Retail Business Management System

Documentation

(Using Oracle's PL/SQL and JDBC)

Course: Database Systems

Project Team Members: Chetas Mehta (cmehta1@binghamton.edu)

J Divya Nuti (jnuti1@binghamton.edu)

Sequences:

When a sequence number is generated, the sequence is incremented, independent of the transaction committing or rolling back. If two users concurrently increment the same sequence, then the sequence numbers each user acquires may have gaps, because sequence numbers are being generated by the other user. One user can never acquire the sequence number generated by another user. After a sequence value is generated by one user, that user can continue to access that value regardless of whether the sequence is incremented by another user.

Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables. It is possible that individual sequence numbers will appear to be skipped, because they were generated and used in a transaction that ultimately rolled back. Additionally, a single user may not realize that other users are drawing from the same sequence.

/*Here we have created sequence pur_# to get value as purchase id starting from 100000 and incremented by 1 every time we need to enter value of it into the table*/

create sequence pur_# minvalue 100000 maxvalue 999999 start with 100000 increment by 1;

/*Here we have created sequence sup_# to get value as supplier id starting from 1000 and incremented by 1 every time we need to enter value of it into the table*/

create sequence sup_# minvalue 1000 maxvalue 9999 start with 1000 increment by 1;

/*Here we have created sequence pur_# to get value as log id starting from 10000 and incremented by 1 every time we need to enter value of it into the table*/

create sequence log_# minvalue 10000 maxvalue 99999 start with 10000 increment by 1;

To access any of the sequences **.nextval** is used.

Functions:

A **stored function** (also called a **user function** or **user-defined function**) is a set of PL/SQL statements you can call by name. Stored functions are very similar to procedures, except that a function returns a value to the environment in which it is called. User functions can be used as part of a SQL expression.

function show_emp return emp;

This function will show the content of employees table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from employees;

Exceptions are handled here too.

function show_cust return cust;

This function will show the content of customers table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from customers;

Exceptions are handled here too.

function show_prod return prod;

This function will show the content of products table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from products;

Exceptions are handled here too.

function show_sup return sup;

This function will show the content of suppliers table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from suppliers;

Exceptions are handled here too.

function show_pur return pur;

This function will show the content of purchases table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from purchases;

Exceptions are handled here too.

function show_logs return log;

This function will show the content of logs table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from logs;

Exceptions are handled too.

function show_supply return supply;

This function will show the content of supply table. To display content, this function will be called. This function returns the cursor created to hold the value.

Query used in: select * from supply;

Exceptions are handled here too.

function report_monthly_sale(prod_id in products.pid%type)return sales_cursor;

This is function to display monthly report for any of the products.

Input is pid of that product. And output is important information like name, the month (the first three letters of the month, e.g., FEB for February), the year (4 digits), the total quantity sold each month, the total dollar amount sold each month, and the average sale price (the total dollar amount divided by the total quantity) of each month.

Procedures:

procedure add_purchases(e_id in purchases.eid%type,p_id in purchases.pid%type,c_id in purchases.cid%type,p_qty in purchases.qty%type);

This procedure will add a tuple in purchase table. It will take input from the user and add that value (insert) into the purchase table. The pur# of any newly added purchase should be automatically generated by your sequence. And total_price should be computed based on the data in the database automatically and ptime would be the current date.

procedure add_products(p_id in products.pid%type,p_name in products.pname%type,qoh_1 in products.qoh%type, qoh_threshold1 in products.qoh_threshold%type,orig in products.original_price%type,disc in products.discnt_rate%type);

This procedure will add a tuple in products table. It will take input from the user and add that value (insert) into the product table.

Triggers:

Adding a tuple to the logs table automatically whenever following tables are modified. Triggers are created to accomplish this.

create or replace trigger InsertPurchase

When a tuple is added to the logs table due to this first event, the table_name should be "purchases", the operation should be "insert" and the key_value should be the pur# of the newly inserted purchase.

create or replace trigger UpdateProducts

When a tuple is added to the logs table due to this second event, the table_name should be "products", the operation should be "update" and the key_value should be the pid of the affected product.

create or replace trigger UpdateCustomers

When a tuple is added to the logs table due to the third event, the table_name should be "customers", the operation should be "update" and the key_value should be the cid of the affected customer.

create or replace trigger InsertSupply

When a tuple is added to the logs table due to the fourth event, the table_name should be "supply", the operation should be "insert" and the key_value should be the sup# of the newly inserted supply.

Before a purchase is actually made (i.e., before a tuple is added into the purchases table), program needs to make sure that, for the involved product, the quantity to be purchased is equal to or smaller than the quantity on hand (qoh). Otherwise, an appropriate message should be displayed “Insufficient quantity in stock.” and the purchase request should be rejected. To accomplish this, we have created following trigger.
create or replace trigger check_qoh

After adding a tuple to the purchases table, the qoh column of the products table should be modified accordingly, that is, the qoh of the product involved in the purchase should be reduced by the quantity purchased. If the purchase causes the qoh of the product to be below qoh_threshold, your program should perform the following tasks:
print a message saying that the current qoh of the product is below the required threshold and new supply is required,
automatically order supply for the product (i.e., add a new tuple to the Supply table): the sup# is generated by a sequence, the pid is the pid of the product involved in the purchase, the sid is the sid of a supplier who has supplied this product before (there should be such information in the current Supply table; if multiple suppliers have supplied this product before, use the smallest sid), the quantity to order is 10 + M, where M is the minimum value for quantity such that M + qoh > qoh_threshold, and use sysdate for sdate,
Increase qoh of the product by the quantity ordered, and
Print another message showing the new value of the qoh of the product. In addition, the insertion of the new tuple in the purchases table may cause the visits_made of the involved customer to be increased by one if the purchase is made on a new date and the last_visit_date may also have a new date. Use triggers to implement the update of qoh, the printing (displaying) of the messages and the updates of visits_made and last_visit_date
To accomplish this, we have created trigger:

create or replace trigger pur_trig

Interface:

We have created java interface to implement this project. It provides following menu options to interact with user:

1: Display employees

2: Display customers

3: Display products

4: Display purchases

5: Display suppliers

6: Display supply

7: Display logs

8: Display monthly sale information

9: insert into products

10: insert into purchases

We have created two different files here:

Display.java

MyPack.java

Project code:

[a] Que:2 & 3

```
create or replace package func_package as

type emp is ref cursor;

type cust is ref cursor;

type prod is ref cursor;

type sup is ref cursor;

type pur is ref cursor;

type supply is ref cursor;

type log is ref cursor;

type sales_cursor is ref cursor;

function show_emp return emp;

function show_cust return cust;

function show_prod return prod;

function show_sup return sup;

function show_pur return pur;

function show_logs return log;

function show_supply return supply;

function report_monthly_sale(prod_id in products.pid%type)return sales_cursor;

end;

/

show errors


-- procedure for show records in employees


create or replace package body func_package as

function show_emp

return emp as

showemp emp;
```

```
begin
open showemp for
select * from employees;
return showemp;
EXCEPTION
WHEN OTHERS THEN
dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);


end;


-- procedure for show records in products


function show_prod
return prod as
showprod prod;
begin
open showprod for
select * from products;
return showprod;
EXCEPTION
WHEN OTHERS THEN
dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);
end;


-- procedure for show records in customers


function show_cust
return cust as
```

```
showcust cust;

begin

open showcust for

select * from customers;

return showcust;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);


end;


-- procedure for show records in suppliers


function show_sup

return sup as

showsup sup;

begin

open showsup for

select * from suppliers;

return showsup;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);


end;


-- procedure for show records in purchases
```

```
function show_pur

return pur as

showpur pur;

begin

open showpur for

select * from purchases;

return showpur;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);

end;


-- procedure for show records in supply


function show_supply

return supply as

showsupply supply;

begin

open showsupply for

select * from supply;

return showsupply;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);

end;



-- procedure for show records in logs
```

```
function show_logs

return log as

showlog log;

begin

open showlog for

select * from logs;

return showlog;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);


end;



-- procedure for showing monthly reocrd for the product entered


function report_monthly_sale(prod_id in products.pid%type)

return sales_cursor as sc sales_cursor;

declare

c number;

begin

open sc for

select count(*) into c from products where pid=prod_id;

if(c=0) then

raise_application_error(-20001, 'exception');

end if;

select p.pname, to_char(pu.ptime,'mon') as month,to_char(pu.ptime,'yyyy') as year,
sum(pu.qty)as totalqty,
        sum(pu.total_price) as total_qty_price,

        avg(pu.total_price/qty) as avg_price

        from purchases pu, products p

        where p.pid = prod_id and p.pid = pu.pid

        group by to_char(pu.ptime,'mon'),to_char(pu.ptime,'yyyy'),p.pname ;

return sc;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line( 'SQL Error: ' || ' SQLCODE=' || SQLCODE || 'SQLERRM=' ||
SQLERRM);


end;


end;


/

show errors
```

Que:04

```
create or replace package insert_purchases as

procedure add_purchases(e_id in purchases.eid%type,p_id in purchases.pid%type,c_id in
purchases.cid%type,p_qty in purchases.qty%type);

procedure add_products(p_id in products.pid%type,p_name in
products.pname%type,qoh_1 in products.qoh%type, qoh_threshold1 in

products.qoh_threshold%type,orig in products.original_price%type,disc in
products.discnt_rate%type);

end;

/


--procedure for adding a tuple in purchases
```

```
create or replace package body insert_purchases as

procedure add_purchases(e_id in purchases.eid%type,p_id in purchases.pid%type,c_id in
purchases.cid%type,p_qty in purchases.qty%type)

is

total_price purchases.total_price%type;

discount_rate products.discnt_rate%type;

original_price products.original_price%type;

pricereduced products.original_price%type;

discnt_price products.original_price%type;

--variable declarations

begin

select prod.discnt_rate,prod.original_price into discount_rate,original_price

from products prod

where prod.pid=p_id;

pricereduced := original_price * discount_rate;

discnt_price := original_price - pricereduced;

total_price := p_qty*discnt_price;

insert into purchases values(pur_#.nextval,e_id,p_id,c_id,p_qty,sysdate,total_price);

end;


--procedure for adding tuple in products


procedure add_products(p_id in products.pid%type,p_name in
products.pname%type,qoh_1 in products.qoh%type, qoh_threshold1 in

products.qoh_threshold%type,orig in products.original_price%type,disc in
products.discnt_rate%type)

is

begin

insert into products values(p_id,p_name,qoh_1,qoh_threshold1,orig,disc);
```

```
end;

end;

/

show errors
```

Que:05

```
--trigger for inserting in purchases


set serveroutput on

create or replace trigger InsertPurchase

after insert on purchases

for each row

begin

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

end;

/

show errors


--trigger for updating products


set serveroutput on

create or replace trigger UpdateProducts

after update of qoh on products for each row

begin

insert into logs

values(log_#.nextval,'user',sysdate,'products','update',:new.pid);

end;
```

```
/
show errors

--trigger for updating customers

set serveroutput on
create or replace trigger UpdateCustomers
after update of visits_made on customers
for each row
begin
insert into logs values(log_#.nextval,'user',sysdate,'customers','update',:new.cid) ;
end;
/
show errors

--trigger for inserting in supply

set serveroutput on
create or replace trigger InsertSupply
after insert  on supply
for each row
begin
insert into logs values(log_#.nextval,'user',sysdate,'supply','insert',:new.sup#) ;
end;
/
show errors
```

Que:06

```
--trigger to check the qoh


create or replace trigger check_qoh
before insert or update on purchases
for each row
declare
quantity_exceeding exception;
prod_qoh products.qoh%type;
begin
select qoh into prod_qoh
from products prod
where prod.pid=:new.pid;
--checking if the quantity entered is less than threshold
if(:new.qty>=prod_qoh) then raise quantity_exceeding;
end if;
exception
when quantity_exceeding then
raise_application_error(-20001,'Insufficient quantity in stock');
end;
/
```

Que:07

```
--creating trigger on inserting in purchases


set serveroutput on
```

```
create or replace trigger pur_trig

after insert on purchases

for each row

declare

qoh_t products.qoh%type;

threshold_t products.qoh_threshold%type;

last_visit_date_t customers.last_visit_date%type;

smin supply.sid%type;

temp products.qoh%type;

M products.qoh%type;

quan supply.quantity%type;

--declaring the variables

--updating the qoh and decrementing it by the quantity ordered

--updating the date

begin

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

        select qoh,qoh_threshold into qoh_t,threshold_t from products where pid =
:new.pid;

        select last_visit_date into last_visit_date_t from customers where cid= :new.cid;

        if(TO_CHAR(last_visit_date_t,'dd-Mon-yyyy')!=TO_CHAR(SYSDATE,'dd-Mon-yyyy'))
then

                update customers set visits_made = visits_made+1, last_visit_date=sysdate
where cid = :new.cid;

        end if;

        -- if the quantity updated is lesser than therehold, supply tuple is inserted and the
supplier with least id is selected

        if(qoh_t < threshold_t) then

                dbms_output.put_line('quantity is less than threshold');

                dbms_output.put_line('qoh:' ||qoh);

                select min(sid) into smin from supply where pid= :new.pid;

                temp := threshold_t-qoh_t;
```

```
                    M := temp+1;

                    quan := M+10+qoh_t;

                    insert into supply values(sup_#.nextval, :new.pid,smin,SYSDATE,quan);

                    qoh_t:=qoh_t+quan;

                    update products set qoh=qoh_t where pid= :new.pid;

                    dbms_output.put_line('new qoh is :' ||qoh_t);

          end if;

end;

/

show errors
```

Mypack.java file

```
import java.util.Scanner;

import java.sql.*;

import oracle.jdbc.*;

import java.math.*;

import java.io.*;

import java.awt.*;

import oracle.jdbc.pool.OracleDataSource;


public class Mypack {


          public void show_employees(Connection conn)

          {

                    try

                    {

                  // calling the function in package func_package

                              CallableStatement cs = conn.prepareCall("{? = call
func_package.show_emp}");
```

```
                        cs.registerOutParameter(1, OracleTypes.CURSOR);
                        // execute the query
                        cs.executeQuery();
                        //retrieve the result
                        ResultSet rs = (ResultSet)cs.getObject(1);


                        //if the table is not empty
                                System.out.println("EID   ENAME      TELEPHONE#");
                                System.out.println("----------------------------");


                        while(rs.next()){
                                System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3));
                        }


                        cs.close();
                }
                catch (SQLException ex) { System.out.println ("\n*** SQLException caught
***\n" + ex.getMessage());}
                catch (Exception e) {System.out.println (e);}
        }


        public void show_customers(Connection conn)
        {
                try
                {
                        CallableStatement cs = conn.prepareCall("{? = call
func_package.show_cust}");
                        cs.registerOutParameter(1, OracleTypes.CURSOR);
                        // execute query
```

```
                    cs.executeQuery();
                    //retrieve the result
                    ResultSet rs = (ResultSet)cs.getObject(1);
                    // if the record is not empty
                System.out.println("CID   CNAME    TELEPHONE#    VISITS_MADE 
LAST_VISIT");
                System.out.println("-----------------------------------------------------");

                    while(rs.next()){
                        System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " + 
rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5));
                    }
                    cs.close();
                }
                catch (SQLException ex) { System.out.println ("\n*** SQLException caught 
***\n" + ex.getMessage());}
                catch (Exception e) {System.out.println (e);}
        }


        public void show_products(Connection conn)
        {
                try
                {
                    CallableStatement cs = conn.prepareCall("{? = call 
func_package.show_prod}");
                    cs.registerOutParameter(1, OracleTypes.CURSOR);
                    // execute query
                    cs.executeQuery();
                    // retrieve the result
                    ResultSet rs = (ResultSet)cs.getObject(1);
```

```
                        System.out.println("PID PNAME   QOH  THRESHOLD  ORIG_PRICE
DISC");
                        System.out.println("------------------------------------------");
                        while(rs.next()){
                                System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5) + " " + rs.getString(6));
                        }
                        cs.close();
                }
                catch (SQLException ex) { System.out.println ("\n*** SQLException caught
***\n" + ex.getMessage());}
                catch (Exception e) {System.out.println (e);}
        }


        public void show_purchases(Connection conn)
        {
                try
                {
                        CallableStatement cs = conn.prepareCall("{? = call
func_package.show_pur}");
                        cs.registerOutParameter(1, OracleTypes.CURSOR);
                        // execute the query
                        cs.executeQuery();
                        //retrieve the result
                        ResultSet rs = (ResultSet)cs.getObject(1);
                        System.out.println("PUR# EID PID CID QTY PTIME   TOTAL_PRICE");
                        System.out.println("--------------------------------------");
                        while(rs.next()){
                                System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5) + " " + rs.getString(6) + " " +
rs.getString(7));

                        }

                        cs.close();

                }

                catch (SQLException ex) { System.out.println ("\n*** SQLException caught
***\n" + ex.getMessage());}

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

        }



        public void show_suppliers(Connection conn)

        {

                try

                {

                        CallableStatement cs = conn.prepareCall("{? = call
func_package.show_sup}");

                        cs.registerOutParameter(1, OracleTypes.CURSOR);

                        // execute the query

                        cs.executeQuery();

                        //retrieve the result

                        ResultSet rs = (ResultSet)cs.getObject(1);

                        System.out.println("SID  SNAME  CITY    TELEPHONE#");

                        System.out.println("------------------------------");

                        while(rs.next()){

                                System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3) + " " + rs.getString(4));

                        }


                        cs.close();
```

```
                    }
                    catch (SQLException ex) { System.out.println ("\n*** SQLException caught
hello there***\n" + ex.getMessage());}
                    catch (Exception e) {System.out.println (e);}
          }



          public void show_supply(Connection conn)
          {
                    try
                    {
                              CallableStatement cs = conn.prepareCall("{? = call
func_package.show_supply}");
                              cs.registerOutParameter(1, OracleTypes.CURSOR);
                              // execute the query
                              cs.executeQuery();
                              //retrieve the result
                              ResultSet rs = (ResultSet)cs.getObject(1);
                              System.out.println("SUP#  PID  SID  SDATE  QUANTITY");
                              System.out.println("-------------------------------");
                              while(rs.next()){
                                        System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5));
                              }
                              cs.close();
                    }
                    catch (SQLException ex) { System.out.println ("\n*** SQLException caught
heyyyloooo***\n" + ex.getMessage());}
                    catch (Exception e) {System.out.println (e);}
          }
```

```
    public void show_logs(Connection conn)
    {
        try
        {
            CallableStatement cs = conn.prepareCall("{? = call func_package.show_logs}");
            cs.registerOutParameter(1, OracleTypes.CURSOR);
            // execute the query
            cs.executeQuery();
                ResultSet rs = (ResultSet)cs.getObject(1);
                //retrieve the result
                System.out.println("LOG#  WHO  OTIME  TABLE_NAME   OPERAT
KEY");
                System.out.println("----------------------------------------");
                while(rs.next()){
                System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3) + " " + rs.getString(4) + " " + rs.getString(5) + " " + rs.getString(6));
                }
            cs.close();
        }
        catch (SQLException ex) { System.out.println ("\n*** SQLException caught ***\n" +
ex.getMessage());}
        catch (Exception e) {System.out.println (e);}
    }


      public void report_monthly_sale(Connection conn)
      {
            Scanner in = new Scanner(System.in);  //prompting the user for pid
            try
```

```
                {

                        System.out.print("Enter the pid: \t");

                        String pid = in.nextLine();


                        CallableStatement cs = conn.prepareCall("{? = call
func_package.report_monthly_sale(?)}"); //calling package_name.funtion_name


                        cs.setString(2, pid);

                        cs.registerOutParameter(1, OracleTypes.CURSOR);

                        // execute the query

                        cs.executeQuery();

                        //retrieve the result

                        ResultSet rs = (ResultSet)cs.getObject(1);

                        System.out.println("PNAME   MONTH  YEAR  QTY AVG  TOTAL");

                        System.out.println("-----------------------------------");

                        while(rs.next()){

                                System.out.println(rs.getString(1)  +" " + rs.getString(2) + " " +
rs.getString(3)+ " " + rs.getString(4)+ " " + rs.getString(5)+ " " + rs.getString(6));

                        }

                        if(!rs.next())

                        {

                                System.out.println("PID not found");

                        }


                        cs.close();

                }


                catch (SQLException ex) {

                if(ex.getErrorCode()==20001) {

                System.out.println("PID not found");
                }
                System.out.println ("\n*** SQLException caught ***\n" + ex.getMessage());}
                catch (Exception e) {System.out.println (e);}


        }



        public void add_products(Connection conn)
        {
                Scanner in = new Scanner(System.in);
                try
                {
                        //prompting the user for all the required product entries
                        System.out.print("Enter the pid: \t");
                        String pid = in.nextLine();
                        System.out.print("Enter the pname: \t");
                        String pname = in.nextLine();
                        System.out.print("Enter the quantity: \t");
                        String qty = in.nextLine();
                        System.out.print("Enter the threshold: \t");
                        String threshold = in.nextLine();
                        System.out.print("Enter the original price: \t");
                        String price = in.nextLine();
                        System.out.print("Enter the discount rate: \t");
                        String Drate = in.nextLine();
               //Prepare to call stored procedure:
                        CallableStatement cs = conn.prepareCall("begin
insert_purchases.add_products(:1,:2,:3,:4,:5,:6); end;");
                        //setting the variables to the entered values
                        cs.setString(1, pid);
```

```
                    cs.setString(2, pname);

                    cs.setString(3, qty);

                    cs.setString(4, threshold);

                    cs.setString(5, price);

                    cs.setString(6, Drate);

                    //cs.registerOutParameter(7, Types.VARCHAR);

                    // execute and retrieve the result set

                    cs.executeQuery();

                    cs.close();

            }

            catch (SQLException ex) { System.out.println ("\n*** SQLException caught
***\n" + ex.getMessage());}

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

    }



    public void add_purchases(Connection conn)

    {

            Scanner in = new Scanner(System.in);

            try

            {                   // prompting the user for reuired entries

                    System.out.print("Enter eid: \t");

                    String eid = in.nextLine();

                    System.out.print("Enter pid: \t");

                    String prodid = in.nextLine();

                    System.out.print("Enter cid: \t");

                    String cid = in.nextLine();

                    System.out.print("Enter purchase quantity: \t");

                    String quantity = in.nextLine();
                    String query = "SELECT qoh FROM products where pid like ?";
                    PreparedStatement pstmt = conn.prepareStatement(query);
                    pstmt.setString(1,prodid);
                    ResultSet rs = pstmt.executeQuery();
                    rs.next();

                    int val =  ((Number) rs.getObject(1)).intValue();
                    int qty = Integer.parseInt(quantity);

                    if (qty > val) {  //if quantity is greater than threshold add it into the
purchase table
                            CallableStatement cs = conn.prepareCall("begin
insert_purchases.add_purchases(:1,:2,:3,:4); end;");

                cs.setString(1, eid);
                cs.setString(2, prodid);
                cs.setString(3, cid);
                cs.setString(4, quantity);
                cs.executeQuery();
                cs.close();
                    }
                    else {   //if quantity is less than therhold , update the values
                    int remQty = val - qty;
                    query = "SELECT qoh_threshold FROM products where pid like ?";
                    pstmt = conn.prepareStatement(query);
                    pstmt.setString(1,prodid);
                    ResultSet rss = pstmt.executeQuery();
                    rss.next();
                    int thres =  ((Number) rss.getObject(1)).intValue();
                            if ( thres > remQty) {  //trigger statement
```

```
                                        System.out.println("quantity less than threshold, new
supply is needed");
                                }


                                CallableStatement cs = conn.prepareCall("begin
insert_purchases.add_purchases(:1,:2,:3,:4); end;");
                                // setting the entries to the corresponding values
                                cs.setString(1, eid);
                                cs.setString(2, prodid);
                                cs.setString(3, cid);
                                cs.setString(4, quantity);
                                //cs.registerOutParameter(5, Types.VARCHAR);
                                // execute and retrieve the result set
                                cs.executeQuery();
                                cs.close();
                        }
                }
                catch (SQLException ex) { System.out.println ("\n*** SQLException caught
***\n" + ex.getMessage());}
                catch (Exception e) {System.out.println (e);}
        }


}
```

Display.java file

```
import java.sql.*;


import java.util.ArrayList;
import java.util.Scanner;
import oracle.jdbc.*;
import java.math.*;
import java.io.*;

import java.awt.*;

import oracle.jdbc.pool.OracleDataSource;


public class Display {

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

                try

                {

                        //connecting 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("jnuti1","a1b2c3d4SRI");
//opening the connection


                        Scanner in = new Scanner(System.in);


                        if(conn!=null){

                                //make an array of valid choices

                                ArrayList<String> validChoices = new ArrayList<String>();

                                for(int i = 0; i < 11; i++){

                                        validChoices.add(Integer.toString(i));

                                }

                                String choice = "";

                                while(!validChoices.equals("0")){

                                        while(!validChoices.contains(choice)){

                                                System.out.print("\n0: Exit\n" +

                                                                "1: Display employees\n2:
Display customers\n3: Display products\n4: Display purchases\n5: Dsiplay suppliers\n6:
Display supply\n" +
                                        "7: Display logs\n8: Display
monthly sale information\n9: insert into products\n10: insert into purchases\n\n" +
                                        "Enter your choice: \t \n");
                              choice = in.nextLine();
                         }

                         Mypack mp = new Mypack();

                         // Displaying options
                    if(choice.equals("1")){
                              mp.show_employees(conn);
                         }else if(choice.equals("2")){
                              mp.show_customers(conn);
                         }else if(choice.equals("3")){
                              mp.show_products(conn);
                         }else if(choice.equals("4")){
                              mp.show_purchases(conn);
                         }else if(choice.equals("5")){
                              mp.show_suppliers(conn);
                         }else if(choice.equals("6")){
                              mp.show_supply(conn);
                         }else if(choice.equals("7")){
                    mp.show_logs(conn);
                    }else if(choice.equals("8")){
                              mp.report_monthly_sale(conn);
                         }else if(choice.equals("9")){
                              mp.add_products(conn);
                         }else if(choice.equals("10")){
                              mp.add_purchases(conn);
                         }
```

```
                                        if(!choice.equals("0")){
                                                choice = "";
                                        }else{
                                                break;
                                        }
                                }
                        }
                        conn.close(); //closing the connection
                }
                catch (SQLException ex) { System.out.println ("\n*** SQLException caught
***\n" + ex.getMessage());}
                catch (Exception e) {System.out.println (e);}
        }
}
```

Team Report:

Meetings, plans and implementations:

We have made consistent effort throughout the period to accomplish this project. Following are the details about our overall effort made:

No	Agenda	Work done	Date	
1	Planning of work distribution and strategy	- Discussed how to accomplish few tasks and estimated the total time needed to complete this project	11/17/2015	
2.	Implemented Show tables to display content of the table	- Discussed and implemented show tables (for ex: show_employees) - Checked for different values	11/18/2015	
3.	Implemented sequences and monthly sale report query	- Discussed and implemented sequences and method to accomplish the task.	11/19/2015	
4.	Implemented add_purchase and add_products query	- Implemented procedure to accomplish the task.	11/22/2015	
5.	Started working on JDBC code with the queries	- Practiced and tried out researching about JDBC and java code.	11/23/2015	
6.	Implemented triggers to update logs table for every update with other tables.	- Implemented trigger and tested them for different set of inputs. And Validations are added too.	11/23/2015	
7.	Planning and implementation of trigger that returns error message if amount entered by user is not there in qoh and started working on triggers for add_purchase	- Implemented trigger to print that message and tested it.	11/25/2015	
8	Worked on triggers for add_purchase	- Worked on and implemented trigger for printing message for question 7 in the project and tested it for various set of input values.	11/26/2015	
9.	Added validations and exceptions to the program and written java code	- Checked and find places where exceptions should be handled and validations are put. Tested programs for corner cases.	11/28/2015	
10	Testing of All the modules	- Testing of all the codes and queries.	12/30/2015	

Responsibilities:

Queries:

Discussed by both and work was divided equally and help each other for few instances.

JDBC:

As mentioned above, JDBC code was implemented while working on the queries. Both have put equal effort to write, validate and test this JDBC code.

Interface:

Interface is implemented by Chetas. Divya has checked for validations and tested the code and corrected few errors.

Overall, It was really great learning experience and good team work experience.

I agree with this team report:

Name: Sign