well as for any data source that uses Microsoft's Open Database Connectivity system.

The combination of Java with JDBC is very useful because it lets the programmer run his/ her program on different platforms. Some of the advantages of using Java with JDBC are:

- Easy and economical
- Continued usage of already installed databases
- Development time is short
- Installation and version control simplified.

2.3 HOW DOES JDBC WORK?

Simply, JDBC makes it possible to do the following things within a Java application:

- Establish a connection with a data source
- Send SQL queries and update statements to the data source
- Process the results at the front-end

Figure 1 shows the components of the JDBC model

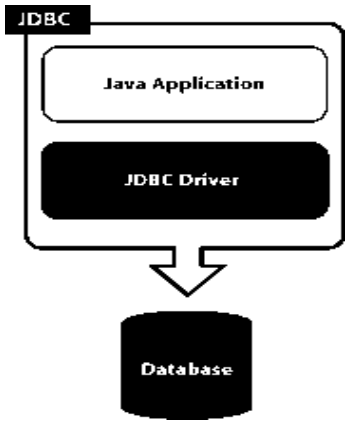

Figure 1: Components of java database connectivity

The Java application calls JDBC classes and interfaces to submit SQL statements and retrieve results.

2.4 JDBC API

Now, we will learn about the JDBC API. The JDBC API is implemented through the JDBC driver. The JDBC Driver is a set of classes that implement the JDBC interfaces to process JDBC calls and return result sets to a Java application. The database (or data store) stores the data retrieved by the application using the JDBC Driver.
The API interface is made up of 4 main interfaces:

- java.sql DriverManager
- java. sql .Connection
- java. sql. Statement
- java.sql.Resultset

In addition to these, the following support interfaces are also available to the developer:

- java.sql.Callablestatement
- java. sql. DatabaseMetaData
- java.sql.Driver
- java. sql. PreparedStatement
- java. sql .ResultSetMetaData
- java. sql. DriverPropertymfo
- java.sql.Date
- java.sql.Time
- java. sql. Timestamp
- java.sql.Types
- java. sql. Numeric

The main objects of the JDBC API include:

- A **DataSource** object is used to establish connections. Although the Driver Manager can also be used to establish a connection, connecting through a DataSource object is the preferred method.

- A **Connection** object controls the connection to the database. An application can alter the behavior of a connection by invoking the methods associated with this object. An application uses the connection object to create statements.

- **Statement** object are used for executing SQL queries.

Different types of JDBC SQL Statements

a) java.sql.Statement : Top most interface which provides basic methods useful for executing SELECT, INSERT, UPDATE and DELETE SQL statements.

b) java.sql.PreparedStatement : An enhanced verion of java.sql.Statement which allows precompiled queries with parameters. A *PreparedStatement* object is used when an application plans to specify parameters to your SQL queries. The statement can be executed multiple times with different parameter values specified for each execution.

c) **java.sql.CallableStatement :** It allows you to execute stored procedures within a RDBMS which supports stored procedures. The Callable Statement has methods for retrieving the return values of the stored procedure.

A **ResultSet** Object act like a workspace to store the results of query. A ResultSet is returned to an application when a SQL query is executed by a statement object. The ResultSet object provides several methods for iterating through the results of the query.

2.5 TYPES OF JDBC DRIVERS

We have learnt about JDBC API. Now we will study different types of drivers available in java of which some are pure and some are impure.

To connect with individual databases, JDBC requires drivers for each database. There are four types of drivers available in Java for database connectivity. Types 3 and 4 are pure drivers whereas Types 1 and 2 are impure drivers. Types 1 and 2 are intended for programmers writing applications, while Types 3 and 4 are typically used by vendors of middleware or databases.

Type 1: JDBC-ODBC Bridge

They are JDBC-ODBC Bridge drivers. They delegate the work of data access to ODBC API. ODBC is widely used by developers to connect to databases in a non-Java environment. This kind of driver is generally most appropriate when automatic installation and downloading of a Java technology application is not important.

Note: Some ODBC native code and in many cases native database client code must be loaded on each client machine that uses this type of driver.

Advantages: It acts as a good approach for learning JDBC. It may be useful for companies that already have ODBC drivers installed on each client machine — typically the case for Windows-based machines running productivity applications. It may be the only way to gain access to some low-end desktop databases.

Disadvantage: It is not suitable for large-scale applications. They are the slowest of all. The performance of system may suffer because there is some overhead associated with the translation work to go from JDBC to ODBC. It doesn't support all the features of Java. User is limited by the functionality of the underlying ODBC driver, as it is product of different vendor.

Type 2: Native-API partly Java technology-enabled driver

They mainly use native API for data access and provide Java wrapper classes to be able to be invoked using JDBC drivers. It converts the calls that a developer writes to the JDBC application programming interface into calls that connect to the client machine's application programming interface for a specific database, such as IBM, Informix, Oracle or Sybase, like, the bridge driver, this style of driver requires that some binary code be loaded on each client machine.

Advantage: It has a better performance than that of Type 1, in part because the Type 2 driver contains *compiled code* that's optimised for the back-end database server's operating system.

Disadvantage: For this, User needs to make sure the JDBC driver of the database vendor is loaded onto each client machine. Must have compiled code for every operating system that the application will run on. Best use is for controlled environments, such as an intranet.

Type 3: A net-protocol fully Java technology-enabled driver

They are written in 100% Java and use vendor independent Net-protocol to access a vendor independent remote listener. This listener in turn maps the vendor independent calls to vender dependent ones. This extra step adds complexity and decreases the data access efficiency. It is pure Java driver for database middleware, which translates JDBC API calls into a DBMS-independent net protocol, which is then translated, to a DBMS protocol by a server. It translates JDBC calls into the middleware vendor's protocol, which is then converted to a database-specific protocol by the middleware server software. This net server middleware is able to connect all of its Java technology-based clients to many different databases. In general, this is the most flexible JDBC API alternative.

Advantage: It has better performance than Types 1 and 2. It can be used when a company has multiple databases and wants to use a single JDBC driver to connect to all of them. Since, it is server-based, so there is no requirement for JDBC driver code on client machine. For performance reasons, the back-end server component is optimized for the operating system that the database is running on.

Disadvantage: It needs some database-specific code on the middleware server. If the middleware is to run on different platforms, then Type 4 driver might be more effective.

Type 4: A native-protocol fully Java technology-enabled driver

It is direct-to-database pure Java driver. It converts JDBC technology calls into the network protocol used by different DBMSs directly. Basically it converts JDBC calls into packets that are sent over the network in the proprietary format used by the specific database. Allows a direct call from the client machine to the database.

Advantage: It again has better performance than Types 1 and 2 and there is no need to install special software on client or server. It can be downloaded dynamically.

Disadvantage: It is not optimized for server operating system, so the driver can’t take advantage of operating system features. (The driver is optimized for the specific database and can take advantage of the database vendor's functionality.). For this, user needs a different driver for each different database.

The following figure shows a side-by-side comparison of the implementation of each JDBC driver type. All four implementations show a Java application or applet using the JDBC API to communicate through the JDBC Driver Manager with a specific JDBC driver.

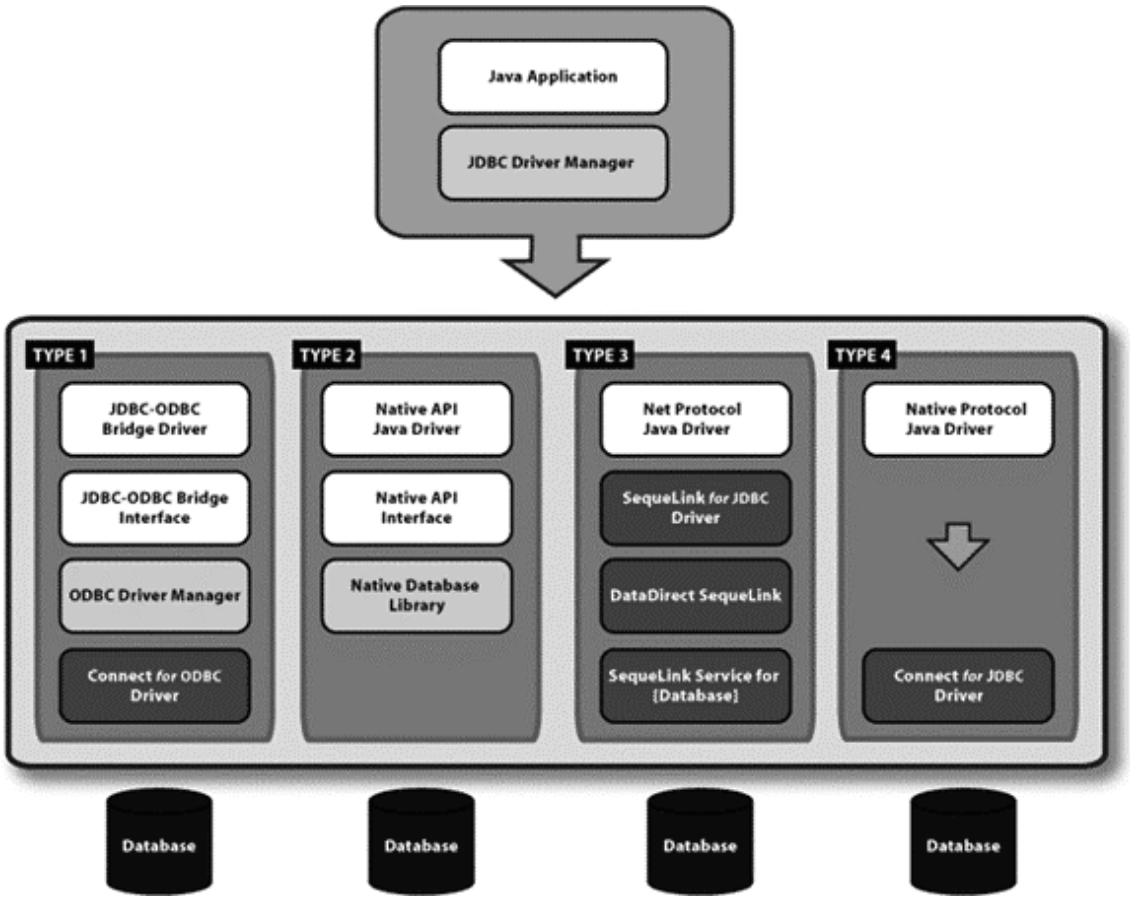

Figure 2: Comparison of different JDBC drivers

Check Your Progress 1

1) State True or False:

a) CallableStatement are used to call SQL stored procedures.
T ☐ F ☐

b) ODBC make use of pointers which have been removed from java.
T ☐ F ☐

To give answers of following questions:

2) What are the advantages of using JDBC with java?

……

……

……

3) Briefly explain the advantages / disadvantages of different types of drviers of JDBC.

……

……

……

4) Why ODBC cannot be used directly with Java programs?

……

……

……

5) What are 3 different types of statements available in JDBC? Where do we use these statements?

……

……

……

2.6 STEPS TO CONNECT TO A DATABASE

Now, we shall learn step-by-step process to connect a database using Java. The interface and classes of the JDBC API are present inside the package called as java.sql package. There any application using JDBC API must import java.sql package in its code.

```
import java.sql.* ;
```

STEP 1: Load the Drivers

The first step in accessing the database is to load an appropriate driver. You can use one driver from the available four drivers which are described earlier. However, JDBC-ODBC Driver is the most preferred driver among developers. If you are using any other type of driver, then it should be installed on the system (usually this requires having the driver jar file available and its path name in your classpath. In order to load the driver, you have to give the following syntax:

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

We can also register the driver (if third party driver) with the use of method **registerMethod()** whose syntax is as follows :

```
DriverManager.registerDriver(Driver dr);
```

Where dr is the new JDBC driver to be registered with the DriverManager.
There are a number of alternative ways to do the actual loading:

1. Use new to explicitly load the Driver class. This hard codes the driver and (indirectly) the name of the database into your program and is not recommended as changing the driver or the database or even the name or location of the database will usually require recompiling the program.

2. Class.forName takes a string class name and loads the necessary class dynamically at runtime as specified in the above example. This is a safe method that works well in all Java environments although it still requires extra coding to avoid hard coding the class name into the program.

3. The System class has a static Property list. If this has a Property jdbc.drivers set to a ':' separated list of driver class names, then all of these drivers will be loaded and registered automatically. Since there is support for loading property lists from files easily in Java, this is a convenient mechanism to set up a whole set of drivers. When a connection is requested, all loaded drivers are checked to see which one can handle the request and an appropriate one is chosen. Unfortunately, support for using this approach in servlet servers is patchy so we will stay with method 2 above but use the properties file method to load the database url and the driver name at runtime:

   ```
   Properties props = new Properties() ;
   FileInputStream in = new FileInputStream("Database.Properties") ;
   props.load(in);
   ```

```
String drivers = props.getProperty("jdbc.drivers") ;
Class.forName(drivers) ;
```

The Database.Properties file contents look like this:

# Default JDBC driver and database specificationjdbc.drivers =
sun.jdbc.odbc.JdbcOdbcDriverdatabase.Shop = jdbc:odbc:Shop

STEP 2: Make the Connection

The getConnection() method of the Driver Manager class is called to obtain the Connection Object. The syntax looks like this:

Connection conn = DriverManager.getConnection("jdbc:odbc:<DSN NAME>");
Here note that getConnection() is a static method, meaning it should be accessed along with the class associated with the method. The DSN (Data Source name) Name is the name, which you gave in the Control Panel->ODBC while registering the Database or Table.

STEP 3: Create JDBC Statement

A Statement object is used to send SQL Query to the Database Management System. You can simply create a statement object and then execute it. It takes an instance of active connection to create a statement object. We have to use our earlier created Connection Object "conn" here to create the Statement object "stmt". The code looks like this:

Statement stmt = conn.createStatement();
As mentioned earlier we may use Prepared Statement or callable statement according to the requirement.

STEP 4: Execute the Statement

In order to execute the query, you have to obtain the ResultSet object similar to Record Set in Visual Basic and call the **executeQuery()** method of the Statement interface. You have to pass a SQL Query like select * from students as a parameter to the executeQuery() method. Actually, the RecordSet object contains both the data returned by the query and the methods for data retrieval. The code for the above step looks like this:

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

If you want to select only the name field you have to issue a SQL Syntax like
Select Name from Student

The executeUpdate() method is called whenever there is a delete or an update operation.

STEP 5: Navigation or Looping through the ResultSet

The ResultSet object contains rows of data that is parsed using the next() method like rs.next(). We use the **getXXX()** like, (getInt to retrieve Integer fields and getString for String fields) method of the appropriate type to retrieve the value in each field. If the first field in each row of ResultSet is Name (Stores String value), then getString method is used. Similarly, if the Second field in each row stores int type, then getInt() method is used like:

System.out.println(rs.getInt("ID"));

STEP 6: Close the Connection and Statement Objects

After performing all the above steps, you must close the Connection, statement and Resultset Objects appropriately by calling the close() method. For example, in our above code we will close the object as:

ResultSet object with

rs.close();

and statement object with

stmt.close();

Connection object with

conn.close();

2.7 USING JDBC TO QUERY A DATABASE

Let us take an example to understand how to query or modify a database. Consider a table named as CUSTOMER is created in MS-ACCESS, with fields cust_id, name, ph_no, address etc.

```
import java.sql.*;

public class JdbcExample1 {

 public static void main(String args[]) {

Connection con = null;

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

Connection Conn = DriverManager.getConnection (jdbc:odbc:Access);

Statement Stmt = Conn.createStatement();

      // To create a string of SQL.

String sql = "SELECT * FROM CUSTOMERS";

      // Next we will attempt to send the SQL command to the database.

     // If it works, the database will  return to us a set of results that JDBC will

      // store in a ResultSet object.

try

  {

     ResultSet results = Stmt.executeQuery(sql);

 // We simply go through the ResultSet object one element at a time and print //out the
fields.  In this example, we assume that the result set will contain three //fields

  while (results.next())

    {
```

```
   System.our.println("Field One: " +  results.getString(1) +  "Field Two: " +
results.getString(2) +  "Field Three: " +  results.getString(3));
    }
  }
// If there was a problem sending the SQL,  we will get this error.
catch (Exception e)
  {
  System.out.println("Problem with Sending Query: " + e);
  }
finally
{
     result.close();
      stmt.close();
      Conn.close();
}
} // end of main method
} // end of class
```

Note that if the field is an Integer, you should use the getInt() method in ResultSet instead of getString().You can use either an ordinal position (as shown in the above example) which starts from 1 for the first field or name of field to access the values from the ResultSet like result.getString("CustomerID");

Compiling JdbcExample1.java

To compile the JdbcExample1.java program to its class file and place it according to its package statements, open command prompt and cd (change directory) into the folder containing JdbcExample2.java, then execute this command:

javac -d . JdbcExample2.java

If the program gets compiled successfully then you should get a new Java class under the current folder with a directory structure JdbcExample2.class in your current working directory.

Running JdbcExample1.java

To run and to see the output of this program execute following command from the command prompt from the same folder where JdbcExample2.java is residing:

java JdbcExample2

2.8 USING JDBC TO MODIFY A DATABASE

Modifying a database is just as simple as querying a database. However, instead of using executeQuery(), you use executeUpdate() and you don't have to worry about a result set. Consider the following example:

```
import java.sql.*;

public class JdbcExample1

 {

  public static void main(String args[])

{

   Connection con = null;

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

            Connection Conn = DriverManager.getConnection (jdbc:odbc:Access);

            Statement Stmt = Conn.createStatement();

 // We have already seen all the above steps

String sql = "INSERT INTO CUSTOMERS +

            " (CustomerID, Firstname, LastName, Email)" +

           " VALUES (004, 'Selena', 'Sol' " + "selena@extropia.com')";

       // Now submit the SQL....

try

  {

     Stmt.executeUpdate(sql);

  }catch (Exception e)

  {

            System.out.println("Problem with Sending Query: " + e);

  }

finally

{

     result.close();

      stmt.close();

      Conn.close();
```

```
}

} // end of main method

} // end of class
```

As you can see, there is not much to it. Add, modify and delete are all handled by the executeUpdate() method or executeUpdate(String str) where str is a SQL Insert, Update or Delete Statement.

Check Your Progress 2

1) What is the most important package used in JDBC?

……

……

……..

2) Explain different methods to load the drivers in JDBC.

……

……

……

3) Assume that there is a table named as Student in MS-Access with the following fields : Std_id, name, course, ph_no.Write a Java program to insert and then display the records of this table using JDBC.

……

……

……

4) What is the fastest type of JDBC driver?

……..……..……

……

……

5) Is the JDBC-ODBC Bridge multi-threaded?

……..……..……

……

……

6) What's the JDBC 3.0 API?

……..……..……

……

……

7) Can we use JDBC-ODBC Bridge with applets?

……..……..……

……

……

8) How should one start debugging problems related to the JDBC API?
……..……..……

……

……

9) Why sometimes programmer gets the error message “java.sql.DriverManager class” not being found? How can we remove these kind of errors?
……..……..……

……

……

10) How can one retrieve a whole row of data at once, instead of calling an individual ResultSet.getXXX method for each column?
……..……..……

……

……

11) Why do one get a NoClassDefFoundError exception when I try and load my driver?
……..……..……

……

……

2.9 SUMMARY

JDBC is an API, which stands for Java Database connectivity, provides an interface through which one can have a uniform access to a wide range of relational databases like MS-Access, Oracle or Sybase. It also provides bridging to ODBC (Open Database Connectivity) by JDBC-ODBC Bridge, which implements JDBC in terms of ODBC standard C API. With the help of JDBC programming interface, Java programmers can request a connection with a database, then send query statements using SQL and receive the results for processing. The combination of Java with JDBC is very useful because it helps the programmer to run the program on different platforms in an easy and economical way. It also helps in implementing the version control.

JDBC API mainly consists of 4 main interfaces i.e. *java.sql DriverManager, java. sql .Connection, java. sql. Statement, java.sql.Resultset.* The main objects of JDBC are Datasource, Connection and statement. A DataSource object is used to establish connections. A Connection object controls the connection to the database. Statement object is used for executing SQL queries. Statement Object is further divided into three categories, which are statement, Prepared Statement and callable statement. Prepared Statement object is used to execute parameterized SQL queries and Callable statement is used to execute stored procedures within a RDBMS.

JDBC requires drivers to connect any of the databases. There are four types of drivers available in Java for database connectivity. First two drivers Type1 and Type 2 are impure Java drivers whereas Type 3 and Type 4 are pure Java drivers. Type 1 drivers act as a JDBC-ODBC bridge. It is useful for the companies that already have ODBC drivers installed on each client machine. Type2 driver is Native-API partly Java technology-enabled driver. It converts the calls that a developer writes to the JDBC application-programming interface into calls that connect to the client machine's application programming interface for a specific database like Oracle. Type-3 driver is A net-protocol fully Java technology-enabled driver and is written in 100% Java. . It

translates JDBC calls into the middleware vendor's protocol, which is then converted to a database-specific protocol by the middleware server software. Type 4 is a native-protocol fully Java technology-enabled driver. It is Direct-to-database pure Java driver. It converts JDBC technology calls into the network protocol used by different DBMSs directly.

To connect a database we should follow certain steps. First step is to load an appropriate driver from any of the four drivers. We can also register the driver by registerMethod(). Second step is to make the connection with the database using a getConnection() method of the Driver Manager class which can be with DSN OR DSN-less connection. Third step is create appropriate JDBC statement to send the SQL query. Fourth step is to execute the query using executeQuery() method and to store the data returned by the query in the Resultset object. Then we can navigate the ResultSet using the next() method and getXXX() to retrieve the integer fields. Last step is to close the connection and statement objects by calling the close() method. In this way we can query the database, modify the database, delete the records in the database using the appropriate query.

2.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1) True or False

   a) True
   b) True

2) JDBC is a standard SQL database access interface that provides uniform access to a wide range of relational databases. It also provides a common base on which higher level tools and interfaces can be built. The advantages of using Java with JDBC are:

   - Easy and economical
   - Continued usage of already installed databases
   - Development time is short
   - Installation and version control simplified

3) There are basically 4 types of drivers available in Java of which 2 are partly pure and 2 are pure java drivers.

Type 1: JDBC-ODBC Bridge.

They delegate the work of data access to ODBC API. This kind of driver is generally most appropriate when automatic installation and downloading of a Java technology application is not important.

Advantages: It acts as a good approach for learning JDBC. It may be useful for companies that already have ODBC drivers installed on each client machine — typically the case for Windows-based machines running productivity applications.

Disadvantage: It is not suitable for large-scale applications. They are the slowest of all. The performance of system may suffer because there is some overhead associated with the translation work to go from JDBC to ODBC. It doesn't support all the features of Java.

Type 2: Native-API partly Java technology-enabled driver

They mainly use native API for data access and provide Java wrapper classes to be able to be invoked using JDBC drivers It converts the calls that a developer writes to

the JDBC application programming interface into calls that connect to the client machine's application programming interface for a specific database, such as IBM, Informix, Oracle or Sybase

Advantage: It has a better performance than that of Type 1, in part because the Type 2 driver contains *compiled code* that's optimized for the back-end database server's operating system.

Disadvantage: In this, user needs to make sure the JDBC driver of the database vendor is loaded onto each client machine. It must have compiled code for every operating system that the application will run on.

Type 3: A net-protocol fully Java technology-enabled driver

They are written in 100% Java and use vendor independent Net-protocol to access a vendor independent remote listener. This listener in turn maps the vendor independent calls to vender dependent ones.

Advantage: It has better performance than Types 1 and 2. It can be used when a company has multiple databases and wants to use a single JDBC driver to connect to all of them. Since, it is server-based, so there is no requirement for JDBC driver code on client machine.

Disadvantage: It needs some database-specific code on the middleware server. If the middleware is to run on different platforms, then Type 4 driver might be more effective.

Type 4: A native-protocol fully Java technology-enabled driver

It is Direct-to-database pure Java driver. It converts JDBC technology calls into the network protocol used by different DBMSs directly. It allows a direct call from the client machine to the database.

Advantage: It again has better performance than Types 1 and 2 and there is no need to install special software on client or server. It can be downloaded dynamically.

Disadvantage: It is not optimized for server operating system, so the driver can't take advantage of operating system features. For this, user needs a different driver for each different database.

4) OBDC (Open Database Connectivity) cannot be used directly with java due to the following reasons:

   a) ODBC cannot be used directly with Java because it uses a C interface. This will have drawbacks in the Security, implementation, and robustness.
   b) ODBC makes use of Pointers, which have been removed from Java.
   c) ODBC mixes simple and advanced features together and has complex structure.

5) Statement object is one of the main objects of JDBC API, which is used for executing the SQL queries. There are 3 different types of JDBC SQL Statements are available in Java:

   a) **java.sql.Statement:** It is the topmost interface which provides basic methods useful for executing SELECT, INSERT, UPDATE and DELETE SQL statements.

b) **java.sql.PreparedStatement:** It is an enhanced version of java.sql.Statement which is used to execute SQL queries with parameters and can be executed multiple times.

c) **java.sql.CallableStatement:** It allows you to execute stored procedures within a RDBMS which supports stored procedures.

Check Your Progress 2

1) The most important package used in JDBC is java.sql package.

2) After importing the java.sql.* package the next step in database connectivity is load the appropriate driver. There are various ways to load these drivers:

1. Class.forName takes a string class name and loads the necessary class dynamically at runtime as specified in the example To load the JDBC-ODBC driver following syntax may be used :

Class.ForName("sun.jdbc.odbc.JdbcOdbcDriver");

2. To register the third party driver one can use the method **registerMethod()** whose syntax is as follows :

DriverManager.registerDriver(Driver dr);

3. Use new to explicitly load the Driver class. This hard codes the driver and (indirectly) the name of the database into your program

4. The System class has a static Property list. If this has a Property jdbc.drivers set to a ':' separated list of driver class names, then all of these drivers will be loaded and registered automatically.

3)

```
import java.sql.*;
public class Student_JDBC {
public static void main(String args[])
{
     Connection con = null;
      Class.forName(sun.jdbc.odbc.JdbcOdbcDriver);
      Connection Conn = DriverManager.getConnection (jdbc:odbc:Access);
      Statement Stmt = Conn.createStatement();
      Statement Stmt2 = Conn.createStatement();
      String sql1 = "INSERT INTO STUDENT +  " (Std_id, name, course, ph_no)" +
          " VALUES (004, 'Sahil', 'MCA' " + "'SAHIL@rediffmail.com')";
       // Now submit the SQL....
      try
      {
      Stmt.executeUpdate(sql);
           String sql2 = "SELECT * FROM STUDENT";
            ResultSet results = Stmt2.executeQuery(sql);
             while (results.next())
              {
                    System.our.println("Std_id: " +  results.getString(1) +  " Name: " +
results.getString(2) +   ", course: " +   results.getString(3) +  “, ph_no:  “ +
results.getString(4));
              }
  }
// If there was a problem sending the SQL,  we will get this error.
```

```
    catch (Exception e)
    {
          System.out.println("Problem with Sending Query: " + e);
     }
    finally
    {
        result.close();
         stmt.close();
         stmt2.close();
         Conn.close();
    }
} // end of main method
} // end of class
```

Compile and execute this program to see the output of this program. The assumption is made that table named as STUDENT with the required columns are already existing in the MS-Access.

4) Type 4 (JDBC Net pure Java Driver) is the fastest JDBC driver. Type 1 and Type 3 drivers will be slower than Type 2 drivers (the database calls are make at least three translations versus two), and Type 4 drivers are the fastest (only one translation).

5) No. The JDBC-ODBC Bridge does not support multi threading. The JDBC-ODBC Bridge uses synchronized methods to serialize all of the calls that it makes to ODBC. Multi-threaded Java programs may use the Bridge, but they won't get the advantages of multi-threading.

6) The JDBC 3.0 API is the latest update of the JDBC API. It contains many features, including scrollable result sets and the SQL:1999 data types.

7) We are not allowed to use of the JDBC-ODBC bridge from an untrusted applet running in a browser, such as Netscape Navigator. The JDBC-ODBC bridge doesn't allow untrusted code to call it for security reasons. This is good because it means that an untrusted applet that is downloaded by the browser can't circumvent Java security by calling ODBC. As we know that ODBC is native code, so once ODBC is called the Java programming language can't guarantee that a security violation won't occur. On the other hand, Pure Java JDBC drivers work well with applets. They are fully downloadable and do not require any client-side configuration.

   Finally, we would like to note that it is possible to use the JDBC-ODBC bridge with applets that will be run in appletviewer since appletviewer assumes that applets are trusted. In general, it is dangerous to turn applet security off, but it may be appropriate in certain controlled situations, such as for applets that will only be used in a secure intranet environment. Remember to exercise caution if you choose this option, and use an all-Java JDBC driver whenever possible to avoid security problems.

8) There is one facility available to find out what JDBC calls are doing is to enable JDBC tracing. The JDBC trace contains a detailed listing of the activity occurring in the system that is related to JDBC operations.

   If you use the DriverManager facility to establish your database connection, you use the DriverManager.setLogWriter method to enable tracing of JDBC operations. If you use a DataSource object to get a connection, you use the DataSource.setLogWriter method to enable tracing. (For pooled connections, you use the ConnectionPoolDataSource.setLogWriter method, and for

connections that can participate in distributed transactions, you use the XADataSource.setLogWriter method.)

9) This problem can be caused by running a JDBC applet in a browser that supports the JDK 1.0.2, such as Netscape Navigator 3.0. The JDK 1.0.2 does not contain the JDBC API, so the DriverManager class typically isn't found by the Java virtual machine running in the browser.

To remove this problem one doesn't require any additional configuration of your web clients. As we know that classes in the java.* packages cannot be downloaded by most browsers for security reasons. Because of this, many vendors of all-Java JDBC drivers supply versions of the java.sql.* classes that have been renamed to jdbc.sql.*, along with a version of their driver that uses these modified classes. If you import jdbc.sql.* in your applet code instead of java.sql.*, and add the jdbc.sql.* classes provided by your JDBC driver vendor to your applet's codebase, then all of the JDBC classes needed by the applet can be downloaded by the browser at run time, including the DriverManager class.

This solution will allow your applet to work in any client browser that supports the JDK 1.0.2. Your applet will also work in browsers that support the JDK 1.1, although you may want to switch to the JDK 1.1 classes for performance reasons. Also, keep in mind that the solution outlined here is just an example and that other solutions are possible.

10) The ResultSet.getXXX methods are the only way to retrieve data from a ResultSet object, which means that you have to make a method call for each column of a row. There is very little chance that it can cause performance problem. However, because it is difficult to see how a column could be fetched without at least the cost of a function call in any scenario.

11) The classpath may be incorrect and Java cannot find the driver you want to use.

To set the class Path:

The CLASSPATH environment variable is used by Java to determine where to look for classes referenced by a program. If, for example, you have an import statement for my.package.mca, the compiler and JVM need to know where to find the my/package/mca class.

In the CLASSPATH, you do not need to specify the location of normal J2SE packages and classes such as java.util or java.io.IOException.

You also do not need an entry in the CLASSPATH for packages and classes that you place in the ext directory (normally found in a directory such as C:\j2sdk\jre\lib\ext). Java will automatically look in that directory. So, if you drop your JAR files into the ext directory or build your directory structure off the ext directory, you will not need to do anything with setting the CLASSPATH.

Note that the CLASSPATH environment variable can specify the location of classes in directories and in JAR files (and even in ZIP files).

If you are trying to locate a jar file, then specify the entire path and jar file name in the CLASSPATH. (Example: CLASSPATH=C:\myfile\myjars\myjar.jar).

If you are trying to locate classes in a directory, then specify the path up to but not including the name of the package the classes are in. (If the classes are in a package called my.package and they are located in a directory called

C:\myclasses\here\my\package, you would set the classpath to be CLASSPATH=C:\myclasses\classname).

The classpath for both entries would be CLASSPATH=C:\myfile\myjars\myjar.jar;C:\myclasses\here.

2.11 FURTHER READINGS/REFERENCES

- Bernard Van Haeckem, *Jdbc: Java Database Connectivity*, Wiley Publication
- Bulusu Lakshman, *Oracle and Java Development*, Sams Publications.
- Prateek Patel, *Java Database Programming*, Corollis
- Justin Couch and Daniel H. Steinberg, *Java 2 Enterprise Edition Bible*,Hungry Minds, Inc.
- Jaworski, *Java 2 Platform*, Techmedia

Reference websites:

- http://www.developers.sun.com
- www.javaworld.com
- www.jdbc.postgresql.org
- www.jtds.sourceforge.net
- www.en.wikipedia.org
- www.apl.jhu.edu
- http://www.learnxpress.com
- http://www.developer.com