servletCode:-

program1:

import java.sql.\*;  
public class CreateTable  
{  
public static void main(String rags[])  
{  
try  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
System.out.println(“driver loaded”);

Connection con =DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
System.out.println(“connection established”);  
Statement stmt=con.createStatement();  
System.out.println(“statement object created”);

stmt.executeUpdate(“create table employee(eno number,ename varchar2(15),esal number(10,2),eaddr varchar2(20))”);

System.out.println(“table created”);  
stmt.close();  
con.close();  
}//try  
catch(SQLException se)  
{  
System.out.println(“database problems”+se);  
}  
catch(ClassNotFoundException cnf)  
{  
System.out.println(cnf);  
}  
catch(Exception e)  
{  
System.out.println(e);  
}

}//main  
}//class

program2:

import java.sql.\*;  
public class InsertingValues  
{  
public static void main(String args[])  
{  
try  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con = DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement stmt = con.createStatement();  
stmt.executeUpdate(“insert into employee values(101,’ram’,9899.56,’dsnr’)”);  
stmt.executeUpdate(“insert into Employee values(102,’raj’,9999.99,’hyderabad’)”);  
stmt.executeUpdate(“insert into Employee values(103,’sam’,7777.77,’chennai’)”);  
System.out.println(“Records inserted”);  
stmt.close();  
con.close();  
}  
catch(SQLException se)  
{  
System.out.println(se);  
}  
catch(ClassNotFoundException e)  
{  
System.out.println(e);  
}  
catch(Exception e)  
{  
System.out.println(e);  
}

}//main  
}//class

program3:  
import java.util.\*;  
import java.sql.\*;  
public class UpdateRecord  
{  
public static void main(String[] args)throws Exception  
{  
DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());  
Connection con =DriverManager.getConnection(“jdbc:odbc:ksreddyDSN”,”scott”,”tiger”);  
Statement st = con.createStatement();  
Scanner s= new Scanner(System.in);  
System.out.println(“enter employee number”);  
int empno=s.nextInt();  
int updateCount = st.executeUpdate(“update employee set esal =esal+5000 where eno=”+empno);  
System.out.println(“records updated”+updateCount);  
st.close();  
con.close();  
}  
}

program4:

import java.sql.\*;  
public class DeleteRecord  
{  
public static void main(String[] args)throws Exception  
{  
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

Connection con = DriverManager.getConnection(“jdbc:odbc:ksreddyDSN”,”scott”,”tiger”);

Statement st = con.createStatement();

int updateCount = st.executeUpdate(“delete from employee where esal<=1000”);  
System.out.println(“records deleted—–“+updateCount);  
st.close();  
con.close();  
}  
}  
/\*  
for deleting all the records  
st.executeUpdate(“delete from employee”);  
\*/

program5:  
import java.sql.\*;  
import java.util.\*;  
public class DropTable  
{  
public static void main(String[] args)  
{  
Connection con=null;  
Statement st=null;  
Scanner s=null;  
try  
{  
s=new Scanner(System.in);  
System.out.println(“Enter table name to delete “);  
String tab=s.next();  
//Db code  
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);  
con=DriverManager.getConnection(“jdbc:odbc:ksreddyDSN”,”system”,”system”);

st=con.createStatement();  
st.executeUpdate(“drop table “+tab);  
System.out.println(“deleted table is”+tab);  
}//try  
catch(ClassNotFoundException cnf)  
{  
//cnf.printStackTrace();  
System.out.println(cnf);  
}  
catch(SQLException se)  
{  
se.printStackTrace();  
}  
catch(Exception e)  
{  
e.printStackTrace();  
System.out.println(e);  
}  
}//main  
}//class

program6:

/\*  
ResultSet is an interface of java.sql package.  
It is a table of data representing a database query result.  
ResultSet cursor points before the fist row of the data.  
To move the cursor to the next record we call next()  
method.  
The ResultSet provides getter methods(getBoolean((), getInt(),getString()) for retrieving column values from the current row.  
Values can be retrieved using either the index number of the column or the name of the column.  
There is a get method for each data type,and each get method has two versions which takes column name and column index.  
XXX getXXX(String columnName)  
XXX getXXX(int columnIndex)  
methods for Navigation  
1]beforeFirst()  
Moves the cursor to just before the first row  
2]afterLast()  
Moves the cursor to just after the last row  
3]first()  
Moves the cursor to the first row  
4]void last()  
Moves the cursor to the last row.  
5]absolute(int row)  
Moves the cursor to the specified row  
6]relative(int row)  
Moves the cursor the given number of rows forward or backwards from where it currently is pointing.  
7]boolean previous()  
Moves the cursor to the previous row.  
8]boolean next()  
Moves the cursor to the next row. This method returns false if there are no more rows in the result set  
9]public int getRow()  
Returns the row number that the cursor is pointing to.  
10]moveToInsertRow()  
Moves the cursor to a special row in the result set that can be used to insert a new row into the database.  
\*/  
import java.sql.\*;  
public class RetrieveRecords  
{  
public static void main(String args[]) throws Exception  
{  
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);  
Connection con = DriverManager.getConnection(“jdbc:odbc:ksreddyDSN”,”scott”,”tiger”);  
Statement stmt = con.createStatement();

ResultSet res = stmt.executeQuery(“select \* from employee”);  
while(res.next())  
{  
System.out.println(res.getInt(1)+”\t” + res.getString(2) + “\t” + res.getDouble(3)+”\t”+res.getString(4));  
}  
res.close();  
stmt.close();  
con.close();  
}  
}  
/\* different ways  
System.out.println(res.getInt(“eno”)+”\t” + res.getString(“ename”) + “\t” + res.getDouble(“esal”)+”\t”+res.getString(“eaddr”));  
System.out.println(res.getString(“eno”)+”\t” + res.getString(“ename”) + “\t” + res.getString(“esal”)+”\t”+res.getString(“eaddr”));  
System.out.println(res.getString(1)+”\t” + res.getString(2) + “\t” + res.getString(3)+”\t”+res.getString(4));  
\*/

program7:

import java.util.\*;  
import java.sql.\*;  
import java.io.\*;  
public class UpdatableResultDemo  
{  
p ublic static void main(String[] args)throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement st = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  
ResultSet rs = st.executeQuery(“select \* from employee”);  
Scanner sn= new Scanner(System.in);  
rs.moveToInsertRow();  
while(true)  
{  
System.out.println(“enter employee number”);  
int eno = sn.nextInt();  
System.out.println(“enter employee name”);  
sn.nextLine();  
String ename = sn.nextLine();  
System.out.println(“enter employee salary”);  
float esal = sn.nextFloat();  
System.out.println(“enter employee address”);  
sn.nextLine();  
String eaddr = sn.nextLine();  
rs.updateInt(1,eno);  
rs.updateString(2,ename);  
rs.updateFloat(3,esal);  
rs.updateString(4,eaddr);  
rs.insertRow();  
System.out.println(“record successfully inserted”);  
System.out.println(“one more record[y/n]”);  
String option = sn.next();  
if(option.equals(“n”))  
break;  
}}}

program8:

import java.sql.\*;  
public class UpdatableResultDemo1  
{  
public static void main(String[] args)throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con= DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement st = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_UPDATABLE);  
ResultSet rs = st.executeQuery(“select \* from employee”);  
rs.absolute(1);// move to first record  
double sal = rs.getDouble(3);  
System.out.println(sal);  
sal = sal +500;  
rs.updateDouble(3,sal);  
rs.updateRow();  
System.out.println(rs.getDouble(3));  
} }

program9&10(sensitive and Insensitive)

import java.sql.\*;  
public class Sensitive  
{  
public static void main(String[] args)throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement st = con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,ResultSet.CONCUR\_READ\_ONLY);  
ResultSet rs = st.executeQuery(“select \* from employee”);  
rs.next();  
System.out.println(“old salary o f the employee”+rs.getFloat(3));  
System.in.read();  
//application is in pause, perform database updations  
rs.refreshRow();  
System.out.println(“new salary of the employee”+rs.getFloat(3));  
}  
}  
/\* JDBC VERSIONS

java 1.1      JDBC 1.0  
java 1.2      JDBC 2.0  
java 1.3      JDBC 2.0  
java 1.4      JDBC 3.0  
java 1.5      JDBC 3.0  
java 1.6      JDBC 4.0  
java 1.7       JDBC 4.1  
java 1.8       JDBC 4.2

• Scroll forward and backward in a result set or move to a specific row.  
• Make updates to a database tables, using methods in the Java programming language instead of using SQL commands.  
• Send multiple SQL statements to the database as a unit, or batch.  
• Use the new SQL3 data types as column values

SQL 99 Datatypes(SQL 3 types)  
Blob  
Clob  
Array  
Struct  
ref  
———————————————–

Batch updates or batch processing:

A batch update is a set of multiple update statements that can be executed at a time as a batch on the database.

batch processing reduces network round trips,network trffic b/w client and db s/w.in the batch of sql queries we should not place select query.

In the batch of sql queries we should not place select query.  
Batch updates can be made with Statement, PreparedStatement and CallableStatement objects.  
Generally, for all SQL statements, by default, autocommit is called.  
For executing batch statements, first we must set autocommit to false so that a transaction is not implicitly committed or rolled back.

The following code illustrates:

con.setAutoCommit(fasle);  
Statement stmt = con.createStatement( );

stmt.addBatch(“insert into Employee values(100, ’ksreddy’, 2000.50)”);  
stmt.addBatch(“insert into Employee values(101, ’3sha’, 4500.50)”);  
stmt.addBatch(“insert into Employee values(102, ’9tara’, 3450.75)”);  
stmt.executeBatch( );  
con.setAutoCommit(true);

With addBatch( ) method, we can group all the SQL statements as a single lot. We can empty all the above insert statements from the Statement object stmt by calling clearBatch( ) method.

\*/

program 11

import java.sql.\*;  
public class BatchUpdatesDemo  
{  
public static void main(String args[]) throws Exception  
{  
Connection con=null;  
Statement st=null;  
ResultSet rs=null;  
try  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
con = DriverManager.getConnection( “jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
con.setAutoCommit(false);  
st=con.createStatement();  
st.addBatch(“insert into employee values(111,’ram’,5550.89,’hyderabad’)”);  
st.addBatch(“insert into employee values(112,’sam’,666.89,’secbad’)”);  
st.addBatch(“delete from employee where eno=102”);  
st.executeBatch();  
con.commit();  
}  
catch(Exception e)  
{  
con.rollback();  
}}}

program 12

/\*  
PreparedStatement  
it is an interface available in java.sql package and extends Statement interface  
PreparedStatement object represents the precompiled SQL statement. Whenever, the SQL statement is precompiled then it stored in the PreparedStatement object which executes the statement many times and it reduces the time duration of execution.

when we submit the sql statement using PreparedStatement object then the SQL statement will be compiled only once first time and precompiled sql statement will be executed every time.

totaltime=req time+compile time+execution time +response time  
5ms+5ms+5ms+5ms=20ms  
first time =1sql statement= 20 ms  
next onwards=5ms+0ms+sms+5ms=15ms

PreparedStatement gives you the plcace holder mechanism for providing the data dynamic to the query. we use ? as place holder

public void setXXX(int paramindex, XXXvalue)  
xxx can be int,String,long,float,Date

we can create the PreparedStatemetnt using the following methods of Connection inteface.  
1]public PreparedStatement prepareStatement(String sql)  
2]public PreparedStatement prepareStatement(String sql,int,int)  
3]public PreparedStatement prepareStatement(String sql,int,int,int)

After creating the PreparedStatement we can use any of the following method to submit sql sttement

1]public int executeUpdate()  
2]public boolean execute()  
3]public ResultSet executeQuery()  
\*/  
import java.sql.\*;  
import java.io.\*;  
import java.util.\*;  
public class PSDemo  
{  
public static void main(String args[])  
{  
try  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con = DriverManager.getConnection( “jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);

PreparedStatement pst=con.prepareStatement(“insert into Employee(eno,ename,esal,eaddr)values(?,?,?,?)”);

//? is the place holder for the data to be inserted dynamically later.  
Scanner sn= new Scanner(System.in);

System.out.print(“Employee ID No:”);  
int no = sn.nextInt();

System.out.print(“Employee name:”);  
sn.nextLine();  
String name = sn.nextLine();

System.out.print(“Employee Salary:”);  
double sal = sn.nextDouble();  
System.out.print(“Employee addr:”);  
sn.nextLine();  
String address =sn.nextLine();  
pst.setInt(1,no);  
pst.setString(2,name);  
pst.setDouble(3,sal) ;  
pst.setString(4,address) ;  
pst.executeUpdate();  
pst.close();  
con.close();

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

program13(CallableStatement)

it is used to call procedures and function.

/\*  
create or replace function myfunction(i in number) return number as  
begin  
return i \*100;  
end;  
/  
sql>select myfunction(5) from dual;

500  
\*/  
import java.sql.\*;  
public class CallableDemo1  
{  
public static void main(String args[])throws Exception  
{  
oracle.jdbc.driver.OracleDriver od=new oracle.jdbc.driver.OracleDriver();  
Connection con = DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
CallableStatement cst=con.prepareCall(“{?=call myfunction(?)}”);  
// set input parameters  
cst.setInt(2,5);  
//2 represents 2nd question mark  
//register output types  
cst.registerOutParameter(1,Types.INTEGER);  
cst.execute();  
System.out.println(“Value returned by function =” + cst.getInt(1));  
}}

program 14:

/\*  
Creating a procedure at the database:  
create or replace procedure first\_pro(x in number,y out number) as  
begin  
y := x\*x;  
end ;  
/

SQL> var result number;  
SQL> exec first\_pro(10,:result);  
SQL> print result;  
100  
\*/  
import java.sql.\* ;  
public class CallableDemo2  
{  
public static void main( String args[])throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
CallableStatement cst=con.prepareCall(“{CALL first\_pro(?,?)}”);  
cst.registerOutParameter(2,Types.INTEGER) ;  
cst.setInt(1,10);  
cst.execute();  
//OUT parameter, the second place holder ? is be registered  
int i=cst.getInt(2);  
System.out.println(“Result is: ” +i);  
}  
}

program15: inserting a image into the database

/\*  
BLOB (binary large object)  
it is datatype to store images like .gif or jpg into the database table  
when we create a table to insert a image we need to declare the type as blob  
create table emp(empno int,photo blob);

\*/  
import java.sql.\*;  
import java.io.\*;  
class BlobInsert  
{  
public static void main(String args[]) throws Exception  
{  
File f = new File(“t9.gif”);  
FileInputStream fis =new FileInputStream(f);  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con  
=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
PreparedStatement stmt = con.prepareStatement(“insert into emp values(?,?)”);  
stmt.setInt(1,101);  
stmt.setBinaryStream(2,fis,(int)f.length());  
stmt.executeUpdate();  
System.out.println(“image inserted”);  
con.close();  
}  
}

sql>select \* from emp;

empno              photo

101                     0010010132100100

we canot view images on sql command line

program 16: retrieving the image from database

/\*  
Blob:  
it is a interface from java.sql which represents sql blob datatype value;  
it contains the binary data.  
we can get Blob object by using the method getBlob() of ResultSet.  
Blob b=rs.getBlob();  
\*/  
import java.sql.\*;  
import java.io.\*;  
class BlobGet  
{  
public static void main(String args[]) throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con  
=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement stmt =con.createStatement();  
ResultSet rs = stmt.executeQuery(“select \* from emp”);  
rs.next();  
Blob b =rs.getBlob(2);  
System.out.println(“picture is Retrived”);  
FileOutputStream fos = new FileOutputStream(“h.gif”);  
byte ab[]=b.getBytes(1,(int)b.length());  
fos.write(ab);  
con.close();  
}}

program 17:  inserting  text file data into the database.

/\*  
clob:  
it is a datatype in oracle to store text(a group of characters)  
to insert a text file into the data base  
create table text(col1 clob,col2 int);  
insert into text(col2) values(100);  
\*/  
import java.io.\*;  
import java.sql.\*;  
class ClobDemo  
{  
public static void main(String[] args) throws IOException,SQLException  
{  
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver( ));  
Connection con = DriverManager.getConnection( “jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
PreparedStatement stmt = con.prepareStatement(“update text set col1 = ? where col2=100”);  
File f= new File(“abc.txt”);  
FileReader fr = new FileReader(f);  
//store the file into col1 as character stream  
stmt.setCharacterStream(1,fr,(int)f.length());  
//display the file size  
System.out.println(“FileSize”+f.length());  
System.out.println(“NO of rows effected “+stmt.executeUpdate());  
stmt.close();  
con.close();  
}}

program 18: retriveing the  text from database and create a file

/\*  
Clob  
it is a interface from java.sql which contains sql clob datatype value  
getClob() of ResultSet returns the Clob object.  
Clob c=rs.getClob();  
use getCharacterStream () method to get the file data from the Clob object into the reader object  
Reader r=c.getCharacterStream();  
store the data from Reader into a new file with the name ba.txt

blob vs Blob  
clob vs Clob

\*\*/  
import java.io.\*;  
import java.sql.\*;  
public class ClobGet  
{  
public static void main(String[] args) throws Exception  
{  
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver( ));  
Connection con = DriverManager.getConnection( “jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement stmt=con.createStatement();  
ResultSet rs =stmt.executeQuery(“select \* from text”);  
//go to first row  
rs.next();  
// read the data from the col1  
Clob c = rs.getClob(“col1”);  
//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 and write tinto file  
FileWriter fw= new FileWriter(“ba.txt”,true);  
int ch;  
while((ch=r.read()) != -1)  
{  
System.out.print((char)ch);  
fw.write((char)ch);  
}  
fw.close();  
con.close();  
}}

NetBeans  
->Type : IDE software for developing for java,j2ee &frame work software based applications  
->version 7.0  
->vendor sunmicrosystem(oracle)  
->compatible jdk 1.5 and above  
->opensource software  
->Gives glass fish as in built server  
->down load www.netbeans.org  
psvm+tabspace  
ctrl+shift+i–>package import  
————————————————–  
procedure to develop java application by using netbeans IDE to drop databse table.

step1: launch netbeans IDE  
step2:create java project from netbeans IDE

filemenu->new project->java->java application->next->projectname ->deselect create main class–>select main project finish  
step3: add java class to the project  
right click on the project –>new –>java class->class name(Test)–>finish  
step4: run the application  
right click in the source code of file ->run file–>click on it

meta-data  
———  
data about data is called as meta data that means gathering much information about given information is nothing but meta-data.

in jdbc meta data programming is usefull to know limitations,capabilities and facilities of underlying database software and its resources.

jdbc supports 3 meta data programming concepts:  
1]DatabaseMetaData  
2]ResultSetMetaData  
3]ParameterMetaData

DatabaseMetaData:  
it is a interface in java.sql packagae  
we can get this object by using getMetaData() of Connection

EX]DataBaseMetaData dbmd=con.getMetaData();

by invoking various methods on DatabaseMetaData object we can gather the details such has  
1]Database name and version  
2]support for PL/SQL procedures and functions  
3]max characters in table name and column name etc

program19: //data base meta program  
package com.configindia;  
import java.io.\*;  
import java.sql.\*;  
public class DBcap  
{  
public static void main(String args[])throws Exception  
{  
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());  
Connection con=DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
DatabaseMetaData dbmd =con.getMetaData();  
System.out.println(“database name “+dbmd.getDatabaseProductName());  
System.out.println(“database version “+dbmd.getDatabaseProductVersion());  
System.out.println(“jdbc driver version “+dbmd.getDriverVersion());  
System.out.println(“usesLocalFiles”+dbmd.usesLocalFiles());  
System.out.println(“sql key words =”+dbmd.getSQLKeywords());  
System.out.println(“numeric functions “+dbmd.getNumericFunctions());  
}}  
——————————————————————  
ResultSetMetaData:–  
it is a interface from java.sql  
it gives varoius details about the database table that is represented by jdbc Resultset object like column name,column data types,column count.

ResultSet rs=st.executeQuery(“select \* from student”);  
ResultSetMetaData rsmd=rs.getMetaData();

program20:

package com.configindia;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.Statement;  
public class RSMD  
{  
public static void main( String args[]) throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con = DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
Statement stmt=con.createStatement();  
ResultSet rs =stmt.executeQuery(“select \* from employee”);  
ResultSetMetaData rsmd=rs.getMetaData();  
// to print the column labels  
for(int i=0;i<rsmd.getColumnCount( );i++)  
System.out.print( rsmd.getColumnLabel(i+1)+”\t”);  
System.out.println();

while(rs.next())  
{  
System.out.println(rs.getInt(1)+”\t”+rs.getString(2)+”\t”+ rs.getDouble(3)+”\t”+rs.getString(4));  
}  
rs.close();  
stmt.close();  
con.close();  
}}  
——————————————————-  
ParameterMetaData:  
It is a interface from java.sql  
gives details about parameters available in sql queries of jdbc PreparedStatement and callableStatement

PreparedStatement ps=con.PrepareStatement(“insert into student values(?,?,?)”);  
(? ? ? are parametrs of query)

ParameterMetaData pmd=ps.getParameterMetaData();

using pmd object we can gather the details of parameters like parameter name,type and other details.

if any method calls of MetaData programming returns 0 or null then we need to understand the underlying jdbc driver is not capable of gathering that information.

program21:

package com.configindia;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ParameterMetaData;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.Statement;  
public class PMData  
{  
public static void main( String args[]) throws Exception  
{  
Class.forName(“oracle.jdbc.driver.OracleDriver”);  
Connection con = DriverManager.getConnection(“jdbc:oracle:thin:@localhost:1521:xe”,”system”,”system”);  
PreparedStatement ps=con.prepareStatement(“INSERT Into student VALUES(?, ?, ?)”);  
ParameterMetaData pmd = ps.getParameterMetaData();  
for (int i = 1; i < pmd.getParameterCount(); i++) {  
System.out.println(“Parameter number ” + i);  
System.out.println(“Class name is ” + pmd.getParameterClassName(i));  
// Note: Mode relates to input, output or inout  
System.out.println(” Mode is ” + pmd.getParameterMode(i));  
System.out.println(” Type is ” + pmd.getParameterType(i));  
System.out.println(” Type name is ” + pmd.getParameterTypeName(i));  
System.out.println(” Precision is ” + pmd.getPrecision(i));  
System.out.println(” Scale is ” + pmd.getScale(i));  
System.out.println(” Nullable? is ” + pmd.isNullable(i));  
System.out.println(” Signed? is ” + pmd.isSigned(i));

}}}

reading the data from properies file

db.properties

#db.properties  
#type 4  
driver=oracle.jdbc.driver.OracleDriver  
url=jdbc:oracle:thin:@localhost:1521:xe  
user=system  
password=system

program 22(crud)

import java.sql.\*;  
import java.util.\*;  
import java.io.\*;  
public class PDemo  
{  
public static void main(String rags[])  
{  
try  
{  
Properties p= new Properties();  
p.load(new FileInputStream(“db.properties”));

String driver=p.getProperty(“driver”);  
String url=p.getProperty(“url”);  
String user=p.getProperty(“user”);  
String pass=p.getProperty(“password”);  
Class.forName(driver);  
Connection con=DriverManager.getConnection(url,user,pass);  
Statement stmt=con.createStatement();  
System.out.println(“statement object created”);  
stmt.executeUpdate(“create table student(sno number,sname varchar2(30),saddr varchar2(45))”);  
System.out.println(“table created”);  
stmt.executeUpdate(“insert into student values(101,’ram’,’dsnr’)”);  
stmt.executeUpdate(“insert into student values(102,’raj’,’koti’)”);  
System.out.println(“records inserted”);  
//process the result

ResultSet rs=stmt.executeQuery(“select \* from student”);  
while(rs.next()==true)  
{  
System.out.println(rs.getInt(1)+” “+rs.getString(2)+” “+rs.getString(3));  
}  
//close jdbc objects  
rs.close();  
stmt.close();

con.close();  
}//try  
catch(SQLException se)  
{  
System.out.println(“database problems”+se);  
}  
catch(ClassNotFoundException ce)  
{  
System.out.println(“class not available”+ce);  
}  
catch(Exception e)  
{  
System.out.println(e);  
}  
}//main  
}//class  
/\*  
advantage of using properties file is ,we can migrate from one driver to another driver with out effecting the appliaction logic  
\*/

**database MYSQL**  
Developer(s) Oracle corporation  
Stable release 5.6  
Written in C, C++  
Operating system [Cross-platform]  
Type RDBMS(opensource)  
Website www.mysql.com  
dev.mysql.com  
defaultportno 3306  
defaultuser root/root

type4  
——-  
driverclass– com.mysql.jdbc.Driver  
url——-jdbc:mysql://localhost:3306/mysql  
jar file–mysql-connector-java-5.1.19-bin.jar  
—————————————————–  
mysql>show databases;  
mysql> use mysql  
Database changed  
mysql>show tables;  
mysql> create table student(sno int,name varchar(20));  
———————————————–  
package com.configindia.hyd;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
public class MySQLType4 {  
public static void main(String args[]) {

try {  
Class.forName(“com.mysql.jdbc.Driver”);  
System.out.println(“Driver is loaded”);  
Connection c = DriverManager.getConnection(“jdbc:mysql://localhost:3306/mysql”, “root”, “root”);  
System.out.println(“Connection created”);  
String s = “insert into student(sno,name) values (?,?)”;  
PreparedStatement ps = c.prepareStatement(s);  
ps.setInt(1, 101);  
ps.setString(2, “ram”);  
ps.execute();  
c.close();  
System.out.println(“Inserted”);  
} catch (Exception e) {  
System.out.println(“Exception : ” + e);

}  
}

}

jdbc key:

1. Java Database Connectivity (**JDBC**) is a standard Java API to interact with relational databases form Java.**JDBC**has set of classes and interfaces which can use from Java app lication and talk to database without learning RDBMS details and using Database Specific JDBC Drivers. Pattern-bridge design pattern.
2. **:**The significances are given below:  
   -JDBC is the acronym stands for Java Database Connectivity.  
   -Java Database Connectivity (JDBC) is a standard Java API .  
   -It’s purpose is to interact with the relational databases in Java.  
   -JDBC is having a set of classes & interfaces which can be used from any Java application.  
   -By using the Database Specific JDBC drivers, it interacts with a database without the applications of RDBMS.
3. —JDBC-ODBC Bridge plus ODBC driver, also called Type 1

— Native-API, partly Java driver, also called Type 2

— JDBC-Net, pure Java driver, also called Type 3

— Native-protocol, pure Java driver, also called Type 4

and fastest—Type4 driver.

1. No. You can open only one Statement object per connection when you are using the JDBC-ODBC Bridge.
2. ODBC-open database connectivity OCL-oracle callable interface
3. ODBC is an open interface which can be used by any application to communicate with any database

system, while JDBC is an interface that can be used by Java applications to access databases.

7. DriverManager, Driver, Connection, Statement, ResultSet

1. Following are the basic steps to create a JDBC application:
2. Import packages containing the JDBC classes needed for database programming.
3. Register the JDBC driver, so that you can open a communications channel with the database.
4. Open a connection using the DriverManager.getConnection () method.
5. Execute a query using an object of type Statement.
6. Extract data from result set using the appropriate ResultSet.getXXX () method.
7. Clean up the environment by closing all database resources relying on the JVM’s garbage collection.
8. No
9. No

11.Type3-driver

1. yes
2. Class.forName(“—-”);
3. ClassNotFoundException
4. “No suitable driver” is occurs during a call to the DriverManager.getConnection method, may be of any of the following reason:

-Due to failing to load the appropriate JDBC drivers before calling the getConnection method.

-It can be specifying an invalid JDBC URL, one that is not recognized by JDBC driver.

-This error can occur if one or more the shared libraries needed by the bridge cannot be loaded.

1. Connection interface consists of methods for contacting a database. The connection object represents communication context.
2. Connection con = DriverManager.getConnection();
3. Statement, PreparedStatement, CallableStatement
4. Statement stmt = con.createStatement();
5. missing question
6. a. int b. boolean c. ResultSet
7. stmt.close();(within try/catch block)
8. sun.jdbc.odbc.driver.JdbcOdbcDriver
9. 1.It is a pure java driver. 2. It is lightweight and easy to install. 3.This driver does not requires software to be installed on client side. 4.This driver communicates directly with datasource using javasockets.

Disadvantage**:**Needs seprate driver for each database**.**

1. ojdbc14.jar/ojdbc5.jar
2. jdbc:oracle:thin:@localhost:1521:XE
3. These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data. The java.sql.ResultSet interface represents the result set of a database query. Methods: next(), hasMoreElements()
4. Statement   stmt = conn.createStatement( );

ResultSet rs = stmt.executeQuery();

1. DatabaseMetaData
2. getColumnName()
3. JDBC API has two Metadata interfaces : **DatabaseMetaData & ResultSetMetaData**. The meta data provides comprehensive information about the database as a whole. The implementation for these interfaces is implemented by database driver vendors to let users know the capabilities of a Database.
4. ResultSet rs = stmt.executeQuery();
5. execute(), executeUpdate(), executeQuery(), close().
6. In order to prevent SQL Injection attacks in Java, PreparedStatement doesn’t allow multiple values for one placeholder (?) who makes it tricky to execute SQL query with IN clause. Following example of SQL query with IN clause using prepared Statement will not work in Java:

36.Callable statements are mainly used in the JDBC applications.  
Callable statements are used to invoke stored procedures  
This is mainly used in functions.

37.static

1. String SQL = “any sql query”;

PreparedStatement  pstmt = conn.prepareStatement(SQL);

1. prepareCall(“—-”)
2. IN
3. Create or replace Procedure-name (Input parameters ,Output Parameters (If required))

As

Begin

Sql statement used in the stored procedure

End.

1. 2
2. 2 ways
3. Prepared statements offer better performance, as they are pre-compiled. Prepared statements reuse the same execution plan for different arguments rather than creating a new execution plan every time. Prepared statements use bind arguments, which are sent to the database engine. This allows mapping different requests with same prepared statement but different arguments to execute the same execution plan. Prepared statements are more secure because they use bind variables, which can prevent SQL injection attack.
4. interface
5. getColumnCount(), getColumnName(), getColumnSize()
6. getDatabaseProductName(),getDriverName(),getDriverVersion(),getDatabaseProductVersion()
7. Boolean
8. getColumnCount()
9. IN, OUT,INOUT

51.yes

52.yes

1. 2
2. Type2 Driver
3. BLOB, CLOB

56.The JDBC driver performance depends on:  
– The driver code quality  
-The driver code size  
-The database server & its load capability  
-The network topology  
-The number of times the request is being translated to a different API.

57.Callable statements are mainly used in the JDBC applications.  
Callable statements are used to invoke stored procedures  
This is mainly used in functions.

1. 3 types: ResultSet.TYPE\_FORWARD\_ONLY, ResultSet.TYPE\_SCROLL\_INSENSITIVE,

ResultSet.TYPE\_SCROLL\_SENSITIVE

1. 1.Scrollable ResultSet. 2. Methods to update ResultSet or Database table. 3. Batch updates.
2. Supports datatypessuch as BLOB,CLOB,Array and Ref.

60.Statement stmt =  con.createStatement(ResultSet.TYPE\_SCROLL\_SENSITIVE,

ResultSet.CONCUR\_UPDATABLE);

1. SQLException, ClassNotFoundException, BatchUpdateException
2. The difference between these two statements are  
   1) SQL Query will be compiled every time it is executed. whereas [Stored Procedures](http://www.geekinterview.com/Interview-Questions/J2EE/JDBC)are compiled only once

when they are executed for the first time.  
2) Stored Procedure is efficient & fast execution than SQL query.

1. beforeFirst(), afterLast(), first(), last(), absolute(int row), relative(int row), previous(), next(), getRow(),

.

1. moveToCurrentRow(),deleteRow(),moveToInsertRow(), moveToCurrentRow(),insertRow()

65.BLOB(BinayLarge Object) and CLOB(CharacterLargeObject) are SQL2 datatypes, which are introduced

in JDBC 2.0 version.

1. Sometimes the table will generate a primary key, then you can use Statement getGeneratedKeys () method to

obtain the automatically generated primary key values.

1. ResultSet
2. In order to prevent SQL Injection attacks in Java, PreparedStatement doesn’t allow multiple values for one

placeholder (?) who makes it tricky to execute SQL query with IN clause. Following example of SQL query with

INclause using prepared Statement will not work in Java.

1. A transaction is a logical unit of work. To complete a logical unit of work, several actions may need to be taken against a database. Transactions are used to provide data integrity, correct application semantics, and a consistent view of data during concurrent access.
2. Follow steps as below:

//turn off the implicit commit

Connection.setAutoCommit(false);

//your insert/update/delete goes here

Connection.Commit();

71.

72.Class.forName(“—–“);

Connection con = DriverManager.getConnection(“——-“,  “—“,”—-“);

DatabaseMetaData dbmd = con.getMetaData();

1. setBlob()

74.getConnection()—->getConnection() method in DriverManager class.Which is used to get object of Connection

interface.

1. ResultSetMetaData rsmd = rs.getMetaData();
2. executeUpdate()

77.yes

1. ResultsetMetaData/DatabaseMetaData
2. Blob b1 = conn.createBlob();

80.**:**1.Addition ofSavePoints.

2.Defines several standard connection pool properties such as maxStatement,initialPoolsize

1. Supports multiple open ResultSet.
2. Two new datatypes have been added java.sql.Types.Datalink and java.sql.Types.Boolean.
3. When a connection is created, it is in auto-commit mode. This means that each individual SQL statement is treated as a transaction and will be automatically committed right after it is executed. By setting auto-commit to false no SQL statements will be committed until you explicitly call the commit method.
4. Following are the reasons:

-To increase performance.

-To maintain the integrity of business processes.

-To use distributed transactions

83.A savepoint marks a point that the current transaction can roll back to. Instead of rolling all of its changes back, it can choose to roll back only some of them. For example, suppose you:

-start a transaction

-insert 10 rows into a table

-set a savepoint

-insert another 5 rows

-rollback to the savepoint

-commit the transaction

1. SQLWarning objects are a subclass of SQLException that deal with database access warnings. Warnings do not stop the execution of an application, as exceptions do. They simply alert the user that something did not happen as planned. A warning can be reported on a Connection object, a Statement object (including PreparedStatement and CallableStatement objects), or a ResultSet object. Each of these classes has a getWarnings method.

85.Batch Processing allows you to group related SQL statements into a batch and submit them with one call to the database.

1. Typical sequences of steps to use Batch Processing with Statement or PrepareStatement Object are:
2. In case of Batch processing using PrepareStatement object, create SQL statements with placeholders.
3. Create a Statement or PrepareStatement object using either createStatement() or prepareStatement() methods respectively.
4. Set auto-commit to false using setAutoCommit().
5. Add as many as SQL statements you like into batch using addBatch() method on created statement object.
6. Execute all the SQL statements using executeBatch() method on created statement object.
7. Finally, commit all the changes using commit() method.

1. Follow steps as below:

//turn off the implicit commit

Connection.setAutoCommit(false);

//your insert/update/delete goes here

Connection.Commit();

1. java.sql.Blob has better performance as it does not extract any data from the database until you explicitly ask it to.
2. java.sql.Clob has better performance as it does not extract any data from the database until you explicitly ask it to.
3. Make sure the resultset is updatable.  
   1. move the cursor to the specific position.  
   uprs.moveToCurrentRow();

2. set value for each column.  
uprs.moveToInsertRow();//to set up for insert  
uprs.updateString(“col1” “strvalue”);  
uprs.updateInt(“col2”, 5);

3. call inserRow() method to finish the row insert process.  
uprs.insertRow();

To delete a row: move to the specific position and call deleteRow() method:  
uprs.absolute(5);  
uprs.deleteRow();//delete row 5

To see the changes call refreshRow();  
uprs.refreshRow();

1. conn.setAutoCommit(true);
2. jdbc:oracle:thin:@localhost:1521: Databasename.
3. char and varchar.

94.Apache POI is a powerful Java library to work with different Microsoft Office file formats such as Excel, Power point, Visio, MS Word etc. The name POI was originally an acronym for **Poor Obfuscation Implementation**.

Use: to read/ write excel file in java using POI.

1. Follow the steps below:

**1st STEP:** First setup the new ODBC datasource. Goto Administrative Tools->Data Sources (ODBC)->System DSN tab->Add->Driver do Microsoft Excel(\*.xls)->Finish. Now give the Data Source Name (SampleExcel) & Description. Next, click Select Workbook and point to your excel sheet.

**2nd STEP:** In the code make to following code additions:

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

Connection conn=DriverManager.getConnection(“jdbc:odbc:SampleExcel”,””,””);

stmt=conn.createStatement();

sql=”select \* from [Sheet1$]”;

rs=stmt.executeQuery(sql);

Where Sheet1 is the excel sheet name.

1. Placeholder is used for supplying value explicitly.
2. 1 JDBC1.0 2 JDBC2.0 3 JDBC3.0  4 JDBC4.0
3. Here are few advantages of JDBC 4.0

-Auto-loading by JDBC driver class.  
-Enhanced Connection management  
-RowId SQL enabled.  
-DataSet implemented by SQL by using Annotations  
-Enhancements of SQL exception handling  
-Supporting SQL XML files.

1. Follow the steps below:

**1st STEP:** First setup the new ODBC datasource. Goto Administrative Tools->Data Sources (ODBC)->System DSN tab->Add->Driver do Microsoft Excel(\*.xls)->Finish. Now give the Data Source Name (SampleExcel) & Description. Next, click Select Workbook and point to your excel sheet.

**2nd STEP:** In the code make to following code additions:

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

Connection conn=DriverManager.getConnection(“jdbc:odbc:SampleExcel”,””,””);

stmt=conn.createStatement();

sql=”select \* from [Sheet1$]”;

rs=stmt.executeQuery(sql);

Where Sheet1 is the excel sheet name

1. absolute()—–>Moves the cursor to the specified row. relative()—->Moves the cursor the given number of rows forward or backwards from where it currently is pointing.