JAVA DATABASE CONNECTIVITY

CHAPTER 32

Interconnection between two or more computers is called a 'network'. A network might consist of just two computers or two thousand computers. The advantage of a network is to send and receive information from one computer to another computer. A computer that receives information or some service through the network is called 'client' and the other computer which provides the required service is called 'server'. A network consists of many clients and servers.

Database Servers

A database is a repository of data. We can store data permanently in a database and retrieve it later whenever needed by using some query commands. Databases like Oracle, Sybase, MySQL and SqlServer are in use nowadays. Data is stored generally in the form of tables in these databases. To retrieve data from the tables and give it to the users, we need some program. This program is called 'database client' program. To understand the database client, let us take an example. If oracle10g is installed in your computer, go to 'Start' menu and click:

```
Start -> Programs -> Oracle Database 10g Express Edition -> Run SQL command
```

It opens a window at DOS prompt where SQL> prompt appears. Type connect and, it asks to enter user-name and password. Type 'scott' and 'tiger' as default values. Then you would be connected to the oracle database. Please see the Figure 32.1. This application is called `SQL*Plus` which is a client program to connect to the oracle server at any moment.

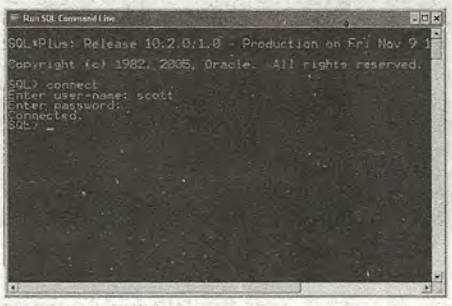

Figure 32.1 SQL*Plus client to connect to Oracle database

Database Clients

Using SQL*Plus application, it is possible to store data into oracle database. Also retrieving the data as well as updating it is also possible. Since, SQL*Plus is retrieving data from the oracle server in your system, SQL*Plus is called database client and oracle database is called database server. Now type a query as:

```
SQL> select * from tab;
```

It displays all the tables available in oracle database. Press Cntrol+Alt+Del to see Task Manager Dialog box as shown in Figure 32.2 below, where you can spot out Oracle.exe file running in your system. This is nothing but the oracle database. It is possible to load SQL*Plus client software in a system and the oracle database server program in another system connected on a network. The client will be able to connect to the database server and retrieve the data on the network. See Figure 32.5.

Figure 32.2 Oracle.exe running in the system

Similarly, MySQL is a database. To connect to this database, go to Start-> MySQL -> MySQL Server 5.0 -> MySQL Command line client. It asks for password, where you type: 'root' at DOS prompt. Now you can see the screen shown in Figure 32.3.

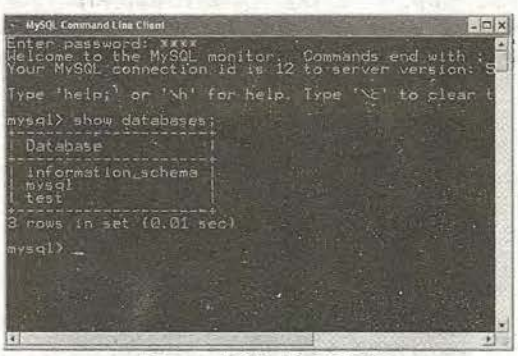

Figure 32.3 MySQL client

To see the names of databases in MySQL, give the following command:

```
mysql> show databases;
```

To come out of MySQL, we can type quit at `mysql>` prompt. We can understand that MySQL program is client program and the server program `mysqld-nt.exe` which runs internally is called MySQL database server. To see the MySQL server in your system, press Control+Alt+Del to display Task Manager Dialog as shown in Figure 32.4.

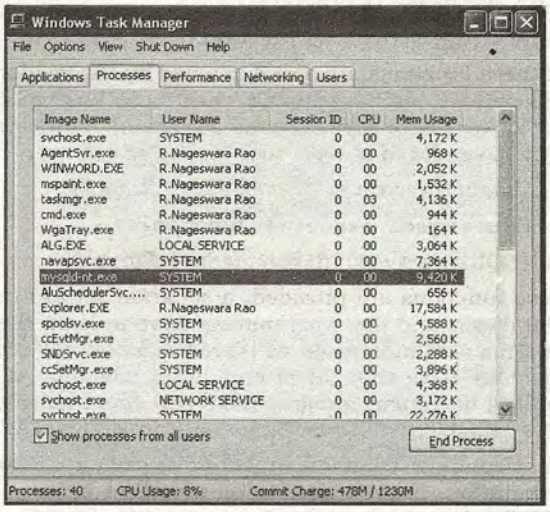

Figure 32.4 MySQL server running in the system

`SQL*Plus` or MySQL client programs connect to database servers and retrieve the data from them. The same thing can be done through a Java program which connects to a database and retrieves data from it. This technology is called Java Database Connectivity (JDBC).

JDBC (Java Database Connectivity)

JDBC is an API (Application Programming Interface) that helps a programmer to write Java programs to connect to a database, retrieve the data from the database and perform various operations on the data in the Java program.

Important Interview Question

What is JDBC?

JDBC (Java Database Connectivity) is an API that is useful to write Java programs to connect to any database, retrieve the data from the database and utilize the data in a Java program.

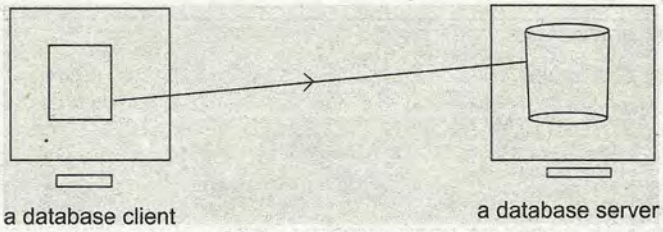

Figure 32.5 A database client and server

Every database vendor will provide a document representing all the commands to connect and utilize the database features. This document is called 'API (Application Programming Interface) document'. API document is a file that contains description of all the features of a software, a product or a technology. For example, the vendor of oracle database has given the following information for the programmers who use his product, in his API document:

- To connect to oracle database, use `oLog()` function.
- To execute a command, use `oexec()` function.
- To disconnect from oracle, use `oLogoff()` function.

The syntax and usage of these functions are also explained by the vendor in his API document. Any programmer wishing to use oracle database should learn these functions. This is not enough. If the same programmer wants to work with MySQL database, he should learn the syntax and usage of functions provided by MySQL vendor in his API document. For example,

- To connect to MySQL database, use `mysql_connect()` function.
- To execute a command, use `mysql_execute()` function.
- To disconnect from MySQL, use `mysql_disconnect()` function.

In the above manner these functions are provided, a separate set of functions are provided in API documents of different databases and the programmers have to learn them every time they want to use a different database. This is cumbersome as there are hundreds of functions to learn. This is the reason, software scientists have thought of creating a common API document. By using the functions of this common API document, programmers can communicate with any database in the world. Such a document is called ODBC (Open Database Connectivity) API.

ODBC is a document that contains common functions to communicate with any database. It is created by Microsoft Corporation. If any organization creates a software depending on this ODBC document, it is called ODBC driver.

In the same way, Sun Microsystems Inc. has also created an API document named JDBC (Java Database Connectivity) API and the actual software which is created according to JDBC API, is called JDBC driver. JDBC API is defined in java.sql package. This package contains interfaces like Connection, Statement, ResultSet, ResultSetMetaData, PreparedStatement, Driver, CallableStatement and classes like Date, Time, DriverManager etc. Several companies have started developing software containing these interfaces and classes. These softwares are called JDBC drivers. For example, `classes12.jar` is a driver developed by Oracle corporation using which we can connect to oracle database and communicate with it. Similarly `gate.jar` is a JDBC driver from `inet` to connect to oracle. Similarly, Mysql-connector-java-3.0.11-stable-bin.jar is a driver to connect to MySQL database. Also, we have `jdbc-odbc` driver provided by Sun Microsystems.

Working with Oracle Database

Oracle is the most popular database all over the world. After connecting to oracle database, SQL> prompt appears where you should enter SQL (Structured Query Language) commands. Let us see how to use SQL commands to do certain useful tasks on the Oracle database.

```
SQL> select * from tab;
TNAME                TABTYPE CLUSTERID
-------------------------------------------------------------
ACCOUNT              TABLE
DEPT                 TABLE
EMP                  TABLE
SALGRADE             TABLE

4 rows selected.
```

Suppose, we want to create a table with the name 'emptab' with 3 columns: eno (int type), ename (String type) and sal(float type), we can type SQL command as:

```
SQL> create table emptab(eno int, ename varchar2(20), sal float);
```

Table created.

It is seen that using the above SQL command we have created a table by the name of 'emptab', in which varchar2(20) represents that we can store a maximum of 20 characters in ename column. To see the information about the table created by us, we can use description command, as:

```
SQL> desc emptab;
Name          Null?          Type
-----------------------------------------------
ENO                          NUMBER(38);
ENAME                        VARCHAR2(20);
SAL                          FLOAT(126);
```

Let us store data into the table 'emptab' created by us. For this purpose, we should use 'insert' command as:

```
SQL> insert into emptab values(&eno, '&ename', &sal);
Enter value for eno: 1001
Enter value for ename: Venkat Rao
Enter value for sal: 8900.95
old 1: insert into emptab values(&eno, '&ename', &sal)
new 1: insert into emptab values(1001, 'Venkat Rao', 8900.95)
1 row created.
```

In this way, we inserted a row into oracle database. To enter another row, the above command can be repeated by typing '/' at SQL> prompt as:

```
SQL>/
Enter value fro eno: 1002
Enter value for ename: Gopal
Enter value for sal: 5600.55
```

To save all entered rows into the table, we can use 'commit' command, as:

```
SQL> commit;
```

To see all the rows of our emptab, we should use 'select' command in the following way:

```
SQL> select * from emptab;
ENO          ENAME                SAL
-----------------------------------------------------
1001         Venkat Rao           8900.95
1002         Gopal                5600.55
1003         Laxmi                5000
1004         Nitin Prakash  12000
```

In the above command, '*' represents all the columns of the rows. If we want only eno and ename columns, we can give select command, as:

```
SQL> select eno,ename from emptab;
ENO          ENAME
-------------------------------------------
1001         Venkat Rao
1002         Gopal
```

```
1003          Laxmi
1004          Nitin Prakash
```

In case, we want to retrieve the names of employees whose salary is more than Rs. 6000.00, we can give the following command:

```
SQL> select ename from emptab where sal>6000;

ENAME
-------------
Venkat Rao
Nitin Prakash
```

Let us now update the salary of an employee in emptab. Let us increase the salary by Rs. 1000 to the employee whose eno is 1002.

```
SQL> update emptab set sal= sal+1000 where eno=1002;
```

1 row updated.

If we want to delete an employee row whose eno is 1001:

```
SQL> delete from emptab where eno= 1001;
```

1 row deleted.

Now, we want to insert one row related to a new employee 'Mahendra' as:

```
SQL> insert into emptab values(9999, 'Mahendra', 13000);
```

1 row created.

To save all the changes into the database, we can use 'commit' command. Similarly, to un-save the changes, 'rollback' can be used.

```
SQL> rollback;
Rollback completed.
```

Now we will close the SQL client giving 'exit':

```
SQL> exit;
```

The commands we used are called SQL commands. The same commands can be used on any other database with slight modifications.

Working with MySQL Database

MySQL is a free database software provided by mysql.com. We can create only one database in oracle, but in MySQL, it is possible to create multiple databases. After going into MySQL, we can see mysql> prompt where we should enter commands. To see what databases are existing in MySQL give the following command:

```
mysql> show databases;
```

Database
mysql
test

The above output means that there are 2 databases currently
mysql and test. Let us see which tables are available in the dat
test database, as:

```
mysql> use test;
Database changed
```

Now, to see the tables in test database,

```
mysql> show tables;
```

Tables_in_test
emp
mytab
student

Now let us create a table with the name 'emptab' in test datal
columns in emptab.

```
mysql> create table emptab(eno int, ename char(20)
```

Query OK, 0 rows affected
Let us see the description of emptab by giving 'desc' command.

```
mysql> desc emptab;
```

Field	Type	Null	Key
eno	int(11)	YES	
ename	char(20)	YES	
sal	float	YES	

3 rows in set
We did not store any data so far into the emptab. Let us now store

```
mysql> insert into emptab values(1001, "Nagesh", 7
```

Query OK, 1 row affected

```
mysql> insert into emptab values(1002, "Ganesh", 1
```

Query OK, 1 row affected
In this way, we can store some rows into emptab. To see all the r
query as:

```
mysql> select * from emptab;
```

eno	ename	sal
1001	Nagesh	7800
1002	Ganesh	10000
1003	Vinaya	5000.55
1004	Rahul	7800

4 rows in set

Suppose we want to retrieve employees names whose salary is Rs. 7800.

```
mysql> select ename from emptab where sal = 7800;
```

ename
Nagesh
Rahul

2 rows in set

Suppose we want to delete the row of an employee whose employee number is 1001.

```
mysql> delete from emptab where eno = 1001;
```

Query OK, 1 row affected

To update the salary to Rs. 15000 of an employee whose name is "Rahul",

```
mysql> update emptab set sal= 15000 where name= "Rahul";
```

Query OK, 1 row affected

Rows matched: 1 changed: 1 warnings: 0

Finally, to close MySQL, we can give 'quit' command to see the system prompt, as:

```
mysql> quit;
```

Bye

In the similar manner, we can work with Sybase, SQL Server or MSAccess databases. We can give commands to create tables, store data in the tables as well as retrieve data from the tables and process data.

Without the help of database clients, it is possible to connect to a database server by using a Java program as its client. This is where JDBC API plays significant role.

Stages in a JDBC Program

The following stages are used by Java programmers while using JDBC in their programs:

- **Registering the driver:** A database driver is a software containing classes and interfaces written according to JDBC API. Since there are several drivers available in the market, we

should first declare a driver which is going to be used for communication with the data base server in a Java program.

- **Connecting to a database:** In this stage, we establish a connection with a specific database through the driver which is already registered in the previous step.
- **Preparing SQL statements in Java:** We should create SQL statements in our Java program using any of the interfaces like Statement, PreparedStatement, CallableStatement, etc., which are available in `java.sql` package.
- **Executing the SQL statements on the database:** For this purpose, we can use `execute()`, `executeQuery()` and `executeUpdate()` methods of Statement interface.
- **Retrieving the results:** The results obtained by executing the SQL statements can be stored in an object with the help of interfaces like ResultSet, ResultSetMetaData and DatabaseMetaData.
- **Closing the connection:** We should close the connection between the Java program and the database by using `close()` method of Connection interface.

Registering the Driver

This is the first step to connect to a database. A programmer should specify which database driver he is going to use to connect to the database. There are 4 ways to register a driver.

- By creating an object to driver class of the driver software, we can register the driver. For example, to register JdbcOdbcDriver of the Sun Microsystems, we can create an object to the driver class: JdbcOdbcDriver, as shown below:

```
sun.jdbc.odbc.JdbcOdbcDriver obj = new sun.jdbc.odbc.JdbcOdbcDriver();
```

- The second way to register a driver is by sending the driver class object to `registerDriver()` method of DriverManager class. For example, the object of JdbcOdbcDriver class can be passed to `registerDriver()` method, as:

```
DriverManager.registerDriver( new sun.jdbc.odbc.JdbcOdbcDriver());
```

- The third way to register the driver is to send the driver class name directly to `forName()` method, as:

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

- In case, the user should specify the driver name at the time of running the program, we can use `getProperty()` method of System class to receive the driver name, as:

```
String dname = System.getProperty("driver");
Class.forName(dname);
```

Here, `getPropert()` method accepts the driver name from the user and stores it in dname. Then `forName()` method creates an object to the class whose name is in dname. The question is how to provide the driver name while running the program? The following syntax will make it clear:

```
C:\> java -Ddriver = driverclassname Programname
```

For example, to send the driver class name: JdbcOdbcDriver to Myprog, we can type at system prompt, as:

```
C:\> java -Ddriver = sun.jdbc.odbc.JdbcOdbcDriver  Myprog
```

In this case, the name: `sun.jdbc.odbc.JdbcOdbcDriver` is sent to `System.getProperty()` method and then `Class.forName()` method will create an object to it.

Important Interview Question

What is a database driver?

A database driver is a set of classes and interfaces, written according to JDBC API to communicate with a database.

How can you register a driver?

To register a database driver, we can follow one of the 4 options:

- By creating an object to driver class

- By sending driver class object to DriverManager.registerDriver() method.

- By sending the driver class name to Class.forName() method.

- By using System class getProperty() method.

Connecting to a Database

To connect to a database, we should know 3 things:

- **URL of the database:** Here, URL(Uniform Resource Locator) represents a protocol to connect to the database. Simply speaking it locates the database on the network.
- **Username:** To connect to a database, every user will be given a username which is generally allotted by the database administrator.
- **Password:** This is the password allotted to the user by the database administrator to connect to the database.

Let us take an example. To connect to oracle database using Sun Microsystems' jdbc-odbc driver, we can write the following statement:

```
DriverManager.getConnection("jdbc:odbc:oradsn", "scott", "tiger");
```

Here, "`jdbc:odbc:oradsn`" is the URL. oradsn represents the DSN (data source name), a name given to the database for the reference in the Java program. "scott" is the username and "tiger" is password.

Let us take another example. To connect to oracle database using the thin driver provided by Oracle corp, we can write the following statement:

```
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:oracle", "scott",
"tiger");
```

We advise you to go through the user manuals supplied by the database vendors where they explain clearly the ways to connect to the database with examples.

Important Interview Question

What is DSN?

Data Source Name (DSN) is a name given to the database to identify it in the Java program. The DSN is linked with the actual location of the database.

Preparing SQL Statements

We need SQL(Structured Query Language) statements which are useful to make different operations like adding data to the database, updating the data of the database and deleting un-useful data from the database and also retrieving the data from the database. SQL statements can be classified into two types: select and non-select statements.

- **select statements:** These statements help to retrieve the data from the database in the form of rows. For example, to retrieve all the rows and with all columns, we can write:

```
select * from emptab;
```

To pass this type of statements to the database, first of all we should create Statement object as:

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

We should pass the SQL statement to `executeQuery()` method so that it executes the query on the database and gets back the resultant rows.

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

Now the results are available in ResultSet object rs. We can retrieve the results from rs using the methods of ResultSet interface. See table 32.1.

Table 32.1

ResultSet method	Its function
1. boolean next()	This method moves the cursor to the next row in the ResultSet object.
2. int getRow()	Returns the current row number.
3. String getString()	Retrieves a string from the row of the ResultSet.
4. int getInt()	Retrives an integer value from the row of the ResultSet.
5. float getFloat()	Retrieves a float value from the row of the ResultSet.
6. double getDouble()	Retrives a double value from the row of the ResultSet.
7. long getLong()	Retrives a long value from the row of the ResultSet.
8. Date getDate()	Retrives an object from the ResultSet row as java.sql.Date class object.

Important Interview Question

What is ResultSet?

ResultSet is an object that contains the results (rows) of executing a SQL statement on a database.

- **non-select statements:** These statements represent all other statements except select statements. create, update, insert, delete, etc., statements come under this category.

For example, to create the table mytab, we can use create statement as:

```
create table mytab(col1 number, col2 number);
```

To execute the above statement, we should first create Statement object, as:

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

And then, we should pass the SQL statement to `executeUpdate()` method, as:

```
int n = stmt.executeUpdate("create table mytab(col1 number, col2 number)");
```

In the above statement, n represents the number of rows updated in the table. Let us take another example.

```
int n = stmt.executeUpdate("update emptab set sal= 15000 where eno = 1005");
```

Here, we are updating the salary of an employee whose eno is 1005. There is only 1 row in emptab with eno value 1005. So, only 1 row is updated by the above query. So, n value becomes 1.

```
int n = stmt.executeUpdate("delete emptab where eno > 1005");
```

Here, all the rows with eno values greater than 1005 from emptab will be deleted. Now n value represents how many such rows are deleted.

Let us write a Java program to connect to oracle database and retrieve the rows from emptab. In this program, we are using Oracle 10g xe version to create the database table. If you refer to Oracle 10g documentation (user manual), you can find the following URL to be used to connect to the database:

```
jdbc:oracle:thin:@localhost:1521:xe
```

Let us take a table emptab with 3 columns which can be created with the SQL command:

```
create table emptab(eno int, ename varchar2(20), sal float);
```

Insert some rows into the table emptab. Now let us think about retrieving the rows of the table.

Program 1: This is a program to retrieve all the rows from emptab of oracle database.

```
//To retrieve data from Oracle database
import java.sql.*;
class OracleData
{
      public static void main(String args[]) throws Exception
      {
            //Register the driver
            DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());

            //Establish connection with the database
            Connection con=DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");

            //Create a SQL statement
            Statement stmt = con.createStatement();

            //Execute the stmt
            ResultSet rs = stmt.executeQuery("select * from emptab");
```

```
                //all rows of emptab are in rs. Now retrieve column data
                //from rs and display
                while(rs.next()){
                System.out.println(rs.getInt(1));
                System.out.println(rs.getString(2));
                System.out.println(rs.getFloat(3));
                System.out.println("====================");
                }

                //close the connection
                con.close();
        }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac OracleData.java
C:\> java OracleData
1001
Nageswara Rao
7800.55
====================
1002
Vijay Kumar
6000.0
====================
1003
Durga
5000.75
====================
1004
Ganesh
12000.77
====================
```

Observe the following statement in the above program:

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

When this statement is executed, all the rows of the emptab will be retrieved and stored into the ResultSet object rs. Here, 'rs' works like a reference which is positioned initially just before the first row, as shown in Figure 32.6.

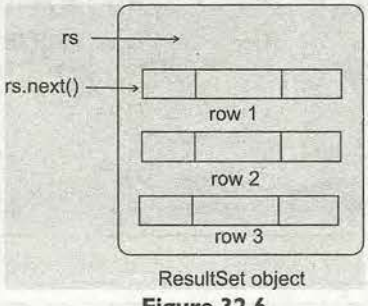

Figure 32.6

To position rs at first row, we can use `rs.next()` method. This method puts the reference rs at first row and returns true. If row is not available in the ResultSet object, then it returns false. By using `rs.next()` method in a loop, we can visit every row of the ResultSet object.

Again, we know that each row contains 3 columns:

- eno is int type
- ename is String type
- sal is float type

To retrieve the data from these columns, we can use getXXX() methods, as:

- `rs.getInt(1);` //here, 1 represents 1st column data to retrieved as int type
- `rs.getString(2);` //retrieve 2nd column data as String type
- `rs.getFloat(3);` //retrive 3rd column data as float type

Observe that, before compilation of the program, we set the classpath to `C:\jars\ojdbc14.jar`. This file represents the driver software supplied by Oracle corp. When we install Oracle 10g xe version, this `ojdbc14.jar` file can be found in the directory: `C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib`. The driver .jar file is copied into a separate directory `C:\jars`, and hence we set the classpath as shown above.

Program 2: In this program, we insert two rows into emptab. The first row is inserted with only one column: eno value 777, as:

```
insert into emptab(eno) values(777);
In this case, a row with eno value 777, ename value null and sal value 0.0 will
be inserted into emptab. We insert another row with all column values, as:
insert into emptab values(779, 'Satyaraj', 5000.00);
//This program demonstrates how to insert rows into a table
import java.sql.*;
class Insertion
{
      public static void main(String args[])throws Exception
      {

            //register oracle driver
            DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());

            //get a connection with the database
            Connection con;
            con = DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");

            //create a statement to insert a row with only eno value as 777
            Statement stmt = con.createStatement();
            int norows = stmt.executeUpdate("insert into emptab(eno)
            values(777)");
            System.out.println("no of rows affected = "+ norows);

            //insert a row with eno, ename and sal values
            norows = stmt.executeUpdate("insert into emptab
            values(779,'Satyaraj',5000.00)");
            System.out.println("no of rows affected = "+ norows);

            //close connection
            con.close();
      }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;

C:\> javac Insertion.java
C:\> java Insertion
no rows affected: 1
```

```
no rows affected: 1
```

Program 3: This is a program to understand how to update and delete rows from emptab in oracle database. We use the SQL statements in the following way:

```
update emptab set sal=30000 where eno>1001;
```

Here, we are storing 30000 into the salary of all the employees whose employee numbers are greater than 1001. Similarly, we use:

```
delete emptab where eno>1001;
```

This will deleted all the rows from the table whose employee numbers are greater than 1001.

```
// This program demonstrates how to update/delete rows
import java.sql.*;
class Updation
{
  public static void main(String args[])throws Exception
  {
   //accept the driver class name from system prompt into dname
   String dname = System.getProperty("driver");

   //create an object to the driver class whose name is in dname
   Class.forName(dname);

   //connect to oracle database
   Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
"tiger");

   //create SQL statement
   Statement stmt = con.createStatement();

   //executed SQL statement to update salary
   int norows = stmt.executeUpdate("update emptab set sal=30000 where
eno>1001");
   System.out.println("no of rows updated = "+ norows);

   System.out.println("Press a key to continue...");
   System.in.read();

   //execute SQL statement to delete a row
   norows = stmt.executeUpdate("delete emptab where eno>1001");
   System.out.println("no of rows deleted = "+ norows);

   //close connection
   con.close();
  }
}
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac Updation.java
C:\> java -Ddriver=orcle.jdbc.driver.OracleDriver Updation
no rows updated: 1
Press a key to continue...

no rows deleted: 2
```

Observe that in the above program, we are passing the driver name from system prompt. And the System class's `getProperty()` method is used to access it.

Using jdbc-odbc Bridge Driver to Connect to Oracle Database

Sun Microsystems Inc. provides a default driver called jdbc-odbc bridge driver along with Java software that is useful to connect to any database. Let us see how to use this driver with oracle database.

Step 1. First of all, we should create a DSN (Data Source Name) that represents the name for the database, by clicking on the Start button and following the path:

```
Start -> Settings -> Control Panel -> Administrative Tools -> Data Sources
(ODBC).
```

We can find ODBC Data Source Administrator dialog box as shown in Figure 32.7.

Step 2. In this dialog box, select User DSN tab and click the Add button at right side. Now the list of drivers available will be displayed in a separate dialog box. Select Microsoft ODBC for Oracle driver in the list and click the Finish button. See Figure 32.8.

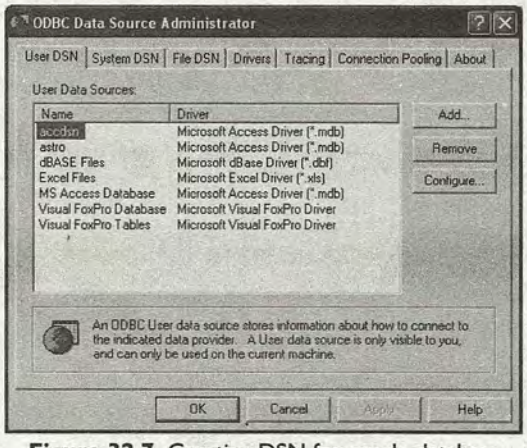

Figure 32.7 Creating DSN for oracle database

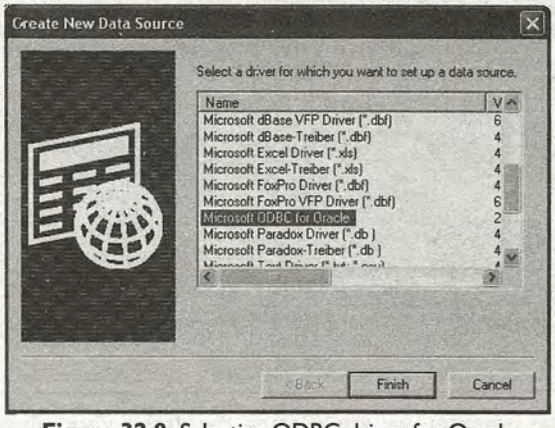

Figure 32.8 Selecting ODBC driver for Oracle

Step 3. Now, you can see Microsoft ODBC for Oracle Setup dialog box appears. Type the DSN name and user name and then click the OK button, as shown in Figure 32.9.

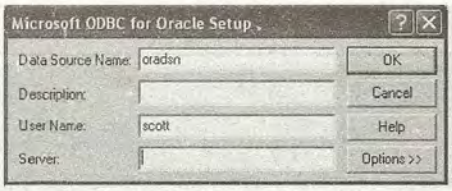

Figure 32.9 DSN and User Name

Now, we can verify that a new DSN with the name 'oradsn' has been created in UserDSN tab. Click the OK button to close the ODBC Data Source Administrator.

After creating the DSN name as shown above, the next thing we should do is to register the driver. To register Sun's jdbc-odbc bridge driver, we can use the following statement in our program:

```
DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
```

To connect to the database, using this driver, we can write:

```
Connection con=DriverManager.getConnection(
"jdbc:odbc:oradsn","scott","tiger");
```

Here, oradsn is the DSN name that we have just created. scott is the username and tiger is the password.

Program 4: Let us see how to use Sun's jdbc-odbc bridge driver to connect to oracle database and retrieve all the rows from emptab. Now we need not use `ojdbc14.jar` since it is another driver called by the name 'thin driver'.

```
//To retrieve data from Oracle database using jdbc-odbc bridge driver
import java.sql.*;
class OracleData
{
     public static void main(String args[]) throws Exception
     {
          //Register the driver
          DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());

          //Establish connection with the database
          Connection con=DriverManager.getConnection(
          "jdbc:odbc:oradsn","scott","tiger");

          //Create a SQL statement
          Statement stmt = con.createStatement();

          //Execute the stmt
          ResultSet rs = stmt.executeQuery("select * from emptab");

          //all rows of emptab are in rs. Now retrieve column data
          //from rs and display
          while(rs.next()){
          System.out.println(rs.getInt(1));
          System.out.println(rs.getString(2));
          System.out.println(rs.getFloat(3));
          System.out.println("====================");
          }

          //close the connection
          con.close();
     }
}
```

Output:

```
C:\> javac OracleData.java
C:\> java OracleData

1001
Nageswara Rao
7800.55
======================
1002
Vijay Kumar
6000.0
======================
1003
Durga
5000.75
======================
1004
Ganesh
12000.77
======================
```

Retrieving Data from MySQL Database

Let us open MySQL database and see which databases are there, by typing 'show databases' command, as:

```
mysql> show databases;
```

Database
mysql
test

The above output means that there are 2 databases currently available in MySQL and they are: mysql and test. Let us first move into test database, as:

```
mysql> use test;
```

Database changed

Create a table emp now using create statement in the database 'test', as:

```
mysql> create table emp(eno int, ename char(20), sal float);
```

Then enter some rows into the emp using insert statement as:

```
mysql> insert into emp values(100, "Sirisha", 5000.55);
```

Create and store some rows into the database as shown in preceding steps, and finally give commit command to save all these transactions.

```
mysql> commit;
```

Now, let us write a Java program to see how to retrieve all the rows from the emp of test database that is available in MySQL database. `mysql.com` company provides a driver `mysql.jar` along with

a user manual where they provide help regarding how to register their driver. Register the mysql driver in the following manner:

```
DriverManager.registerDriver(new com.mysql.jdbc.Driver());
```

To connect to the database, we should use:

```
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/test?user=root&password=student");
```

Here, user name is 'root' and password is 'student'.

Program 5: In this program, we retrieve the data from the table emp of test database.

```
//To retrieve data from MySQL database
import java.sql.*;
class MysqlData
{
      public static void main(String args[])throws Exception
      {
            //Register the driver
            DriverManager.registerDriver(new com.mysql.jdbc.Driver());

            //Establish connection
            Connection con=DriverManager.getConnection(
            "jdbc:mysql://localhost:3306/test?user=root&password=student");

            //Create a SQL statement
            Statement stmt = con.createStatement();

            //Execute the stmt
            ResultSet rs = stmt.executeQuery("select * from emp");

            //retrieve from ResultSet and display column data
            while(rs.next()){
            System.out.println(rs.getInt(1));
            System.out.println(rs.getString(2));
            System.out.println(rs.getFloat(3));
            System.out.println("====================");
            }

            //close connection
            con.close();
      }
}
```

Output:

```
C:\> set classpath=c:\jars\mysql.jar;.;
C:\> javac MysqlData.java
C:\> java MysqlData
100
Sirisha
5000.55
====================
101
Nataraj
9000.90
====================
```

Retrieving Data from MS Access Database

Microsoft Access is also one of the databases used by many organizations. We can create a database in MS Access and create tables in the database. For this, select the options:

```
Start -> Programs -> Microsoft Office -> Microsoft Access -> Create a new file -
> Blank database
```

File New Database dialog box will be displayed as shown in Figure 32. 10. Type the database name as 'Mydb' in a directory, for example, D:\rnr. Then click the Create button.

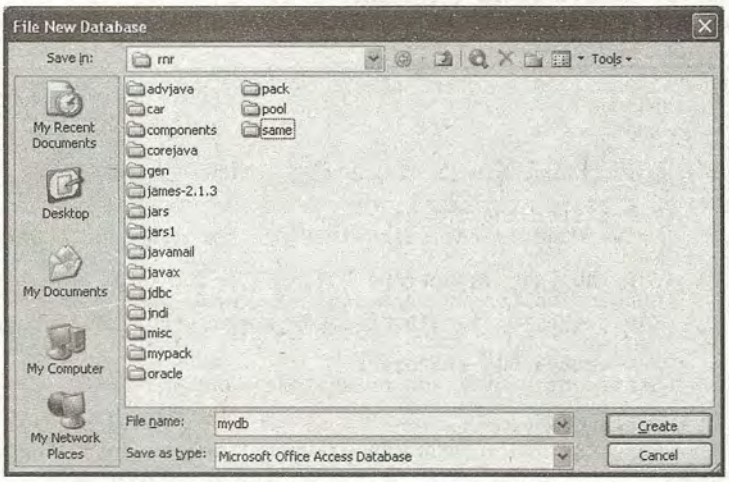

Figure 32.10 Creating a database in MS Access

When we enter the MS Access, a dialog box is displayed where double click 'Create table by entering data' option. This is shown in Figure 32.11.

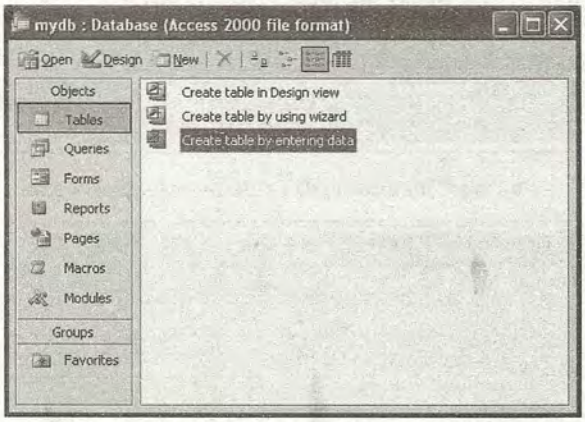

Figure 32.11 Creating a table in MS Access

Then, we will see a spread sheet with several rows and columns where we can enter column names in the table and column data as shown in Figure 32.12. Rename Field1 as eno, Field2 as ename and Field3 as sal. Then enter data.

Figure 32.12 Entering data into the table in MS Access

Then select File -> Save option and Enter Filename as 'emptab' and then click OK button.

MS Access prompts us to enter a primary key. Click on 'NO' button. And close the table. Come out of MS Access now.

To connect to MS Access, we can use Sun Microsystems' jdbc-odbc bridge driver. To use this driver, it is necessary to create a DSN (Data Source Name) by going to Start -> Settings -> Control Panel -> Administrative Tools -> Data Sources (ODBC).

We can find ODBC Data Source Administrator dialog box where select User DSN tab and click the Add button at right side. Now the list of drivers available will be displayed in a separate dialog box. Select Microsoft Access (`*.mdb`) driver in the list and click Finish button. It displays ODBC Microsoft Access Setup dialog box as shown in Figure 32.13. In this box, enter:

```
Data Source Name: accdsn
```

Then click the Select button to select the database. Then select the directory with the name: `D:\rnr` where `mydb.mdb` will appear. Click on that directory. Then click the OK button and you will come back to ODBC Microsoft Access Setup dialog box where you click OK button and close it.

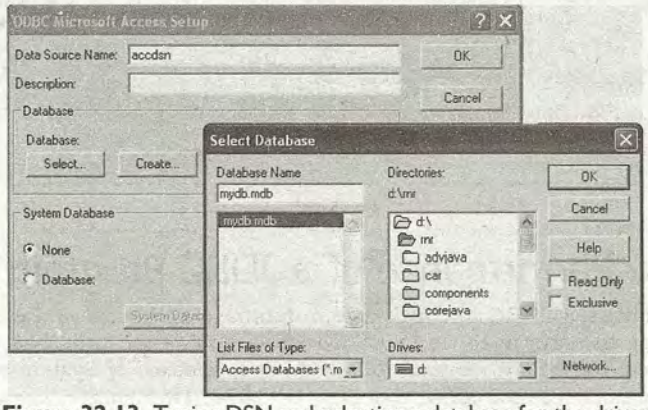

Figure 32.13 Typing DSN and selecting a database for the driver

Now DSN is created, we can mention it while connecting it to MS Access database, as:

```
Connection con= DriverManager.getConnection("jdbc:odbc:accdsn","","");
```

Here, no need to give username and password. So, we just used "" and "" in these places.

Program 6: In this program, we connect to MS Access database and retrieve the data from emptab.

```
//To retrieve data from MSAccess database
import java.sql.*;
class AccessData
{
      public static void main(String args[])throws Exception
      {
            //Register the driver
            DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());

            //Establish connection
            Connection con=
            DriverManager.getConnection("jdbc:odbc:accdsn","","");

            //Create a SQL statement
            Statement stmt = con.createStatement();

            //Execute the stmt
            ResultSet rs = stmt.executeQuery("select * from emptab");

            //retrieve from ResultSet and display column data
            while(rs.next()){
            System.out.println(rs.getInt(1));
            System.out.println(rs.getString(2));
            System.out.println(rs.getFloat(3));
            System.out.println("====================");
            }
      }
}
```

Output:

```
C:\> javac AccessData.java
C:\> java AccessData
1001
Nageswara Rao
8900.0
====================
1002
Vijaya
9000.9
====================
1003
Manoj Kumar
12000.75
====================
```

Improving the Performance of a JDBC Program

A Java program that uses JDBC to connect to a database and retrieve the data from the databa may take some time to perform these tasks. To know how much time has been taken by a JDB program, we can take the help of `currentTimeMillis()` method of System class. This method giv the current time in milliseconds since January 1st 1970. By using `currentTimeMillis()` in t beginning of a program and again at the end the program, we can measure the time at t beginning and end of the program. The difference in these times gives the execution time of t program in milliseconds.

By reducing the execution time of a JDBC program, we can improve its performance. The executi time of a JDBC program depends on the following factors:

- The driver used to connect to the database will influence the execution time. Each driver will exhibit different performance and hence, selecting a good driver that gives optimum performance will improve the performance of a JDBC program.
- `setFetchSize()` method of Statement interface is useful to tell the driver how many rows at a time to be fetched from the database. By increasing the number of rows to be retrieved at a time, it is possible to improve the performance of a JDBC program.
- Sometimes, PreparedStatement interface can be used in place of Statement interface for improving the performance.

(a) Affect of Driver

Different vendors provide different drivers to be used in a JDBC program. The performance of these drivers will be different and hence the programmer should first test each driver and adopt a driver which gives better performance for his database.

Let us create a table in oracle database with the name mytab, with 2 columns as:

```
create table mytab(a int, b int);
```

The following program will help us to insert a total of 999 rows into mytab. This program uses oracle driver which is called 'thin' driver supplied by Oracle corp.

Program 7: In this program, we use 'thin' driver to connect to oracle database. We insert a total of 999 rows into the database and find the time for this insertion operation.

```
//To store 999 rows into Oracle database - Oracle driver
import java.sql.*;
class Performance
{
      public static void main(String args[]) throws Exception
      {
            //Register the driver
            DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());

            //Establish connection with the database
            Connection con=DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");

            //Create a SQL statement
            Statement stmt = con.createStatement();

            //count the time before insertion
            long t1 = System.currentTimeMillis();

            //insert 999 rows into mytab
            for(int i=1; i<1000; i++)
            stmt.executeUpdate("insert into mytab values("+i+","+i+")");

            //count the time after insertion
            long t2 = System.currentTimeMillis();

            //display the time taken
            System.out.println("Time = "+ (t2-t1));

            //close the connection
            con.close();
      }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac Performance.java
C:\> java Performance
Time = 6203
```

This means, 'thin' driver has taken 6203 milliseconds of time to insert 999 rows into mytab. This time may vary from system to system. Please observe the following looping statement to insert the rows in the above program:

```
for(int i=1; i<1000; i++)
     stmt.executeUpdate("insert into mytab values("+i+","+i+")");
```

Here, we have taken

```
insert into mytab values(
```

as a string. Then we attach i value as a string to it. So, we wrote +i. Then we attach a comma using +",". Then we attach again another i value as +i. Then a simple brace is attached by writing +")". So the total statement would yield a form like:

```
insert into mytab values(1,1);   //when i is 1.
```

Now, let us write the same program using Sun's jdbc-odbc bridge driver. For this purpose, we should create DSN as oradsn.

Program 8: In this program, we use 'jdbc-odbc' driver to connect to oracle database and find the time for inserting 999 rows.

```
//To store 999 rows into Oracle database - jdbc-odbc driver
import java.sql.*;
class Performance
{
     public static void main(String args[]) throws Exception
     {
          //Register the driver
          DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());

          //Establish connection with the database
          Connection con=DriverManager.getConnection(
          "jdbc:odbc:oradsn","scott","tiger");

          //Create a SQL statement
          Statement stmt = con.createStatement();

          //count the time before insertion
          long t1 = System.currentTimeMillis();

          //insert 999 rows into mytab
          for(int i=1; i<1000; i++)
          stmt.executeUpdate("insert into mytab values("+i+","+i+")");

          //count the time after insertion
          long t2 = System.currentTimeMillis();

          //display the time taken
          System.out.println("Time = "+ (t2-t1));

          //close the connection
          con.close();
     }
```

```
}
```

Output:

```
C:\> javac Performance.java
C:\> java Performance
Time = 7125
```

Please observe the time taken by jdbc-odbc driver. It is taking more time than the thin driver. So, thin driver is faster and hence offering better performance than jdbc-odbc driver.

Important Interview Question

Will the performance of a JDBC program depend on the driver?

Yes, each driver offers a different performance.

(b) Affect of setFetchSize()

`setFetchSize()` method of Statement interface is useful to fetch a number of rows at a time from the database. `setFetchSize(1)` will make the driver fetch only one row at a time from the database. If we specify `setFetchSize(100)` then the driver will be able to fetch 100 rows at a time from the database and hence the time of execution will be less.

Program 9: In this program, we use `setFetchSize()` method to retrieve only one row at a time from the database. We calculate the time taken to retrieve all the 999 rows from mytab.

```
//To retrieve 999 rows from Oracle database one at a time.
import java.sql.*;
class Performance1
{
      public static void main(String args[]) throws Exception
      {
            //Register the driver
            DriverManager.registerDriver(new
            oracle.jdbc.driver.OracleDriver());

            //Establish connection with the database
            Connection con=DriverManager.getConnection(
            "jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");

            //Create a SQL statement
            Statement stmt = con.createStatement();

            //retrieve at a time 1 row only
            stmt.setFetchSize(1);

            //count the time before retrieval
            long t1 = System.currentTimeMillis();

            //retrieve all rows from mytab
            ResultSet rs = stmt.executeQuery("select * from mytab");

            //display all the rows
            while(rs.next())
            System.out.println(rs.getInt(1)+"\t"+rs.getInt(2));

            //count the time after retrieval
            long t2 = System.currentTimeMillis();

            //display the time taken
            System.out.println("Time = "+ (t2-t1));
```

```
                //close the connection
                con.close();
            }
        }
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac Performance1.java
C:\> java Performance1
Time = 5547
```

This means `setFetchSize(1)` has taken 5547 milliseconds of time to retrieve all the rows of mytab. In the above program, change the `setFetchSize()` method to retrieve 100 rows at a time, as shown below:

```
stmt.setFetchSize(100);
```

Then re-run the program, to see the

```
Time = 2032
```

This means when we retrieve 100 rows at a time from the database, the time taken by the program is considerably reduced and we are able to achieve better performance.

Do not be under the impression that the more the number of rows are fetched the less the time of execution it takes. No. Please observe the Table 32.2 to understand how the number of rows in `setFetchSize()` will affect the execution time. The execution time may vary on different systems.

Table 32.2

setFetchSize()	Time taken in milliseconds
setFetchSize(1);	5547
setFetchSize(100);	2032
setFetchSize(150);	2016
setFetchSize(200);	2008
setFetchSize(250);	1984
setFetchSize(300);	2047
setFetchSize(350);	2068

From the above table, one can understand that the time taken is minimum when `setFetchSize()` has retrieved between 200 and 250 rows at a time. This value should be used by the programmer in his JDBC program to get maximum performance. When `setFetchSize()` is retrieving more than 300 rows, why the time is more again? The reason is: to retrieve those many rows, the driver has to allocate a lot of memory and fill the memory every time it retrieves the rows. This would take more time. So, `setFetchSize()` with too higher values may reduce the performance.

(c) Affect of PreparedStatement

Using PreparedStatement in place of Statement interface will improve the performance of a JDBC program. Let us first write a program using Statement interface to insert 999 rows into mytab and find how much time it is taking.

Program 10: In this program, we use Statement object to insert 999 rows into mytab.

```
//Using Statement
import java.sql.*;
public class Performance2
{
      public static void main(String args[]) throws Exception
      {

      //register the driver
      DriverManager.registerDriver(new
      oracle.jdbc.driver.OracleDriver());

      //establish the connection
      Connection con =

      DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
      "tiger");

      //create Statement object
      Statement stmt = con.createStatement();

      //calculate time before insertion
      long t1 = System.currentTimeMillis();

      //insert 999 rows into mytab
      for(int i=1; i<1000; i++)
      {
            stmt.executeUpdate("insert into mytab values("+i+","+i+")");
      }

      //calculate time after insertion
      long t2 = System.currentTimeMillis();

      System.out.println("Time= "+ (t2-t1));

      //close connection
      con.close();
      }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac Performance2.java
C:\> java Performance2
Time = 6312
```

When a SQL statement is sent to database, the following tasks are performed:

- The SQL statement's syntax should be verified to know whether it is correct or not. The SQL statement is divided into small pieces called 'tokens'. These tokens are also verified to know whether they are in SQL format or not.
- Then another verification is done to know whether the table or view mentioned in the statement exists or not.

The above two stages are called 'parsing' and takes some time. When a SQL statement is executed, Statement interface does parsing every time the statement is executed. In the above program, the

for loop is executed 999 times and a total of 999 rows are inserted into the table. So, Statement interface will conduct parsing for 999 times. This takes more time.

On the other hand, if we use PreparedStatement, it does parsing only once. Since the same statement is repeated 999 times, parsing is enough if one only once. This saves time and hence PreparedStatement offers better performance.

Let us see how to use PreparedStatement. First of all, we should create PreparedStatement object using `prepareStatement()` method of Connection interface.

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

Here, ? represents the value to be passed to the columns of the row. The first ? represents the value to be passed to first column and the second ? represents the value for second column. These values can be passed usig setXXX() methods, as:

```
stmt.setInt(1, i); //replace first ? with i value
stmt.setFloat(2, 15.5f);  //replace second ? with 15.5
```

Program 11: Let us now re-write the previous program using PreparedStatement and insert 999 rows into mytab.

```
// Demonstrates how to use a PreparedStatement
import java.sql.*;
public class Performance3
{
      public static void main(String args[]) throws Exception
      {

      //register the driver
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

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

      //create PreparedStatement object
      PreparedStatement stmt = con.prepareStatement("insert into mytab
      values(?,?)");

      //calculate time before insertion
      long t1 = System.currentTimeMillis();

      //insert 999 rows into mytab
      for(int i=1; i<1000; i++)
      {
            //set values for ?,?
            stmt.setInt(1,i);
            stmt.setInt(2,i);

            //execute the statement
            stmt.executeUpdate();
      }
      //calculate time after insertion
      long t2 = System.currentTimeMillis();

      System.out.println("Time= "+ (t2-t1));

      //close connection
      con.close();
      }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac Performace3.java
C:\> java Performance3
Time = 2828
```

Please observe the execution time when PreparedStatement is used. You will find that the time is less as compared to that of Statement. Remember, using PreparedStatement improves performance when we want to execute the same statement repeatedly.

Important Interview Question

What is parsing?

Parsing represents checking the syntax and grammar of a statement as a whole and also word by word.

What is the difference between Statement and PreparedStatement?

Statement parses a statement before its execution on the database. This parsing is done every time the statement is executed, and hence it may take more time when the same statement gets executed repeatedly. PreparedStatement conducts parsing only once when the same statement is executed repeatedly and hence it gives better performance.

Here, two more examples of using PreparedStatement is furnished:

```
1. PreparedStatement stmt = con.prepareStatement("update emptab set sal = ?
where eno= ? ");
stmt.setFloat(1, 10000.55f); //set 10000.55 to first ?
stmt.setInt(2,1002);  //set 1002 to second ?
int n = stmt.executeUpdate();  //n gives no. of rows affected
```

```
2. PreparedStatement stmt = con.prepareStatement("select * from emptab");
ResultSet rs = stmt.executeQuery();  //rs contains rows of emptab
```

Stored Procedures and CallableStatement

Stored procedure is a set of statements written using PL/SQL (Procedural language/ Structured Query language). To perform some calculations on the tables of a database, we can use stored procedures. Stored procedures are written and stored at database server. When a client contacts the server, the stored procedure is executed, and the results are sent to the client. Let us see why stored procedures are written in Oracle, at server side.

When a client/server software is created, we observe two main parts of the software:

- The first part represents the screens which accepts the input from the user and also display results to the user. The program code that helps to create such screens is called 'Presentation logic'.
- The second part represents the logic that converts the input into the output. It contains some business procedures and calculations related to the activities of an organization. This is called 'Business logic'.

It is advisable to create presentation logic and business logic separately without mixing them together. When they are developed separately, their maintenance becomes easy. For example, to modify presentation logic, the programmer need not think about business logic. He can modify any one without affecting the other. Similarly, it is possible to develop presentation logic and business

logic using separate technologies. For example, the screens representing the presentation logic can be created using Java or VisualBasic whereas it is possible to create the business logic in Oracle in the form of 'stored procedures'.

Important Interview Question

What are stored procedures?

A stored procedure represents a set of statements that is stored and executed at database server, sending the results to the client.

To understand a stored procedure concept, let us write a stored procedure `'myproc.sql'` to increase the salary by Rs. 500 of an employee whose eno is 1003 in emptab. Here we assuming that emptab already exists in the database.

```
-- myproc.sql
-- to increase the salary of employee whose eno=1003
create or replace procedure myproc(no in int, isal out float) as
salary float;
begin
 select sal into salary from emptab where eno=no;
 isal := salary+500;
end;
```

In the above code, we use created or replace key word to create the procedure. The procedure name is 'myproc'. It has two parameters: 'no' which is called in parameter and 'isal' which is called out parameter. When a client calls this procedure, it supplies the value for in parameter. The result is calculated and stored in out parameter, which is sent back to the client.

When 'no' is sent by the client, the corresponding row is retrieved by the following statement:

```
select sal into salary from emptab where eno=no;
```

This will store the salary of the employee into 'salary' variable. This salary is then incremented by Rs.500 and then stored into the out parameter 'isal'. The value in isal is only available to the client.

Type this stored procedure at SQL> prompt in oracle. Or, you can type the stored procedure in a Notepad file then copy and paste it at SQL> prompt. Then type '/' and press <Enter> to execute it. It displays a message: 'Procedure created'.

The next step is calling this stored procedure from our JDBC program. To do so, we need CallableStatement interface. For example, we can write:

```
CallableStatement stmt =  con.prepareCall("{ call  myproc(?,?)}");
```

In the above statement, `prepareCall()` is used to call the stored procedure: 'myproc'. After procedure's name, we got two '?' marks which represent the in parameter and out parameter. Let us now provide some value to in parameter, as:

```
stmt.setInt(1,1004);  //pass 1004 to in parameter
```

To retrieve the data from the out parameter, first we should register the type of the parameter as:

```
stmt.registerOutParameter(2,Types.FLOAT);
```

Here, the out parameter is registered as FLOAT type. All types are defined in Types class of java.sql package. Some types are:

```
Types.INTEGER
Types.FLOAT
Types.CHAR
Types.BOOLEAN
Types.DOUBLE
Types.DATE
```

Once the 2nd parameter (i.e., out parameter) is registered, the CallableStatement should be executed, as:

```
stmt.execute();
```

Now the result comes from the stored procedure into 2nd parameter, which should be collected into incsal variable, as:

```
float incsal= stmt.getFloat(2);
```

These steps are shown in Program 12.

Program 12: In this program, we create CallableStatement to call the stored procedure and retrieve the result from oracle server.

```
/* Demonstrates how to call a stored procedure. Already the procedure is stored
at server. Aim of the procedure is to increment the salary of employee whose
eno=1003.
*/

import java.sql.*;
class CallProc
{
     public static void main(String args[])throws Exception
     {

     //register the driver
     DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

     //establish a connection with database
     Connection con;
     con = DriverManager.
     getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott","tiger");

     //create CallableStatement to call myproc
     CallableStatement stmt = con.prepareCall("{ call  myproc(?,?)}");

     //set 1003 as employee number to in parameter
     stmt.setInt(1,1003);

     //register the out parameter as of float type
     stmt.registerOutParameter(2,Types.FLOAT);

     //execute CallableStatement
     stmt.execute();

     //get the result into incsal variable
     float incsal= stmt.getFloat(2);
     System.out.println("Incremented salary= "+ incsal);

     //close connection
     con.close();
     }
```

```
    }
```

Output:

```
C:\> javac CallProc.java
C:\> java CallProc
Incremented salary= 15500.00
```

When we executed the stored procedure 'myproc', notice that it will not modify the actual salary in the emptab. To modify the actual salary, we should use 'update' statement also in the stored procedure.

CallableStatement is useful to call not only stored procedures, but also functions written in PL/SQL. To understand this, let us first write a function that takes a number from us and returns its square value.

```
-- myfun.sql
-- function that returns square of a number
```

create or replace function myfun(i number) return number as:

```
BEGIN
   return i*i;
END myfun;
```

Here, we use create or replace key word to create function. Our function name is 'myfun' that accepts a number through in parameter 'i'. Then square value is calculated by the expression i*i and returned by this function.

Type this function code at SQL> prompt. At the end type '/' and press <Enter>. It displays a message: Function created.

Let us write a Java program to call this myfun function.

Program 13: This program uses CallableStatement to call a function and gets the returned value from the function.

```
/* Demonstrates how to call a function
   To increment salary of employee whose eno=1003
*/
import java.sql.*;
class CallFun
{
   public static void main(String args[])throws Exception
   {
    //register a driver
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    //establish a connection with database
    Connection con;
    con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:oracle","scott"
,"tiger");

     //create CallableStatement and call myfun. Here, first ? is out parameter
     //and second ? is in parameter.
     CallableStatement stmt =  con.prepareCall("{?= call  myfun(?)}");

    //register the out parameter as integer type
      stmt.registerOutParameter(1,Types.INTEGER);
       //set the in parameter to 50
       stmt.setInt(2,50);
```

```
            //execute the CallableStatement
            stmt.execute();

            //get the result from out parameter into value
            int value= stmt.getInt(1);
            System.out.println("Square value= "+value );

        }
    }
```

Output:

```
C:\> javac CallFun.java
C:\> java CallFun
Square value= 2500
```

Important Interview Question

What is the use of CallableStatement?

CallableStatement is useful to call stored procedures and functions which run at a database server and get the results into the client.

Types of Result Sets

There are two types of Result sets, namely, Forward ResultSet and Scrollable ResultSet. So far, in our programs, we retrieved the results into a ResultSet object rs, as shown:

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("select * from emptab");
```

Since the results are available in 'rs', we can retrieve the rows one by one in forward direction using `rs.next()` method. Suppose we want to retrieve the rows in reverse direction, is there any method like `rs.previous()`? Fortunately this method is available, but it is not supported by the `ResutlSet` object 'rs'. This means using `ResultSet`, it is possible to move in forward direction only. This is called 'forward result set'.

There is another type of result set, where we can move in both forward and backward directions. This is called 'scrollable result set'. This type of result set is created by passing two constants to `createStatement()` method, as:

```
Statement stmt = con.createStatement(CONST1, CONST2);
```

Here, `CONST1` may take any one of the following:

```
ResultSet.TYPE_SCROLL_SENSITIVE
ResultSet.TYPE_SCROLL_INSENSITIVE
```

And `CONST2` may be one of the following:

```
ResultSet.CONCUR_READ_ONLY
ResultSet.CONCUR_UPDATABLE
```

The constant `ResultSet.TYPE_SCROLL_SENSITIVE` represents that any changes done to the `ResultSet` will also affect the database. Where as, the constant `ResultSet.TYPE_SCROLL_INSENSITIVE` indicates that any changes done to the ResultSet will not reflect in the database. Most of the driver vendors provide this type of result set only

```
rs.updateRow();
```

The same way, if we want to insert a new row into the table:

- Allot memory for storing one row's data.

```
rs.moveToInsertRow();
```

- Store new data into the row.

```
rs.updateInt(1, 1000);
rs.updateString(2, "xxxxxx");
rs.updateFloat(3, 9999.90f);
```

- Now insert the new row into the table, as:

```
rs.insertRow();
```

The way to delete a row follows as:

- Go to the row to be deleted. To delete 4th row:

```
rs.absolute(4);
```

- Now, delete the row from the table.

```
rs.deleteRow();
```

Oracle 10g xe is made scroll sensitive. So we should use this database in our program. Earlier versions like Oracle 8 and Oracle 8i do not support scrollable result sets.

Program 14: This program uses scrollable result set to retrieve the data from the database and also perform operations like inserting new row, deletion and updation of rows.

```
// Demonstrates how to use scrollable result sets
import java.sql.*;
class RS
{
      public static void main(String args[])throws Exception
      {
      //register the driver
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      //connet to Oracle 10g xe database
      Connection con =
      DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
      "tiger");

      //create scroll sensitive, scrollable result set
      Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
      ResultSet.CONCUR_UPDATABLE);

      //execute the query
      ResultSet rs = stmt.executeQuery("select * from emptab");

      //display all the rows from result set
      while(rs.next()){
            System.out.println(rs.getString(1));
            System.out.println(rs.getString(2));
            System.out.println(rs.getString(3));
            System.out.println("========================");
```

```
        }

        //display only first row
        rs.first();
        System.out.println("=======first()==========");
        System.out.println(rs.getInt(1));
        System.out.println(rs.getString(2));
        System.out.println(rs.getFloat(3));
        System.out.println("========================");

        //display 3rd row
        rs.absolute(3);
        System.out.println("=======absolute(3)======");
        System.out.println(rs.getInt(1));
        System.out.println(rs.getString(2));
        System.out.println(rs.getFloat(3));
        System.out.println("========================");

        //wait till Enter pressed
        System.in.read();

        //find how many rows are there in this resultset
        rs.last();
        System.out.println("No of Rows= "+rs.getRow());

        //wait till Enter pressed
        System.in.read();

        //make the query again
        rs = stmt.executeQuery("select eno,ename,sal from emptab");

        //update 3rd row in result set and store it into the database
        rs.absolute(3);
        rs.updateInt(1,1006);
        rs.updateString(2,"Laxman Kumar");
        rs.updateFloat(3,4500f);
        rs.updateRow();

        //wait till Enter pressed
        System.in.read();

        //insert a new row
        rs.moveToInsertRow();
        rs.updateInt(1,1000);
        rs.updateString(2,"Mahesh");
        rs.updateFloat(3,3990f);
        rs.insertRow();

        //wait till Enter pressed
        System.in.read();

        //delete 4th row
        rs.absolute(4);
        rs.deleteRow();

        //close the result set
        rs.close();

        //close connection with the database
        con.close();

        }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac RS.java
C:\> java RS
777
Vijaya
5600.75
====================
779
Satyaraj
5000
====================
1001
Nagesh
12000.5
====================
1002
Venkat
2000
====================
======first()==========
777
Vijaya
5600.75
====================
======absolute(3)======
1001
Nagesh
12000.5
====================
<Enter>
No of Rows= 4
<Enter>
<Enter>

C:\>
```

Now check the database to find out if the above changes are also refl

Storing Images Into Database

SQL offers BLOB (Binary Large Object) data type to store image fi database table. At the time of creating table, the data type of the col image should be declared as 'blob' type. Suppose we want to stor 'photo'. The number is the next column 'no' . The table can be create

```
create table bigtab(photo blob, no int);
```

To store an image into photo column of the 'bigtab' table, we should

- Load the image into a File object.

```
File f = new File("x.gif");
```

- Attach the File object to FileInputStream for reading the image.

```
FileInputStream fis = new FileInputStream(f);
```

- Read the image using FileInputStream and store it into the table using `setBinaryStream()` method of `Statement` interface.

```
stmt.setBinaryStream(1, fis, (int)f.length());
```

Here, 1 represents the first column of the table where the image should be stored. fis represents the `FileInputStream` object that reads the image and `f.length()` gives the image size in bytes.

- Execute the statement using `executeUpdate()`. Then the image is stored into the table.

```
stmt.executeUpdate();
```

Program 15: In this program, we store an image `x.gif` into the first column of the bigtab table. In this program, we use `PreparedStatement`.

```
/* This program demonstrates how to store binary data(image) */
import java.io.*;
import java.sql.*;
public class StoreImage
{
    public static void main(String args[])throws Exception
    {

    //register the driver
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    //establish connection
    Connection con;
    con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
"tiger");

    //use PreparedStatement to update the table: bigtab
    PreparedStatement stmt = con.prepareStatement("update bigtab set photo
=? where no = 10");

    //Load the photo or image into a File object.
    File f = new File("plane.GIF");

    //Attach the file to FileInputStream for reading the image
    FileInputStream fis = new FileInputStream(f);

    //Write the file contents into the table
    stmt.setBinaryStream(1,fis,(int)f.length());
    System.out.println("Image length = " + f.length());

    //Execute the statement
    System.out.println("No of rows affected = "+stmt.executeUpdate());

    }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac StoreImage.java
C:\> java StoreImage

Image length = 34040
No of rows affected = 1
```

Retrieving Images From Database

- Get the image from the table into Blob class object.

```
Blob b = rs.getBlob(1);
```

Here, 1 represents the 1st column from where the image is retrieved and stored into b.

- Create a byte array and store the image from b into the byte array.

```
byte arr[] = new byte[size of image];
arr = b.getBytes(1, (int)b.length());
```

Here, `getBytes()` method is retrieving the image from b, starting from 1st byte and the entire length of the image is retrieved(all bytes).

- Write this byte array arr into a file.

```
FileOutputStream fos = new FileOutputStream("x1.gif");
fos.write(arr);
```

It means that the image is stored into the file as `x1.gif` file.

Program 16: This program retrieves the image from 1st column of bigtab and stores it in the current directory with the name `x1.gif`.

```
/* Demonstrates how to retrieve binary data(image) */
import java.sql.*;
import java.io.*;

public class GetImage
{
     public static void main(String args[])throws Exception
     {
     //register the driver
     DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

     //establish connection
     Connection con =
     DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
     "tiger");

     //create a statement
     Statement stmt = con.createStatement();

     //execute the statement
     ResultSet rs = stmt.executeQuery("select * from bigtab");

     //go to first row
     rs.next();

     //Get the image from the table into Blob object.
     Blob b = rs.getBlob(1);

     //Create a byte array having the size of image
     byte b1[] = new byte[(int)b.length()];

     //store the Blob object into the byte array.
     b1 = b.getBytes(1,(int)b.length());

     System.out.println("Image length = "+ b.length());

     //Write this byte array into a file: x1.gif
     FileOutputStream fos = new FileOutputStream("x1.gif");
```

```
            fos.write(b1);

            //close the file
            fos.close();

            //close connection
            con.close();
            }
      }
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac GetImage.java
C:\> java GetImage
Image length = 34040
```

Open `x1.gif` file to see the image that is stored into the bigtab of the database.

Storing a file into database

To store a large volume of data as well as text files into a table, we can take the help of CLOB (Character Large Object) datatype of SQL. Using CLOB, it is possible to store an entire file, a text book or a resume etc., into a column of the table.

First, create a table with the name myclob to store a text file.

```
create table myclob(col1 clob, col2 int);
```

Here the first column is declared as 'clob' type where we can store a text file. Now insert a row into myclob as:

```
insert into myclob(col2) values(100);
```

Here, we are not inserting any thing into col1, but inserting 100 into col2.

Now, following the steps to insert a text file into col1 of myclob table.

- Load myfile.txt into File object as:

```
File f = new File("myfile.txt");
```

- To read data from File object f, connect it to a FileReader.

```
FileReader fr = new FileReader(f);
```

- Now, read data from the FileReader and send it to 1st column of the table using `setCharacterStream()` method.

```
Stmt.setCharacterStream(1, fr, (int)f.length());
```

Here, 1 represents that the data should be set to 1st column. The data is read from 'fr' and the length of the file is given by `f.length()`.

Program 17: In this program, we store `myfile.txt` file into myclob table. See that the file `myfile.txt` already exists in the present directory.

```
//To store a file content into a table
import java.io.*;
import java.sql.*;
```

```
class ClobDemo
{
      public static void main(String args[]) throws Exception
      {
      //register the driver
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      //establish connection
      Connection con =
      DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
      "tiger");

      //create SQL statement
      PreparedStatement stmt = con.prepareStatement("update myclob set col1= ?
      where col2 = 100");

      //load the file into File object
      File f = new File("myfile.txt");

      //connect the File to FileReader for reading
      FileReader fr = new FileReader(f);

      //store the file into col1 as character stream
      stmt.setCharacterStream(1, fr, (int)f.length());

      //disply file size
      System.out.println("File size= "+ f.length());

      //execute the statement
      System.out.println("No. of rows affected= "+ stmt.executeUpdate());

      //close connection
      con.close();
      }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac ClobDemo.java
C:\> java ClobDemo
File size= 1881
No. of rows affected= 1
```

Retrieving a File from the Database

To retrieve the file from the column of a table, we can follow the steps:

- First, we should retrieve the file from the result set using `getClob()` method.

```
Clob c = rs.getClob(1);
```

Here, `getClob()` method is retrieving data from 1st column and storing it into Clob class object c.

- Use `getCharacterStream()` method to get the file data from the Clob object into a reader object.

```
Reader r = c.getCharacterStream();
```

- Store the data from Reader into a new file with the name `newfile.txt`.

```
FileWriter fw = new FileWriter("newfile.txt");
while((ch = r.read()) != -1)
  fw.write((char)ch);
```

Here, we are reading data from Reader object r and writing it into `FileWriter` which stores it into `newfile.txt`.

Program 18: In this program, we read the contents of myfile.txt file from mydb table and store the same into newfile.txt.

```
//To retrieve text file content from a table
import java.io.*;
import java.sql.*;
class ClobDemo1
{
      public static void main(String args[]) throws Exception
      {
      //register the driver
      DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

      //establish connection
      Connection con =
      DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
      "tiger");

      //create SQL statement
      Statement stmt = con.createStatement();

      //read table rows into ResultSet
      ResultSet rs = stmt.executeQuery("select * from myclob");

      //go to first row
      rs.next();

      //read data from col1
      Clob c = rs.getClob(1);

      //display file length
      System.out.println("File size= "+ c.length());

      //read file data from c and store into Reader object
      Reader r = c.getCharacterStream();

      //read data from Reader and write into newfile.txt
      int ch;
      FileWriter fw = new FileWriter("newfile.txt");
      while((ch = r.read()) != -1)
      fw.write((char)ch);

      //close the file
      fw.close();

      //close the connection
      con.close();
      }
}
```

Output:

```
C:\> set classpath=C:\jars\ojdbc14.jar;.;
C:\> javac ClobDemo1.java
C:\> java ClobDemo1
File size= 1881
```

Important Interview Question

What is BLOB?

Binary Large Object (BLOB) is a SQL datatype that represents binary data to be stored into a database. BLOB helps us to store images into a database.

What is CLOB?

Character Large Object (CLOB) is a SQL datatype that represents larger volumes of text data. CLOB helps to store text files into a database.

ResultSetMetaData

`ResultSetMetaData` is an interface which contains methods to get information about the types and properties of the columns in a `ResultSet` object. The following program demonstrates how to use retrieve the information about the `ResultSet`.

ResultSet has a method `getMetaData()` which returns the information about the result set into `ResultSetMetaData` object. It can be called as:

```
ResultSetMetaData rsmd =rs.getMetaData();
```

Program 19: Let us find out the information of the columns of the table: emptab This information is available in ResultSet object which can be again retrieved into ResultSetMetaData.

```
/* Demonstrates how to find out ResultSet information
   using ResultSetMetaData
*/

import java.sql.*;
class RSInfo
{
     public static void main(String args[])throws Exception
     {

     //register the driver
     Class.forName("oracle.jdbc.driver.OracleDriver");

     //establish a connection
     Connection con =
     DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
     "tiger");

     //create the statement
     Statement stmt = con.createStatement();

     //get rows into ResultSet object
     ResultSet rs = stmt.executeQuery("select * from emptab");

     //get information about result into ResultSetMetaData
     ResultSetMetaData rsmd =rs.getMetaData();

     //count no. of columns in resultset
     int n= rsmd.getColumnCount();
     System.out.println("No. of columns= "+n);

     //display information about each column
     for(int i=1; i<=n; i++)
     {
     System.out.println("Column Number: "+i);
     System.out.println("=====================");
```

```
            System.out.println("Column name= "+rsmd.getColumnName(i));
            System.out.println("Column type= "+rsmd.getColumnTypeName(i));
            System.out.println("Column width= "+rsmd.getColumnDisplaySize(i));
            System.out.println("Column Precision= "+rsmd.getPrecision(i));
            System.out.println("Is currency= "+rsmd.isCurrency(i));
            System.out.println("Is Read Only= "+rsmd.isReadOnly(i));
            System.out.println("Is Writable= "+rsmd.isWritable(i));
            System.out.println("Is Searchable= "+rsmd.isSearchable(i));
            System.out.println("Is Signed= "+rsmd.isSigned(i));
            }

            //close connection
            con.close();
            }
      }
```

Output:

```
C:\> set classpath=c:\jars\ojdbc14.jar;.;
C:\> javac RSInfo.java
C:\> java RSInfo

No. of columns= 3
Column Number: 1
=====================
Column name= ENO
Column type= NUMBER
Column width= 22
Column Precision= 38
Is currency= true
Is Read Only= false
Is Writable= true
Is Searchable= true
Is Signed= true
Column Number: 2
====================
Column name= ENAME
Column type= VARCHAR2
Column width= 20
Column Precision= 20
Is currency= false
Is Read Only= false
Is Writable= true
Is Searchable= true
Is Signed= true
Column Number: 3
=====================
Column name= SAL
Column type= NUMBER
Column width= 22
Column Precision= 126
Is currency= true
Is Read Only= false
Is Writable= true
Is Searchable= true
Is Signed= true
```

DatabaseMetaData

DatabaseMetaData is an interface to get comprehensive information about the database as a whole. This interface is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS) in combination with the JDBC driver that is used with it.

To know which tables and views are available in a database, we can use `getTables()` method, which is in the form shown below:

```
ResultSet  rs = dbmd.getTables(String catalog, String schemaPattern, String
tableNamePattern, String[] types);
```

Here, we can pass null objects to catalog, schemaPattern and tableNamePattern. For the 4th parameter, we can pass a String type array that contains the type: table or view, as:

```
String x[] = {"TABLE"};  //if this is passed, information of all tables is
retrieved
String x[] = {"VIEW"};  //if this is passed, information of all views is
retrieved
```

Program 20: In this program, let us connect to oracle database with thin driver and let us know which features are offered by the database and driver vendors.

```
//Demonstrates how to find out Database Capabilities  using DatabaseMetaData
import java.sql.*;
class DBCap
{
      public static void main(String args[])throws Exception
      {

      //register driver
      Class.forName("oracle.jdbc.driver.OracleDriver");

      //establish connection with database
      Connection con =
      DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","scott",
      "tiger");

      //get the information about the database into DatabaseMetaData
      DatabaseMetaData dbmd =con.getMetaData();

      //display information about the database
      System.out.println("DB Name= "+dbmd.getDatabaseProductName());
      System.out.println("DB Version= "+dbmd.getDatabaseProductVersion());
      System.out.println("DB Driver Name= "+dbmd.getDriverName());
      System.out.println("Driver Major Version= "+
      dbmd.getDriverMajorVersion());
      System.out.println("Driver Minor Version= "+
      dbmd.getDriverMinorVersion());
      System.out.println("URL of DB= "+dbmd.getURL());
      System.out.println("Current UserName= "+dbmd.getUserName());

      System.out.println("==========TABLES==========");
      String t[] = {"TABLE"};
      ResultSet rs= dbmd.getTables(null,null,null,t);
      while(rs.next())
      {
            System.out.println(rs.getString("TABLE_NAME"));
      }

      //wait till Enter pressed
      System.in.read();

      System.out.println("==========VIEWS==========");
      String v[] = {"VIEW"};
      rs= dbmd.getTables(null,null,null,v);
      while(rs.next())
      {
            System.out.println(rs.getString("TABLE_NAME"));
      }
```

```
            //close connection
            con.close();
            }
    }
```

Output:

```
C:\> set classpath=c:\jars\ojdbc14.jar;.;
C:\> javac DBCap.java
C:\> java DBCap

DB Name= Oracle
DB Version= Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
DB Driver Name= Oracle JDBC driver
Driver Major Version= 10
Driver Minor Version= 2
URL of DB= jdbc:oracle:thin:@localhost:1521:xe
Current UserName= SCOTT
==========TABLES==============
DR$CLASS
DR$DBO
DR$DELETE
DR$INDEX
:
:
COUNTRIES
DEPARTMENTS
EMPLOYEES
JOBS
JOB_HISTORY
LOCATIONS
REGIONS
:
:
<Enter>
==========VIEWS==============
CTX_CLASSES
CTX_INDEXES
CTX_INDEX_ERRORS
CTX_INDEX_OBJECTS
CTX_INDEX_PARTITIONS
CTX_INDEX_SETS
CTX_INDEX_SET_INDEXES
CTX_INDEX_SUB_LEXERS
CTX_INDEX_SUB_LEXER_VALUES
CTX_INDEX_VALUES
CTX_OBJECTS
CTX_OBJECT_ATTRIBUTES
CTX_OBJECT_ATTRIBUTE_LOV
CTX_PARAMETERS
CTX_PENDING
CTX_PREFERENCES
:
:
```

Types of JDBC Drivers

There are 4 types of JDBC drivers. Each driver will have its own advantages and disadvantages, as well. Let us have a look at these drivers one by one.

Type 1

- **JDBC-ODBC Bridge driver:** This driver receives any JDBC calls and sends them to ODBC (Open DataBase Connectivity) driver. ODBC driver understands these calls and communicates with the database library provided by the vendor. So, the ODBC driver, and the vendor database library must be present on the client machine. See Figure 32.14.
- **Advantages:** The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available on the client machine.
- **Disadvantages:** The performance of this driver is less, since the JDBC call goes through the bridge to the ODBC driver, then to the native database connectivity library. The result comes back through the reverse process.

  Another requirement is that the ODBC driver and the native database connectivity library must already be installed on the client machine.

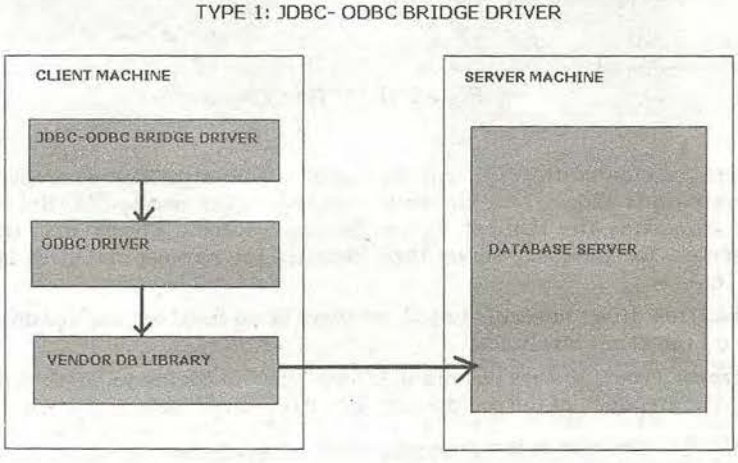

Figure 32.14 Type I driver

Type 2

- **Native API-partly Java driver:** It converts JDBC calls into database-specific calls with the help of vendor database library. The type 2 driver communicates directly with the database server; therefore it requires that some binary code be present on the client machine. See Figure 32.15.
- **Advantages:** Type 2 drivers typically offer better performance than the JDBC-ODBC Bridge.
- **Disadvantages:** The vendor database library needs to be loaded on each client machine. This is the reason; type 2 drivers cannot be used for the Internet. Type 2 drivers show lower performance than type 3 and type 4 drivers.

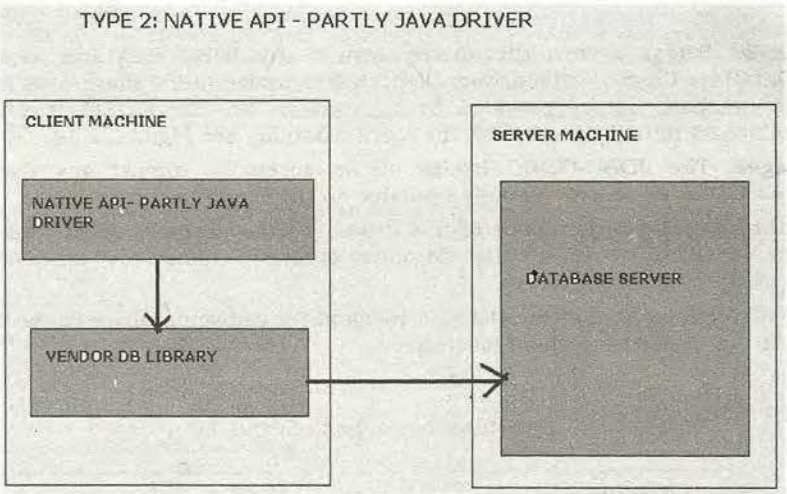

Figure 32.15 Type 2 driver

Type 3

- **Net protocol-pure Java driver:** It follows a three-tiered approach whereby the JDBC database requests are passed through the network to a middle-tier server (Ex: Net Server). The middle-tier server translates the request to the database-specific library and then sends it to the database server. The database server then executes the request and gives back the results. See Figure 32.16.
- **Advantages:** This driver is server-based, so there is no need for any vendor database library to be present on the client machines.
- **Disadvantages:** Type 3 drivers require database-specific coding to be done in the middle tier (in NetServer). Maintenance of the middle-tier server becomes costly.

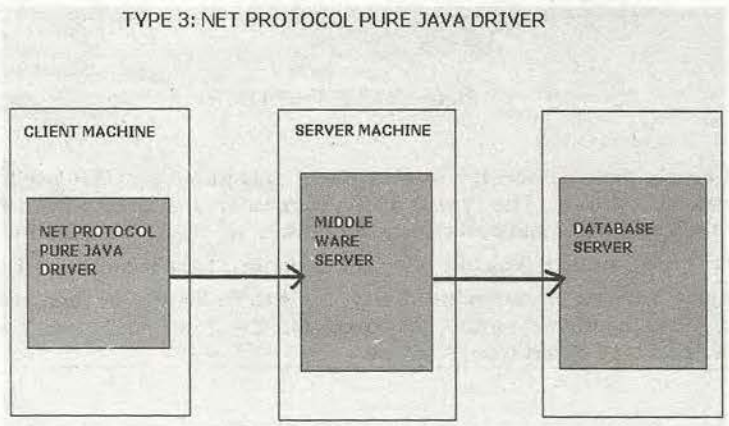

Figure 32.16 Type 3 driver

Type 4

- **Native protocol-pure Java driver:** This driver converts JDBC calls into the vendor-specific database management system (DBMS) protocol so that client applications can communicate directly with the database server. Level 4 drivers are completely implemented in Java to achieve

platform independence and eliminate deployment administration issues. See Figure 32.17. Generally type 4 drivers are used on Internet.

- **Advantages:** This driver has better performance than types 1 and 2. Also, there's no need to install any special software on the client or server.
- **Disadvantages:** With type 4 drivers, the user needs a different driver for each database. For example, to communicate with Oracle server, we need Oracle driver and to communicate with Sybase server, we need Sybase driver.

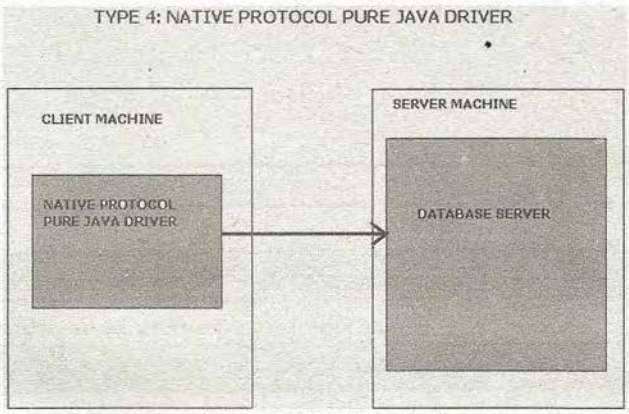

Figure 32.17 Type 4 driver

Conclusion

JDBC helps to communicate with any database through a Java program which is an essential feature for any project development environment, since each software requires data representing different transactions to be stored and retrieved later on demand. JDBC eliminates the need of mastering the commands to communicate with different databases by providing a common mechanism to communicate with any database in the world. Each time the database is changed, the Java program need not be changed and hence this provides better maintenance of code across different platforms.