CS 7700  Transaction Design for Microsoft Access Database with JDBC

Purpose

The purpose of this tutorial is to introduce the process of developing transactions for a Microsoft Access Database with Java Database Connectivity (JDBC).

Problem

The XYZ Corporation needs a simple database system to store departments, employees, projects, and dependents information. To support the corporation’s business objectives, users need to be able to perform the following transactions:

- Add new records into a current table
- Delete records from a current table
- Modify records in a current table
- Retrieve records from a current table
Database Schema

This company relational database schema comes from the textbook (Fundamentals of Database systems, 6th Edition, by Ramez Elmasri and Shamkant Navathe).

**EMPLOYEE**

<table>
<thead>
<tr>
<th>Fname</th>
<th>Minit</th>
<th>Lname</th>
<th>Ssn</th>
<th>Bdate</th>
<th>Address</th>
<th>Sex</th>
<th>Salary</th>
<th>Super_ssn</th>
<th>Dno</th>
</tr>
</thead>
</table>

**DEPARTMENT**

<table>
<thead>
<tr>
<th>Dname</th>
<th>Dnumber</th>
<th>Mgr_ssn</th>
<th>Mgr_start_date</th>
</tr>
</thead>
</table>

**DEPT_LOCATIONS**

<table>
<thead>
<tr>
<th>Dnumber</th>
<th>Dlocation</th>
</tr>
</thead>
</table>

**PROJECT**

<table>
<thead>
<tr>
<th>Pname</th>
<th>Pnumber</th>
<th>Plocation</th>
<th>Dnum</th>
</tr>
</thead>
</table>

**WORKS_ON**

<table>
<thead>
<tr>
<th>Essn</th>
<th>Pno</th>
<th>Hours</th>
</tr>
</thead>
</table>

**DEPENDENT**

<table>
<thead>
<tr>
<th>Essn</th>
<th>Dependent_name</th>
<th>Sex</th>
<th>Bdate</th>
<th>Relationship</th>
</tr>
</thead>
</table>
1. Introduction

Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases which also called the data sources. Data source accesses within the Java program require JDBC drivers.

Open Database Connectivity (ODBC) is a standard protocol for programs (such as Microsoft Access) to obtain access to SQL database servers (such as Microsoft SQL Server or Oracle).

One of the JDBC drivers, JDBC-ODBC bridge, is applied in this tutorial. The JDBC-ODBC bridge employs an ODBC driver to connect to a target database and translates JDBC method calls into ODBC function calls. It is usually used for a database lacking a JDBC driver.

2. Create Database and tables

Look over the CS 4700/6700 tutorial for the step by step description for the process of creating a database in Microsoft Access.

3. Connect JDBC to Microsoft Access

For a 64-bit operating system with 64-bit Microsoft Access, the 64-bit Microsoft Access Database Engine 2010, AccessDatabaseEngine_x64.exe, should be installed first. It can be downloaded at http://www.microsoft.com/en-us/download/details.aspx?id=13255
Open Control Panel, and select Administrative Tools, then select and open Data Sources (ODBC).

**Note:** A 64-bit Windows operating system has two odbcad32.exe files: 32-bit and 64-bit. The default shortcut in Administrative Tools is for the 64-bit one. 32-bit Microsoft Access users need to open the 32-bit ODBC Administrator. To do so, right click on “Data Sources (ODBC)” and go to its properties. In properties, change following terms and click “OK”:

- **Target** from “%windir%\system32\odbcad32.exe” to “%windir%\SysWOW64\odbcad32.exe”
- **Start in** from “%windir%\system32” to “%windir%\SysWOW64”

After these path changes, 32-bit ODBC Administrator can be open by the shortcut in Administrative Tools.
Once open the **Data Sources (ODBC)**, in the pop-up **ODBC Data Source Administrator**, go to **System DSN** then click on **Add** button.
Then in the pop-up window, select **Microsoft Access Driver (*.mdb, *.accdb)** then click on **Finish**.

On the pop-up **ODBC Microsoft Access Setup** page, type your desired **Data Source Name** (**JDBCdsn** is the name used here). This is the name you will be using in the Java code to connect to the database, so ideally try to keep the database name and the DSN name to be the same. And then click on **Select** button.
Then find and select the Access database file you created previously, like the Company.accdb here and click OK. Then click OK in the ODBC Microsoft Access Setup page.

Finally, the Data Source you created appears in the ODBC Data Source Administrator as shown below.
4. Design Transactions

Now the next step is to design and execute transactions based on a Java program and embed SQL queries. Before being able to process JDBC function calls with Java, it is necessary to import the JDBC class libraries java.sql.* which can be found at http://docs.oracle.com/javase/7/docs/api/java/sql/Statement.html

Note:
1. The JDBC-ODBC Bridge has been removed in JDK 8. So JDK 7 or less should be used as the Java platform.
2. If 32-bit Microsoft Access Database (or Data Source) is used, then keep your Microsoft Access Driver (or ODBC Driver), JDK and even Java IDE all in 32-bit. While if 64-bit Data source is in use, keep them all in 64-bit.

Generally, a complete transaction includes five main parts. They are: loading the driver, establishing a connection to data source, executing queries, committing transactions, and closing the connection.

1) Loading the driver

To establish a connection with the data source, first you must load the driver.

```java
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    System.out.println("driver loaded");
} //end of try
catch(java.lang.ClassNotFoundException e) {
    System.err.print("ClassNotFoundException: ");
    System.err.println(e.getMessage());
} //end of catch
```

2) Establishing a connection

When a connection is created, it is in auto-commit mode. This means that each individual SQL statement is treated as a transaction and is automatically committed right after it is executed. The way to allow two or more statements to be grouped into a transaction is to disable auto-commit mode. Once auto-commit mode is disabled, no SQL statements are
committed until you call the method `commit` explicitly. This is demonstrated in the following lines of codes, where `conn` is an active connection:

```java
try {
    conn = DriverManager.getConnection("jdbc:odbc:JDBCdsn");
    System.out.println("Connected to the database");
    conn.setAutoCommit(false); // This removes the auto-commit mode
} // end of try
catch (SQLException se) {
    System.out.println("Not connected to database");
    System.out.println(se);
} // end of catch
```

3) Executing queries

In JDBC, the `Statement` Objects define the methods and properties than enable you to send SQL commands and receive data from your database. First, a `Statement` object should be create using the `createStatement()` method. Once a `Statement` is created, there are three execute methods can be used to execute a SQL statement:

- `boolean execute(String SQL)` : Returns a boolean value of true if a `ResultSet` object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- `int executeUpdate(String SQL)` : Returns the numbers of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an insert, update, or delete statement.
- `ResultSet executeQuery(String SQL)` : Returns a `ResultSet` object. Use this method when you expect to get a result set, as you would with a SELECT statement.

A `ResultSet` object is a table of data representing a database result set. The `ResultSet` interface provides methods for retrieving and manipulating the results of executed queries.
The data in a ResultSet object is accessed through a cursor, which points to its current row of data in the ResultSet. When a ResultSet object is first created, the cursor is positioned before the first row. The next method moves the cursor to the next row.

The ResultSet interface provides getter methods (getBoolean, getLong, and so on) 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. Columns are numbered from 1. Column names used as input to getter methods are case insensitive. When a getter method is called with a column name and several columns have the same name, the value of the first matching column will be returned.

The getter method of the appropriate type retrieves the value in each column. For example, the first column in each row of ResultSet rs is “Dname”, which stores a value of SQL type TEXT. The method for retrieving the value is getString. The second column in each row stores a value of SQL type INT, and the method for retrieving values of that type is getInt. Each time the method next is invoked, the next row becomes the current row, and the loop continues until there are no more rows in rs. An example code segment for a select operation is shown below.

```java
try {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM Department");

    while (rs.next()) {
        System.out.print(rs.getString("Dname"));
        System.out.print("\t");
        System.out.print(rs.getInt("Dnumber"));
        System.out.print("\t");
        System.out.println(rs.getString("Mgr_ssn"));
        System.out.print("\t");
        System.out.println(rs.getDate("Mgr_start_date"));
    }
}
```

rs.close();
stmt.close();
```

```
catch (Exception excep){
    System.out.println("Failed to execute query\n"+excep);
}
```
For insert, delete and update operations, two argument should be added to the createStatement method. The first argument indicates the type of a ResultSet object: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE. The second argument is one of two ResultSet constants for specifying whether a result set is read-only or updatable: CONCUR_READ_ONLY and CONCUR_UPDATABLE. The point to remember here is that if you specify a type, you must also specify whether it is read-only or updatable. Also, you must specify the type first, because both parameters are of type int, the compiler will not complain if you switch the order. Specifying the constant TYPE_FORWARD_ONLY creates a nonscrollable result set, that is, one in which the cursor moves only forward. If you do not specify any constants for the type and updatability of a ResultSet object, you will automatically get one that is TYPE_FORWARD_ONLY and CONCUR_READ_ONLY. The details of three ResultSet type are given below.

- **TYPE_FORWARD_ONLY** — The result set is not scrollable; its cursor moves forward only, from before the first row to after the last row. The rows contained in the result set depend on how the underlying database materializes the results. That is, it contains the rows that satisfy the query at either the time the query is executed or as the rows are retrieved.

- **TYPE_SCROLL_INSENSITIVE** — The result set is scrollable; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position.

- **TYPE_SCROLL_SENSITIVE** — The result set is scrollable; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position.

An example code segment for insert, delete and update operations is given below.
try {
    Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                            ResultSet.CONCUR_UPDATABLE);
    stmt.executeUpdate(query);
    System.out.println("Executed the query successfully");
    stmt.close();
} // end of try
catch (Exception excep){
    System.out.println("Failed to execute query");
    // end of catch

4) Committing transactions

After all the SQL statements are executed, method commit should be call. Then all statements executed after the previous call to the method commit are included in the current transaction and committed together as a unit.

try {
    conn.commit();
    System.out.println("Changes successfully committed");
} // end of try
catch (Exception e){
    System.out.println("Failed to commit changes: 
    + e);
    System.exit(0);
} // end of catch

5) Closing the connection

Once the transactions are finished, the connection with the data source can be closed by the following code.

try {
    conn.close();
    System.out.println("Connection successfully closed");
} // end of try
catch (Exception excep){
    System.out.println("Unable to close connection: 
    + excep);
    System.exit(0);
} // end of catch
5. Example:

A sample code which can be used to insert, delete, update and retrieve records in the department table is given below. Please pay attention to the previously described parts and see how they can be put together in a program.

```java
import java.io.*;
import java.sql.*;
import java.text.*;

public class newDatabase {
    static boolean exit = false;
    static Connection conn = null;

    public static void main(String[] args) {
        String choice;
        int N = 0;
        
        while(!exit) {
            displayConsole();
            choice = captureInput();
            try{
                N = Integer.parseInt(choice);
            } //end of try
            catch(Exception e){
                System.err.println("Enter valid option"+e);
                displayConsole();
            } //end of catch
            switchCase(N);
        } //end of while
    } // end of main

    public static void displayConsole() {
        //boolean exit = false;
        System.out.println("***************************************************
        ***
        Welcome to Database tutorial 
        ******************************************************
        1. Insert record to database
        2. Delete record from database
        3. Update record in database
        4. Retrieve record in database
        5. Exit
        Enter your choice:");
    }
```
public static String captureInput() {
    String choice = "";
    try {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        choice = in.readLine();
    } //try
    catch (IOException e) {
        System.out.println("Enter valid numeric choice");
    } //end of catch
    return choice;
} //end of captureInput

public static java.sql.Date Inputdate() {
    java.util.Date utildate = null;
    java.sql.Date sqldate = null;
    SimpleDateFormat formatter = new SimpleDateFormat("MM/dd/yyyy");
    try {
        String userInput = captureInput();
        utildate = formatter.parse(userInput);
        sqldate = new java.sql.Date(utildate.getTime());
    } catch (ParseException e) {
        // execution will come here if the String that is given
        // does not match the expected format.
        e.printStackTrace();
    }
    return sqldate;
} //end of captureInput

public static void switchCase(int N) {
    switch (N) {
    case 1:
    {
        String Dname;
        int Dnumber;
        String Mg_ssn;
        java.sql.Date Mgr_start_date;
        boolean quit=false;
        while(!quit)
        {
            System.out.println("Enter Department Name:");
            Dname = captureInput();
            System.out.println("Enter Department Number:");
            Dnumber = Integer.parseInt(captureInput());
            System.out.println("Enter Department Manager SSN");
        }
    }
}
Mg_ssn = captureInput();
System.out.println("Enter Manage Start Date (mm/dd/yyyy):");  
Mgr_start_date = Inputdate();

String query="INSERT INTO Department VALUES('"+Dname+"','"+Dnumber+"','"+Mg_ssn+"','"+Mgr_start_date+"');
//System.out.println(query);
loadDriver();
openConnection();
try {
    Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATA BLE);
    stmt.executeUpdate(query);
    System.out.println("Executed the query successfully");
    ResultSet rs = stmt.executeQuery("SELECT *
FROM Department");
    while( rs.next() ){
        System.out.println(rs.getString("Dname") );
        System.out.println("\t");
        System.out.println(rs.getInt("Dnumber") );
        System.out.println("\t");
        System.out.println( rs.getString("Mgr_ssn") );
        System.out.println("\t");
        System.out.println( rs.getDate("Mgr_start_date") );
    } //end of while
    rs.close();
    stmt.close();
} //end of try
catch (Exception except){
    System.out.println("A record with this primary key value already exists or table to which you are trying to make changes is opened\n" + except);
    System.out.println("Failed to execute query");
} //end of catch
commitDb();
closeConnection();
System.out.println("Do you want to go back to main menu, press y to do so");
String ch=captureInput();
if(ch.equals("Y") || ch.equals("y")){
    quit=true;
} //end of if
} //end of while
break;
} //end of case 1

case 2:{
String Dname;
int Dnumber=0;
String Mgr_ssn;
java.sql.Date Mgr_start_date;
int i;
String ch;
boolean quit=false;

while(!quit)
{
    boolean done=false;
    LoadDriver();
    openConnection();
    try {
        Statement stmt =
    conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = stmt.executeQuery("SELECT * FROM Department");
        while( rs.next() ){
System.out.println(rs.getString("Dname")\"\t\"+rs.getInt("Dnumber")\"\t\"+rs.getString("Mgr_ssn")\"\t\"+rs.getDate("Mgr_start_date")");  
} //end of while 
while(!done)
{
    rs.first();
    System.out.println("Enter a valid department number of the record you want to delete");
    i = Integer.parseInt(captureInput());
    while( rs.next() ){
        if(rs.getInt("Dnumber")==i)
        {
            done=true;
            Dname=rs.getString("Dname");
            Dnumber=i;
            Mgr_ssn=rs.getString("Mgr_ssn");
            Mgr_start_date=rs.getDate("Mgr_start_date");
        } //end of if
    } //end of rs while 
} //end of done while

String query="DELETE FROM Department WHERE Dnumber="+Dnumber+""; 
//System.out.println(query);
stmt.executeUpdate(query);
System.out.println("Executed the query successfully");
rs = stmt.executeQuery("SELECT * FROM Department");

while( rs.next() ){
    System.out.print(rs.getString("Dname");
    System.out.print("\t");
    System.out.print(rs.getInt("Dnumber");
}
System.out.println("\t");

System.out.println( rs.getString("Mgr_ssn");
System.out.println("\t");

System.out.println( rs.getDate("Mgr_start_date");
  } //end of while
  rs.close();
  stmt.close();
} //end of try

  catch (Exception excep){
    System.out.println("The table to which you are trying
to make changes is open, please close\n" + excep);
    System.out.println("Failed to execute query");
  } //catch
  commitDb();
  closeConnection();
  System.out.println("Do you want to go back to main menu, press y to do so");
  String op=captureInput();
  if(op.equals("Y") || op.equals("y"){ quit=true;
    } //end of if
  } //end of while
  break;
} //end of case 2

  case 3:{
    String Dname=null;
    int Dnumber=0;
    String Mgr_ssn=null;
    java.sql.Date Mgr_start_date=null;
    java.sql.Date newdate;
    int i=0;
    String ch;
    String query;
    boolean quit=false;

    while(!quit)
    {
      boolean done=false;
      LoadDriver();
      openConnection();
      try {
        Statement stmt =
        conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_UPDATABLE);
        ResultSet rs = stmt.executeQuery("SELECT * FROM Department");
        while( rs.next() ){
          System.out.println(rs.getString("Dname")+"\t"+rs.getInt("Dnumber")+"\t"+rs.getString("Mgr_ssn")+"\t"+rs.getDate("Mgr_start_date")
          )
        } //end of while
      } //end of try
    } //end of while
  } //end of case 3
{
    rs.first();
    System.out.println("Enter a valid department number of the record you want to update: ");
    i = Integer.parseInt(captureInput());
    while(rs.next()){
        if(rs.getInt("Dnumber") == i)
        {
            done=true;
            Dname = rs.getString("Dname");
            Dnumber = i;
            Mgr_ssn = rs.getString("Mgr_ssn");
            Mgr_start_date = rs.getDate("Mgr_start_date");
        }
    }
    System.out.println("Enter a new department number you want to update the record to:");
    i = Integer.parseInt(captureInput());
    Dnumber = i;
    query = "UPDATE Department SET Dnumber='"+Dnumber+"' WHERE Dname='"+Dname+"';"
    //System.out.println(query);
    stmt.executeUpdate(query);
    System.out.println("DO you want to update the Dname of the record, press y to update");
    ch = captureInput();
    if(ch.equals("y") || ch.equals("Y"))
    {
        System.out.println("Enter a new department name you want to update the record to:");
        ch = captureInput();
        Dname = ch;
    }
    System.out.println("DO you want to update the Mgr_ssn of the record, press y to update");
    ch = captureInput();
    if(ch.equals("y") || ch.equals("Y"))
    {
        System.out.println("Enter a new manager ssn you want to update the record to:");
        ch = captureInput();
        Mgr_ssn = ch;
    }
    System.out.println("DO you want to update the Mgr_start_date of the record, press y to update");
    ch = captureInput();
    if(ch.equals("y") || ch.equals("Y"))
    {
        System.out.println("Enter a new manage start date you want to update the record to:");
        newdate = Inputdate();
        Mgr_start_date = newdate;
    }
}*/
```
query="UPDATE Department SET 
Dname=’"+Dname+"’,Mgr_ssn=’"+Mgr_ssn+"’,Mgr_start_date=’"+Mgr_start_date+"’
WHERE Dnumber=’"+Dnumber+"’;

//System.out.println(query);
stmt.executeUpdate(query);
System.out.println("Executed the query successfully");
rs = stmt.executeQuery("SELECT * FROM Department");
while( rs.next() ){
    System.out.print(rs.getString("Dname")) ;
    System.out.print("\t");
    System.out.print(rs.getInt("Dnumber"));
    System.out.print("\t");
    System.out.println( rs.getString("Mgr_ssn")) ;
    System.out.print("\t");
    System.out.println( rs.getDate("Mgr_start_date"));
}
rs.close();
stmt.close();
} //end of try
catch (Exception excep){
    System.out.println("A record with this primary key value already exists or table to which you are trying to make changes is opened\n" + excep);
    System.out.println("Failed to execute query");
} //catch
commitDb();
closeConnection();
System.out.println("Do you want to go back to main menu, press y to do so");
String op=captureInput();
if(op.equals("Y") || op.equals("y")){
    quit=true;
} //end of if
} //end of while
break;
} //end of case 3

} //end of case 4:
{
    loadDriver();
    openConnection();
    try{
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Department");

        while( rs.next() ){
            System.out.print(rs.getString("Dname")) ;
            System.out.print("\t");
            System.out.print(rs.getInt("Dnumber"));
            System.out.print("\t");
        }

    } //end of while
} //end of try
catch (Exception excep){
    System.out.println("A record with this primary key value already exists or table to which you are trying to make changes is opened\n" + excep);
    System.out.println("Failed to execute query");
} //catch
commitDb();
closeConnection();
System.out.println("Do you want to go back to main menu, press y to do so");
String op=captureInput();
if(op.equals("Y") || op.equals("y")){
    quit=true;
} //end of if
} //end of while
break;
} //end of case 3

case 4:
{
    loadDriver();
    openConnection();
    try{
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Department");

        while( rs.next() ){
            System.out.print(rs.getString("Dname")) ;
            System.out.print("\t");
            System.out.print(rs.getInt("Dnumber"));
            System.out.print("\t");
        }

    } //end of while
} //end of try
catch (Exception excep){
    System.out.println("A record with this primary key value already exists or table to which you are trying to make changes is opened\n" + excep);
    System.out.println("Failed to execute query");
} //catch
commitDb();
closeConnection();
System.out.println("Do you want to go back to main menu, press y to do so");
String op=captureInput();
if(op.equals("Y") || op.equals("y")){
    quit=true;
} //end of if
} //end of while
break;
} //end of case 3

} //end of case 4:
{
    loadDriver();
    openConnection();
    try{
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM Department");

        while( rs.next() ){
            System.out.print(rs.getString("Dname")) ;
            System.out.print("\t");
            System.out.print(rs.getInt("Dnumber"));
            System.out.print("\t");
        }

    } //end of while
} //end of try
catch (Exception excep){
    System.out.println("A record with this primary key value already exists or table to which you are trying to make changes is opened\n" + excep);
    System.out.println("Failed to execute query");
} //catch
commitDb();
closeConnection();
System.out.println("Do you want to go back to main menu, press y to do so");
String op=captureInput();
if(op.equals("Y") || op.equals("y")){
    quit=true;
} //end of if
} //end of while
break;
} //end of case 3

...
System.out.println( rs.getString("Mgr_ssn"));
System.out.println("\t");
System.out.println( rs.getDate("Mgr_start_date"));
} //end of while
rs.close();
stmt.close();
} //end of try
catch( Exception excep){
    System.out.println("Failed to execute query\n"+excep);
    } //catch
commitDb();
closeConnection();
System.out.println("Do you want to go back to main menu, press any key to do so");
String op=captureInput();
break;
} //end of case 4
case 5:{
    exit=true;
    System.out.println("Have a nice day!");
    break;
} //end of case 5
} //end of switch
} //end of switchCase

public static void loadDriver(){
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        System.out.println("driver loaded");
    } //end of try
    catch(java.lang.ClassNotFoundException e) {
        System.err.print("ClassNotFoundException: ");
        System.err.println(e.getMessage());
    } //end of catch
} // end of loadDriver

public static void openConnection(){
    try {
        conn = DriverManager.getConnection("jdbc:odbc:JDBCdsn");
        System.out.println("Connected to the database");
        conn.setAutoCommit(false);
    } //end of try
    catch( SQLException se){
        System.out.println("Not connected to database");
        System.out.println(se);
    } //end of catch
} //end of openConnection
public static void commitDb(){
    try {
        conn.commit();
        System.out.println("Changes successfully committed");
    } //end of try
    catch (Exception e){
        System.out.println("Failed to commit changes: n" + e);
        System.exit(0);
    } //end of catch
} //end of commitDb

public static void closeConnection(){
    try {
        conn.close();
        System.out.println("Connection successfully closed");
    } //end of try
    catch (Exception excep){
        System.out.println("Unable to close connection: n" +
        excep);
        System.exit(0);
    } //end of catch
} //end of closeConnection