| **DATA 504 Project 2**  **Using PL/SQL and JDBC to**  **Implement the Retail Business Management System**    The honesty statement: “We have done this assignment completely on our own. We have not copied it, nor have we given our solution to anyone else. We understand that if we are involved in plagiarism or cheating we will have to sign an official form that we have cheated and that this form will be stored in our official university records. We also understand that we will receive a grade of 0 for the involved assignment and our grades will be reduced by one level (e.g., from A to A- or from B+ to B) for our first offense, and that we will receive a grade of “F” for the course for any additional offense of any kind.”   * **Priyanka Lalge, Shruti Shukla & Amit Kumar Srivastava** |
| --- |

## Introduction

The project 2 for DATA-504 requires us to write a couple of sequences and triggers along with a package containing some procedures and a function within it. Also, a jdbc code was written to give us a flavor of the application layer connectivity with the database.

## Purpose of this document

The aim of this document is to introduce the pl/sql objects and compilation steps related to the PL/SQL. In addition, a jdbc code is introduced to showcase how the application/business layer interacts with the database.

## PL/SQL

The details of each pl/sql object created are provided below. In addition, code and installation steps are provided for each object separately after the description of the objects involved.

### Sequences:

#### seqpur#

This is the sequence object created for the attribute pur# of **purchases** table in order to automatically generate unique values when new records are inserted into the **purchases** table. The sequence starts with 100001 and increases by 1. This sequence is used when a new purchase has been made by a customer and add\_purchase procedure is executed in the backend.

**Usage:**

<sequence\_name>.nextval

**Example:**

SQL> seqpur#.nextval

#### seqlog#

This is the sequence object created for the attribute *log#* of **logs** table in order to automatically generate unique values when new records are inserted into the **logs** table. The sequence starts with 1001 and increases by 1. This sequence is used when the triggers are executed in the backend.

**Usage:**

<sequence\_name>.nextval

**Example:**

SQL> seqlog#.nextval

### Triggers:

#### purchases\_add

This trigger is created to update the **qoh** value in **products** table, **visits\_made** and **last\_visit\_date** in **customers** table, and display relevant messages during the trigger execution, after a tuple is added to purchases table.

After adding a tuple to the Purchases table, the trigger is generated and the qoh of the product is reduced by the quantity purchased. If the purchase causes qoh to be below qoh\_threshold, then qoh is updated to a value equivalent to qoh\_threshold + 10. Also, visits\_made of the involved customer is increased by 1 if the purchase is made on a new date and last\_visit\_date is updated accordingly.

#### insert\_customers

This trigger is created to insert an audit entry in the **logs** table. When a tuple is added to the **customers** table due to the first visit of a customer, the trigger is generated. The *table\_name* attribute of the logs table is updated as “customers” signifying a change in the **customers** table, the *operation* attribute is updated as “insert” signifying the addition of a new customer and the *tuple\_pkey* attribute is updated as “cid” of the newly inserted customer.

#### update\_lastvisitdate\_customers

This trigger is created to insert an audit entry in the **logs** table. A tuple is added to the **logs** table due to update of the *last\_visit\_date* attribute of the **customers** table. The *table\_name* attribute of the logs table is updated as “customers” signifying a change in the **customers** table, the *operation* attribute is updated as “update” signifying the recent visit date of a customer and the *tuple\_pkey* attribute is updated as “cid” of the affected customer.

#### update\_visitsmade\_customers

This trigger is created to insert an audit entry in the **logs** table. A tuple is added to the **logs** table due to update of the *visits\_made* attribute of the **customers** table. The *table\_name* attribute of the logs table is updated as “customers” signifying a change in the **customers** table, the *operation* attribute is updated as “update” signifying the updated number of visits made by a customer and the *tuple\_pkey* attribute is updated as “cid” of the affected customer.

#### insert\_purchases

This trigger is created to insert an audit entry in the **logs** table. A tuple is added to the **logs** table when a new purchase tuple has been added in the **purchases** table. The *table\_name* attribute of the **logs** table is updated as “purchases” signifying a new purchase made by a customer, the *operation* attribute is updated as “insert” signifying the insertion of a new tuple and the *tuple\_pkey* attribute is updated as “pur#” of the newly inserted purchase.

#### update\_products

This trigger is created to insert an audit entry in the **logs** table. A tuple is added to the **logs** table when the *qoh* attribute has been updated with a new value in the **products** table. The *table\_name* attribute of the **logs** table is updated as “products” signifying an update of the **products** table, the *operation* attribute is updated as “update” signifying an update in the **products** table and the *tuple\_pkey* attribute is updated as “pid” of the affected product.

### Package:

#### RetailBusiness\_Package

The package consists of two parts namely package specification and package body. In the specification we declare the cursors, procedures and functions that can be referenced from outside the package. The body contains the definitions for the objects mentioned in the specification. This package has been created with the below listed procedures and function to perform necessary activities according to the requirements document.

### Procedures:

#### show\_customers

This procedure has no input parameters and displays all the tuples from the **customers** table whenever it is called or executed from the package.

**Usage:**

execute <package\_name>.<procedure\_name>;

**Example:**

SQL> execute RetailBusiness\_Package.show\_customers;

#### show\_employees

This procedure has no input parameters and displays all the tuples from the **employees** table whenever it is called or executed from the package.

**Usage:**

execute <package\_name>.<procedure\_name>;

**Example:**

SQL> execute RetailBusiness\_Package.show\_employees;

#### show\_products

This procedure has no input parameters and displays all the tuples from the **products** table whenever it is called or executed from the package.

**Usage:**

execute <package\_name>.<procedure\_name>;

**Example:**

SQL> execute RetailBusiness\_Package.show\_products;

#### show\_purchases

This procedure has no input parameters and displays all the tuples from the **purchases** table whenever it is called or executed from the package.

**Usage:**

execute <package\_name>.<procedure\_name>;

**Example:**

SQL> execute RetailBusiness\_Package.show\_purchases;

#### show\_logs

This procedure has no input parameters and displays all the tuples from the **logs** table whenever it is called or executed from the package.

**Usage:**

execute <package\_name>.<procedure\_name>;

**Example:**

SQL> execute RetailBusiness\_Package.show\_logs;

#### purchases\_made

Objective of this procedure is to return the name of the customer as well as every purchase (output pid, pur\_date, qty, unit\_price, and total) the customer has made. This procedure has to be called by providing the cid of the customer as the input parameter.

**Usage:**

execute <package\_name>.<procedure\_name>(<c\_id>);

**Example:**

SQL> execute RetailBusiness\_Package.purchases\_made('c001');

#### add\_customer

Objective of this procedure is to add a tuple into the **customers** table. This procedure has to be called by providing the cid, name and telephone# of the new customer as the input parameter. For the attributes *visits\_made* and *last\_visit\_date* of any newly added customer, a default value of 1 and sysdate is considered respectively.

**Usage:**

execute <package\_name>.<procedure\_name>(<c\_id>,<c\_name>,<c\_telephone#>);

**Example:**

SQL> execute RetailBusiness\_Package.add\_customer('c009','Mr. Meng','777-888-9900');

#### add\_purchase

Objective of this procedure is to add a tuple into the **purchases** table. This procedure has to be called by providing the eid, pid, cid, qty and unit\_price of the new purchase as the input parameter. The attribute *pur\_#* is generated by the sequence. The attribute *pur\_date* is considered by the sysdate. The attributes *total* and *saving* are calculated on the go by the procedure.

**Usage:**

execute <package\_name>.<procedure\_name>(<e\_id>, <p\_id>, <c\_id>, <pur\_qty>, <pur\_unit\_price>);

**Example:**

SQL> execute RetailBusiness\_Package.add\_purchase('e01', 'p004', 'c007', 4, 0.6);

### Function:

#### number\_customers

Objective of this function is to report the number of customers who have purchased the product identified by the pid. This function has to be called by providing the pid of the product as the input parameter.

**Usage:**

declare

result number;

begin

-- Call the function

result := package\_name.function\_name(<pid>);

DBMS\_OUTPUT.PUT\_LINE(result);

End;/

OR

select package\_name.function\_name(<pid>) from dual;

**Example:**

SQL> select RetailBusiness\_Package.number\_customers('p009') from dual;

SQL> declare

result number;

begin

-- Call the function

result := RetailBusiness\_Package.number\_customers('p009');

DBMS\_OUTPUT.PUT\_LINE(result);

End;/

## Compilation steps:

After the creation of the required tables below, the following steps should be performed:

* ​​Employees(eid, name, telephone#, email)
* Customers(cid, name, telephone#, visits\_made, last\_visit\_date)
* Products(pid, name, qoh, qoh\_threshold, regular\_price, discnt\_rate)
* Purchases(pur#, eid, pid, cid, pur\_date, qty, unit\_price, total, saving)
* Logs(log#, user\_name, operation, op\_time, table\_name, tuple\_pkey)

Login to the database with a username/password in the sqlplus environment or any compatible sql client connected to the oracle database.

### Step 1: Compile sequences

For a clean compilation we first compile the sequences. In order to compile the sequences the below code snippet should be executed.

*/\* sequence for pur# \*/*

CREATE SEQUENCE seqpur#

INCREMENT BY 1

START WITH 100001

ORDER;

*/\* sequence for log# \*/*

CREATE SEQUENCE seqlog#

INCREMENT BY 1

START WITH 1001

ORDER;

### Step 2: Compile triggers

The second step involves the compilation of the triggers. The below set of codes should be executed in order to compile the necessary triggers as per the requirements document.

SET SERVEROUTPUT ON

*/\* question 6: trigger for inserting a tuple into the Purchases table \*/*

create or replace trigger purchases\_add

after insert on purchases

for each row

declare

qty products.qoh%type;

threshold products.qoh\_threshold%type;

today customers.last\_visit\_date%type;

qoh\_updated products.qoh%type;

begin

update products

set qoh = qoh - :new.qty where pid = :new.pid;

select last\_visit\_date into today from customers where cid = :new.cid;

if (to\_char(today, 'YYYY') = to\_char(sysdate, 'YYYY') and to\_char(today, 'MON') = to\_char(sysdate, 'MON') and to\_char(today, 'DD') = to\_char(sysdate, 'DD'))

then

NULL;

else

update customers

set visits\_made = visits\_made + 1 where cid = :new.cid;

update customers

set last\_visit\_date = sysdate where cid = :new.cid;

end if;

select qoh, qoh\_threshold into qty, threshold from products where pid = :new.pid;

if(qty < threshold) then

dbms\_output.put\_line('The current qoh of the product is below the required threshold and new supply is required');

update products

set qoh = threshold + 10 where pid = :new.pid;

select qoh into qoh\_updated from products where pid= :new.pid;

dbms\_output.put\_line('The new value of qoh after supply is: ' || qoh\_updated );

end if;

end;

/

*/\* question 7.1: trigger for inserting a tuple into the Customers table \*/*

create or replace trigger insert\_customers

after insert on customers

for each row

begin

insert into logs values (seqlog#.nextval, user, 'insert', sysdate, 'customers', :new.cid);

end;

/

show errors

*/\* question 7.2 trigger for updating the last\_visit\_date attribute of the Customers table \*/*

create or replace trigger update\_lastvisitdate\_customers

after update of last\_visit\_date on customers

for each row

begin

insert into logs values (seqlog#.nextval, user, 'update', sysdate, 'customers', :old.cid);

end;

/

show errors

*/\* question 7.3: trigger for updating the visits\_made attribute of the Customers table \*/*

create or replace trigger update\_visitsmade\_customers

after update of visits\_made on customers

for each row

begin

insert into logs values (seqlog#.nextval, user, 'update', sysdate, 'customers', :old.cid);

end;

/

show errors

*/\* question 7.4: trigger for inserting a tuple into the Purchases table \*/*

create or replace trigger insert\_purchases

after insert on purchases

for each row

begin

insert into logs values (seqlog#.nextval, user, 'insert', sysdate, 'purchases', :new.pur#);

end;

/

show errors

*/\* question 7.5: trigger for updating the qoh attribute of the Products table \*/*

create or replace trigger update\_products

after update of qoh on products

for each row

begin

insert into logs values (seqlog#.nextval, user, 'update', sysdate, 'products', :old.pid);

end;

/

show errors

### Step 3: Compile package

The third step involves the compilation of the package which includes the subprograms (procedures and functions). The below set of codes should be executed in order to compile the package as per the requirements document.

SET SERVEROUTPUT ON

*/\* package name specification: RetailBusiness\_Package \*/*

CREATE OR REPLACE PACKAGE RetailBusiness\_Package AS

type cursor1 is ref cursor;

*/\* question 2: procedures to show the tuples in each table \*/*

procedure show\_customers;

procedure show\_employees;

procedure show\_logs;

procedure show\_products;

procedure show\_purchases;

*/\* question 3: procedure to return the name of the customer as well as every purchase \*/*

procedure purchases\_made(c\_id in customers.cid%type);

*/\* question 4: function to report the number of customers who have purchased the product identified by the pid \*/*

function number\_customers(cust\_pid IN purchases.pid%type) return NUMBER;

*/\* question 5: procedure for adding tuples to the Customers table.\*/*

procedure add\_customer(c\_id in customers.cid%type, c\_name in customers.name%type, c\_telephone# in customers.telephone#%type);

*/\* question 6: procedure for adding tuples to the Purchases table.\*/*

procedure add\_purchase(e\_id in purchases.eid%type, p\_id in purchases.pid%type, c\_id in purchases.cid%type, pur\_qty in purchases.qty%type, pur\_unit\_price in purchases.unit\_price%type);

end;

/

*/\* package body \*/*

create or replace package body RetailBusiness\_Package as

*/\* question 2: procedure to show the tuples in employees \*/*

PROCEDURE show\_employees

AS

CURSOR c\_emp IS

SELECT \* FROM employees;

c\_emp\_rec c\_emp%rowtype;

BEGIN

dbms\_output.put\_line( 'EID' || ',' || 'NAME' || ',' || 'TELEPHONE#' || ',' || 'EMAIL');

OPEN c\_emp;

LOOP

FETCH c\_emp INTO c\_emp\_rec;

EXIT WHEN c\_emp%notfound;

dbms\_output.put\_line(c\_emp\_rec.eid || ',' || c\_emp\_rec.name || ',' || c\_emp\_rec.telephone# || ',' ||c\_emp\_rec.email);

END LOOP;

CLOSE c\_emp;

END show\_employees;

*/\* question 2: procedure to show the tuples in customers \*/*

PROCEDURE show\_customers

AS

CURSOR c\_cust IS

SELECT \* FROM customers;

c\_cust\_rec c\_cust%rowtype;

BEGIN

dbms\_output.put\_line('CID' || ',' || 'NAME' || ',' || 'TELEPHONE#' || ',' ||'VISITS\_MADE' || ',' || 'LAST\_VISIT\_DATE');

OPEN c\_cust;

LOOP

FETCH c\_cust INTO c\_cust\_rec;

EXIT WHEN c\_cust%notfound;

dbms\_output.put\_line(c\_cust\_rec.cid || ',' || c\_cust\_rec.name || ',' || c\_cust\_rec.telephone# || ',' ||c\_cust\_rec.visits\_made || ',' || c\_cust\_rec.last\_visit\_date);

END LOOP;

CLOSE c\_cust;

END show\_customers;

*/\* question 2: procedure to show the tuples in products \*/*

PROCEDURE show\_products

AS

CURSOR c\_pr IS

SELECT \* FROM products;

c\_pr\_rec c\_pr%rowtype;

BEGIN

dbms\_output.put\_line('PID' || ',' || 'NAME' || ',' || 'QOH' || ',' || 'QOH\_THRESHOLD' || ',' || 'REGULAR\_PRICE' || ',' || 'DISCNT\_RATE');

OPEN c\_pr;

LOOP

FETCH c\_pr INTO c\_pr\_rec;

EXIT WHEN c\_pr%notfound;

dbms\_output.put\_line(c\_pr\_rec.pid || ',' || c\_pr\_rec.name || ',' || c\_pr\_rec.qoh || ',' ||c\_pr\_rec.qoh\_threshold || ',' || c\_pr\_rec.regular\_price || ',' || c\_pr\_rec.discnt\_rate);

END LOOP;

CLOSE c\_pr;

END show\_products;

*/\* question 2: procedure to show the tuples in purchases \*/*

PROCEDURE show\_purchases

AS

CURSOR c\_pur IS

SELECT \* FROM purchases;

c\_pur\_rec c\_pur%rowtype;

BEGIN

dbms\_output.put\_line('PUR#' || ',' || 'EID' || ',' || 'PID' || ',' || 'CID' || ',' || 'PUR\_DATE' || ',' || 'QTY' || ',' || 'UNIT\_PRICE' || ',' || 'TOTAL' ||','|| 'SAVING');

OPEN c\_pur;

LOOP

FETCH c\_pur INTO c\_pur\_rec;

EXIT WHEN c\_pur%notfound;

dbms\_output.put\_line(c\_pur\_rec.pur# || ',' || c\_pur\_rec.eid || ',' || c\_pur\_rec.pid || ',' ||c\_pur\_rec.cid || ',' || c\_pur\_rec.pur\_date || ',' || c\_pur\_rec.qty || ',' ||

c\_pur\_rec.unit\_price || ',' || c\_pur\_rec.total ||','|| c\_pur\_rec.saving);

END LOOP;

CLOSE c\_pur;

END show\_purchases;

*/\* question 2: procedure to show the tuples in logs \*/*

PROCEDURE show\_logs

AS

CURSOR c\_log IS

SELECT \* FROM logs;

c\_log\_rec c\_log%rowtype;

BEGIN

dbms\_output.put\_line('LOG#' || ',' || 'USER\_NAME' || ',' || 'OPERATION' || ',' || 'OP\_TIME' || ',' || 'TABLE\_NAME' || ',' || 'TUPLE\_PKEY');

OPEN c\_log;

LOOP

FETCH c\_log INTO c\_log\_rec;

EXIT WHEN c\_log%notfound;

dbms\_output.put\_line(c\_log\_rec.log# || ',' || c\_log\_rec.user\_name || ',' || c\_log\_rec.operation || ',' ||c\_log\_rec.op\_time || ',' || c\_log\_rec.table\_name || ',' || c\_log\_rec.tuple\_pkey);

END LOOP;

CLOSE c\_log;

END show\_logs;

*/\* question 3,: procedure to return the name of the customer as well as every purchase \*/*

PROCEDURE purchases\_made(c\_id in customers.cid%type)

IS

cursor cursor1 is

select \* from

(select name from customers where cid = c\_id) a,

(select \* from

(select pid, pur\_date, qty, unit\_price, total from purchases where cid = c\_id)) b;

c\_cid NUMBER;

invalid\_cid EXCEPTION;

BEGIN

c\_cid:= 0;

SELECT count(cid) INTO c\_cid FROM CUSTOMERS WHERE cid = c\_id;

IF(c\_cid = 0) THEN

RAISE invalid\_cid;

ELSE

dbms\_output.put\_line( 'name' || ',' || 'pid' || ',' || 'pur\_date' || ',' || 'qty' || ',' || 'unit\_price' || ',' || 'total');

for c1 in cursor1

loop

dbms\_output.put\_line( c1.name || ',' || c1.pid || ',' || c1.pur\_date || ',' || c1.qty || ',' || c1.unit\_price || ',' || c1.total );

end loop;

END IF;

EXCEPTION

WHEN invalid\_cid THEN

dbms\_output.put\_line('Invalid Customer ID');

END purchases\_made;

*/\* question 4: function to report the number of customers who have purchased the product identified by the pid \*/*

FUNCTION number\_customers(cust\_pid IN purchases.pid%type)

RETURN NUMBER IS

num\_of\_customers NUMBER;

c\_pid NUMBER;

invalid\_pid EXCEPTION;

BEGIN

c\_pid:= 0;

SELECT count(pid) INTO c\_pid FROM PURCHASES WHERE pid = cust\_pid;

IF(c\_pid = 0) THEN

RAISE invalid\_pid;

ELSE

SELECT count(DISTINCT cid) into num\_of\_customers FROM purchases WHERE pid = cust\_pid;

return num\_of\_customers;

END IF;

EXCEPTION

WHEN invalid\_pid THEN

dbms\_output.put\_line('Invalid purchase ID.');

RETURN 0;

END;

*/\* question 5: procedure for adding tuples to the Customers table\*/*

PROCEDURE add\_customer(c\_id IN CUSTOMERS.CID%TYPE, c\_name IN CUSTOMERS.NAME%TYPE, c\_telephone# IN CUSTOMERS.TELEPHONE#%TYPE)

IS

BEGIN

INSERT INTO CUSTOMERS VALUES (c\_id, c\_name, c\_telephone#, 1, sysdate);

commit;

END;

*/\* question 6: procedure for adding tuples to the Purchases table \*/*

PROCEDURE add\_purchase(

e\_id in purchases.eid%type,

p\_id in purchases.pid%type,

c\_id in purchases.cid%type,

pur\_qty in purchases.qty%type,

pur\_unit\_price in purchases.unit\_price%type

)

IS

pur\_total PURCHASES.TOTAL%TYPE;

pur\_saving PURCHASES.SAVING%TYPE;

reg\_price PRODUCTS.REGULAR\_PRICE%TYPE;

pr\_qoh PRODUCTS.QOH%TYPE;

insuf\_qty exception;

BEGIN

pur\_total := pur\_qty \* pur\_unit\_price;

SELECT REGULAR\_PRICE INTO reg\_price FROM PRODUCTS WHERE PID = p\_id;

pur\_saving := (reg\_price \* pur\_qty) - (pur\_qty \* pur\_unit\_price);

SELECT QOH INTO pr\_qoh FROM PRODUCTS WHERE PID = p\_id;

IF(pur\_qty <= pr\_qoh) THEN

*/\*Insert new purchase\*/*

INSERT INTO PURCHASES VALUES (seqpur#.nextval, e\_id, p\_id, c\_id, sysdate, pur\_qty, pur\_unit\_price, pur\_total, pur\_saving);

commit;

ELSE

RAISE insuf\_qty;

END IF;

EXCEPTION

WHEN insuf\_qty THEN

dbms\_output.put\_line('Insufficient quantity in stock.');

END add\_purchase;

END;

### Alternative method: Compile objects via .sql files

1. The alternative of the above steps is to save different objects as different .sql files in the current worksheet of oracle. In our case, we would require 3 files.
   1. Save the sequence code of step 1 as “retailBusiness\_sequence.sql”.
   2. Save the trigger code of step 2 as “retailBusiness\_triggers.sql”.
   3. Save the package code of step 3 as “retailBusiness\_package.sql”.
2. Once the .sql files are saved, the below set of codes should be executed from the worksheet in order to compile the PL/SQL objects.

start <Sequences\_file\_name>

start <Triggers\_file\_name>

start <Package\_file\_name>

#### Note:

#### All the above mentioned objects should be created after the creation of required tables.

#### As per the recommendation we shall compile the objects as: Sequences → Triggers → Package.

#### Whenever we drop any table and recreate, we should recreate the corresponding triggers and sequences which were created on it.

## Java Database Connectivity (JDBC)

Java Database Connectivity (JDBC) is an application programming interface (API) for the programming language Java, which defines how a client may access a database. The Java Database Connectivity (JDBC) API provides universal data access from the Java programming language.

Below the jdbc coding is provided which has two-fold purpose as per the requirements document.

1. Retrieve all tuples of the Customers table: This is achieved by the *getAllCust()* method defined in the jdbc. No input parameters are required.
2. Retrieve the tuple of any given “cid” from the Customers table: This is achieved by the *getOneCust()* method defined in the jdbc in which the “cid” of the customer is passed as an input by the end user through the keyboard.

The JDBC requires it to be deployed in conjunction with the application coding logic in the middleware, however, we don't have an application for the jdbc in question. Hence here we skip the installation process. For our purpose, we will directly execute the jdbc file. The connection string defined in the code helps to authenticate the credentials and connect to the database. Once the connection is established, the code then performs the above defined operations as per the requirements document.

### Java code:

import **java**.**sql**.\*;

import **java**.**io**.\*;

import **oracle**.**jdbc**.**pool**.**OracleDataSource**;

public class **getCust** {

public static void **main**(**String** args[]) throws **SQLException** {

*// Method for Part1 of JDBC - get all customers*

**getAllCust**();

*// Wait for user for Part2*

**System**.out.**println**();

**System**.out.**println**("Press enter to continue...");try{ **System**.in.**read**();}catch(**Exception** e){ e.**printStackTrace**();}

*// Method for Part2 of JDBC - get one customer for a cid*

**getOneCust**();

}

private static void **getAllCust**() {

try {

*// Connection to Oracle server.*

**OracleDataSource** ds = new **oracle**.**jdbc**.**pool**.**OracleDataSource**();

ds.**setURL**("jdbc:oracle:thin:@castor.cc.binghamton.edu:1521:acad111");

**Connection** conn = ds.**getConnection**("asrivas3", "sr927641");

*// Query*

**Statement** stmt = conn.**createStatement**();

*// Save result*

**ResultSet** rset;

rset = stmt.**executeQuery**("SELECT \* FROM customers");

**System**.out.**println**("CID, Name, Telephone#, Visits Made, Last Visit Date ");

*// Print*

while (rset.**next**()) {

**System**.out.**print**(rset.**getString**(1) + ", ");

**System**.out.**print**(rset.**getString**(2) + ", ");

**System**.out.**print**(rset.**getString**(3) + ", ");

**System**.out.**print**(rset.**getString**(4) + ", ");

**System**.out.**println**(rset.**getString**(5) + ", ");

}

*// close the result set, statement, and the connection*

rset.**close**();

stmt.**close**();

conn.**close**();

} catch (**SQLException** ex) {

**System**.out.**println**(

"\n\*\*\* SQLException caught 1st\*\*\*\n" + ex.**getStackTrace**() + ex.**getMessage**()

+ ex.**getLocalizedMessage**());

} catch (**Exception** e) {

**System**.out.**println**(

"\n\*\*\* other Exception caught 2nd\*\*\*\n" + e.**getStackTrace**() + e.**getMessage**()

+ e.**getLocalizedMessage**());

}

}

private static void **getOneCust**() {

try {

*// Connection to Oracle server.*

**OracleDataSource** ds = new **oracle**.**jdbc**.**pool**.**OracleDataSource**();

ds.**setURL**("jdbc:oracle:thin:@castor.cc.binghamton.edu:1521:acad111");

**Connection** conn = ds.**getConnection**("asrivas3", "sr927641");

*// Input CID from keyboard*

**BufferedReader** readKeyBoard;

**String** cid;

readKeyBoard = new **BufferedReader**(new **InputStreamReader**(**System**.in));

**System**.out.**print**("Please enter customer's CID:");

cid = readKeyBoard.**readLine**();

*// Prepare statement and save in resultset*

**PreparedStatement** select1 = conn.**prepareStatement**("SELECT \* FROM customers where cid = '" + cid + "'");

**ResultSet** rset;

select1.**executeUpdate**();

rset = select1.**executeQuery**();

*// Check if CID exists or not and then display desired output*

if (!rset.**isBeforeFirst**()) {

**System**.out.**println**("Customer doesn't exist");

} else {

**System**.out.**println**("CID, Name, Telephone#, Visits Made, Last Visit Date ");

while (rset.**next**()) {

**System**.out.**print**(rset.**getString**(1) + ", ");

**System**.out.**print**(rset.**getString**(2) + ", ");

**System**.out.**print**(rset.**getString**(3) + ", ");

**System**.out.**print**(rset.**getString**(4) + ", ");

**System**.out.**println**(rset.**getString**(5) + ", ");

}

}

*// close the resultset, statement, and the connection*

rset.**close**();

select1.**close**();

conn.**close**();

} catch (**SQLException** ex) {

**System**.out.**println**("\n\*\*\* SQLException caught 1st\*\*\*\n" + ex.**getStackTrace**() + ex.**getMessage**()

+ ex.**getLocalizedMessage**());

} catch (**Exception** e) {

**System**.out.**println**("\n\*\*\* other Exception caught 2nd\*\*\*\n" + e.**getStackTrace**() + e.**getMessage**()

+ e.**getLocalizedMessage**());

}

}

}