**CITC 2340 – Lab 8: Running SQL Using Java (35 Pts) Name: Mihaela McReynolds**

**To understand the concepts applied in this lab, please read/skim the related PowerPoint slides posted in D2L. We will be trying to modify and run Basic Example 1 during this lab. Copy this instruction document to your H: drive for editing and add your name to the top.**

1. From the course [Content] Labs module, download the *ojdbc7* file and the *DBPractice* folder to your H: drive or USB drive. Unzip the files into a folder named *JavaPractice*. Run the NetBeans IDE and choose menu item *File 🡪 Open Project*. Then browse for and select the *DBPractice* project that you just copied to your personal drive.
2. We will need to add the ODBC library to the environment to be linked into the program so that it will compile. One way to do this is to right click the +Libraries path in the left-hand project navigation bar, choose *Add JAR/Folder*; then browse/navigate to the location where you saved your copy of the *ojdbc7.jar* file. Open that *jar* file to include it in your environment.
3. From the *Run* menu, choose *Clean and Build* the project to be sure there aren’t any syntax errors.
4. Open the project source file and examine the code to get an idea of the types of tasks being tested in the program. Also, in SQL Developer, open the tables that this code is manipulating to examine the related metadata and data that currently exists.
5. Run the project to be sure you can get it to “work”. Make up data for a new customer that does NOT conflict with any existing customers and does NOT violate any of the size or data type constraints on the fields that will hold this data. Request assistance if you get stuck. Paste a copy of the bottom output window from NetBeans here (select it, copy, paste here):

run:

Added row to customers table

Updated row in customers table

Deleted row from customers table

Retrieved rows from customers table

customerId = 99

firstName = Mihaela

lastName = McReynolds

dob = 1987-11-15

dobTime = 00:00:00

dobTimestamp = 1987-11-15 00:00:00.0

phone = 423-857-6547

customerId = 1

firstName = Jean

lastName = Brown

dob = 1965-01-01

dobTime = 00:00:00

dobTimestamp = 1965-01-01 00:00:00.0

phone = 800-555-1211

customerId = 2

firstName = Cynthia

lastName = Green

dob = 1968-02-05

dobTime = 00:00:00

dobTimestamp = 1968-02-05 00:00:00.0

phone = 800-555-1212

customerId = 3

firstName = Steve

lastName = White

dob = 1971-03-16

dobTime = 00:00:00

dobTimestamp = 1971-03-16 00:00:00.0

phone = 800-555-1213

customerId = 4

firstName = Gail

lastName = Black

dob = null

dobTime = null

dobTimestamp = null

phone = 800-555-1214

Retrieved row from products table

product\_id = 3

product\_type\_id = 2

productIdInt = 3

priceDouble = 25.99

Created addresses table

Dropped addresses table

BUILD SUCCESSFUL (total time: 52 seconds)

1. Carefully examine the output produced by the program and compare it to the source code. Can you identify which lines of code produced each set of output? If not, ASK questions ☺
2. **Type your answer to the following question here**: When the code at lines 105-130 executes, how many rows should be returned and displayed from the select statement? Explain.

Looking at the table, it should return 5 rows: we had 5 to begin with, we added one, and we deleted another one.

1. **Type your answer to the following question here**: When the code at lines 149-153 executes, how many rows should be returned and displayed from the select statement? Explain why.

Because we specify “where product\_id = 3” and product\_id is a primary key in the products table, this should only return one row.

1. Add code that looks similar to lines 84-89 (or integrates with it) to change the last name of customer#1 to something creative. Run the project to see if your new code works.

// perform SQL UPDATE statement to modify the last\_name

// column of customer #1

lastName = "Cat";

myStatement.executeUpdate(

"UPDATE customers " +

"SET last\_name = '" + lastName + "' " +

"WHERE customer\_id = 1"

);

System.out.println("Updated row in customers table");

1. After you have tested your modifications to be sure they work, copy and paste only the NetBeans output that proves your changes worked here.

customerId = 1

firstName = Jean

lastName = Cat

dob = 1965-01-01

dobTime = 00:00:00

dobTimestamp = 1965-01-01 00:00:00.0

phone = 800-555-1211

1. Modify the existing code to create a new, interesting product (the PRODUCT\_TYPE\_ID should be one of the existing product type categories). Your new product name should use your first name as the prefix (e.g. SHARON MUSIC). Your code should be closely modeled after the code I gave you that INSERTs a new customer (approximately at lines 74-79).
2. Test the program to see if it prints your new product record. Copy and paste only the NetBeans output that proves your changes from step#11 worked here.
3. Close the project in NetBeans and exit NetBeans.
4. Save this Word document as *JavaSQL\_lab\_yourname.docx* and submit it to the *Java Lab* Assignments submission folder. Using Windows File Explorer, navigate to your *DBPractice* folder 🡪 *src* subfolder 🡪 *dbpractice* subfolder and upload *BasicExample1.java* to the *Java Lab* Assignments submission folder.

\*\*\*Challenge (optional for “fun”): Review the *BasicExample2.java* and *BasicExample3.java*, included in the *DBPractice* project folder. Set each one as the main program and compile and run the code (be sure to change usernames and passwords first). Examine the code to get a better understanding of what’s happening. Try modifying the code to implement a set of other useful SQL statements. This is great experience to have and I encourage you to “play” with it more when you have the time.

/\*

BasicExample1.java shows how to:

- import the JDBC packages

- load the Oracle JDBC drivers

- connect to a database

- perform DML statements

- control transactions

- use ResultSet objects to retrieve rows

- use the get methods

- perform DDL statements

Modified by: Mihaela McReynolds

Modification date: 4/25/2019

Modification reason: Lab 8 practice

\*/

package dbpractice;

// import the JDBC packages

import java.sql.\*;

// import oracle.jdbc.\*; -- auto imports in Java 6.x and higher

import javax.swing.\*;

import java.awt.\*;

public class BasicExample1 {

public static void main (String args []) {

// declare Connection and Statement objects

Connection myConnection = null;

Statement myStatement = null;

String userName, pw; // local variables

try {

// register the Oracle JDBC drivers

DriverManager.registerDriver(

new oracle.jdbc.OracleDriver() // make a new Oracle driver object

);

// Prompt for Oracle username and password instead of hard-coding it into

// the program.

userName = JOptionPane.showInputDialog("Enter Oracle username: ");

pw = JOptionPane.showInputDialog("Enter Oracle password: ");

// create a Connection object, and connect to the database

// using the Oracle JDBC Thin driver; we have to include the IP address,

// the port and the SID (service ID)

myConnection = DriverManager.getConnection(

"jdbc:oracle:thin:@198.146.192.57:1521:orcl", // ip address. port.SID

userName,

pw

);

// should use: if (myConnection is not null)

// disable auto-commit mode (in real life, we might want to set to TRUE

myConnection.setAutoCommit(false); // send an SQL\*Plus command to Oracle

// create a Statement object

myStatement = myConnection.createStatement();

JOptionPane.showMessageDialog(null,"Getting ready to add a new customer to CUSTOMERS table.");

// create variables and objects used to represent the fields in our table

// Prompt for those values so that we can

int customerId = Integer.parseInt(JOptionPane.showInputDialog("Enter integer customer ID"));

String firstName = JOptionPane.showInputDialog("Enter customer first name");

String lastName = JOptionPane.showInputDialog("Enter customer last name");

java.sql.Date dob = java.sql.Date.valueOf(

JOptionPane.showInputDialog("Enter customer first date of birth using format yyyy-mm-dd"));

java.sql.Time dobTime;

java.sql.Timestamp dobTimestamp;

String phone = JOptionPane.showInputDialog(

"Enter customer phone# using format area-xxx-xxxx");

// perform SQL INSERT statement to add a new row to the

// CUSTOMERS table using the values set in the previous

// step - the executeUpdate() method of the Statement

// object is used to perform the INSERT

myStatement.executeUpdate(

"INSERT INTO customers " +

"(customer\_id, first\_name, last\_name, dob, phone) VALUES (" +

customerId + ", '" + firstName + "', '" + lastName + "', " +

"TO\_DATE('" + dob + "', 'YYYY, MM, DD'), '" + phone + "')"

);

System.out.println("Added row to customers table");

// perform SQL UPDATE statement to modify the first\_name

// column of customer #1

lastName = "Cat";

myStatement.executeUpdate(

"UPDATE customers " +

"SET last\_name = '" + lastName + "' " +

"WHERE customer\_id = 1"

);

System.out.println("Updated row in customers table");

// perform SQL DELETE statement to remove customer #5

myStatement.executeUpdate(

"DELETE FROM customers " +

"WHERE customer\_id = 5"

);

System.out.println("Deleted row from customers table");

// create a ResultSet object, and populate it with the

// result of a SELECT statement that retrieves the

// customer\_id, first\_name, last\_name, dob, and phone columns

// for all the rows from the customers table - the

// executeQuery() method of the Statement object is used

// to perform the SELECT

// create a result set object, give it a name

ResultSet customerResultSet = myStatement.executeQuery(

"SELECT customer\_id, first\_name, last\_name, dob, phone " +

"FROM customers"

);

// you can count resulted rows and add that info to the message

System.out.println("Retrieved rows from customers table");

// loop through the rows in the ResultSet object using the

// next() method, and use the get methods to read the values

// retrieved from the database columns

while (customerResultSet.next()) {

// would be null if we reach the end; this is how we can figure out if data entered is invalid as well

// extract the data using real column names into local variables I defined in the program

customerId = customerResultSet.getInt("customer\_id");

firstName = customerResultSet.getString("first\_name");

lastName = customerResultSet.getString("last\_name");

dob = customerResultSet.getDate("dob");

dobTime = customerResultSet.getTime("dob");

dobTimestamp = customerResultSet.getTimestamp("dob");

phone = customerResultSet.getString("phone");

System.out.println("customerId = " + customerId);

System.out.println("firstName = " + firstName);

System.out.println("lastName = " + lastName);

System.out.println("dob = " + dob);

System.out.println("dobTime = " + dobTime);

System.out.println("dobTimestamp = " + dobTimestamp);

System.out.println("phone = " + phone);

} // end of while loop

// close the ResultSet object using the close() method

customerResultSet.close();

// rollback the changes made to the database

myConnection.rollback();

// create numeric variables to store the product\_id and price columns

short productIdShort;

int productIdInt;

long productIdLong;

float priceFloat;

double priceDouble;

java.math.BigDecimal priceBigDec;

// create another ResultSet object and retrieve the

// product\_id, product\_type\_id, and price columns for a specific product

myStatement.executeUpdate(

"INSERT INTO products " +

"(product\_id, product\_type\_id, name, description) VALUES (

25 + 4 + 'Beans' + 'Awesome beans' )

);

System.out.println("Added row to customers table");

ResultSet productResultSet = myStatement.executeQuery(

"SELECT product\_id, product\_type\_id, price " +

"FROM products " //+

//"WHERE product\_id = 3"

);

System.out.println("Retrieved row from products table");

while (productResultSet.next()) {

System.out.println("product\_id = " +

productResultSet.getInt("product\_id"));

System.out.println("product\_type\_id = " +

productResultSet.getInt("product\_type\_id"));

// check if the value just read by the get method was NULL

if (productResultSet.wasNull()) {

System.out.println("Last value read was NULL");

}

productIdInt = productResultSet.getInt("product\_id");

priceDouble = productResultSet.getDouble("price");

System.out.println("productIdInt = " + productIdInt);

System.out.println("priceDouble = " + priceDouble);

} // end of while loop

// close the ResultSet object

productResultSet.close();

// perform a SQL DDL CREATE TABLE statement to create a new table

// that may be used to store customer addresses

myStatement.execute(

"CREATE TABLE addresses (" +

" address\_id INTEGER CONSTRAINT addresses\_pk PRIMARY KEY," +

" customer\_id INTEGER CONSTRAINT addresses\_fk\_customers " +

" REFERENCES customers(customer\_id)," +

" street VARCHAR2(20) NOT NULL," +

" city VARCHAR2(20) NOT NULL," +

" state CHAR(2) NOT NULL" +

")"

);

System.out.println("Created addresses table");

// drop this table using the SQL DDL DROP TABLE statement

myStatement.execute("DROP TABLE addresses");

System.out.println("Dropped addresses table");

} catch (SQLException e) {

System.out.println("Error code = " + e.getErrorCode());

System.out.println("Error message = " + e.getMessage());

System.out.println("SQL state = " + e.getSQLState());

e.printStackTrace();

} finally {

try {

// close the Statement object using the close() method

if (myStatement != null) {

myStatement.close();

}

// close the Connection object using the close() method

if (myConnection != null) {

myConnection.close();

}

} catch (SQLException e) {

e.printStackTrace();

}

}

} // end of main()

}