PROJECT REPORT

Retail Business Management System

Created By

Yash Jagda-B00809607

Shrijeet Rupnar – B00808280

CS 532 – Database System Project

PL/SQL and JDBC Connectivity

Department of Computer Science

State University Of New York At Binghamton

2nd April 2020

INTRODUCTION

This project enables the user to access various details regarding the Retail Business. The user can easily interact with the system using a GUI which is connected to the database. The user can also add customers, All operations on the database like insert, update, view tables for this project have been implemented using PL/SQL along with functions, procedures, sequences, triggers which have then been implemented in the Menu driven Java Interface. The Logs table also helps to keep track of the various updates or inserts made into the various tables using triggers and sequences.

IMPLEMENTATION

Implementation consists of the following steps

1) Creation of Database tables in Oracle 11g
2) Package creation which will contain the procedures and functions which will perform various insert, delete and select operations in PL/SQL.
3) Sequence generation to autoincrement the PUR#, LOG# and SUP#.
4) Creation of triggers to update the respective tables as well as to keep a check on various conditions as required by this project.

SEQUENCES CREATED FOR THE PROJECT

These are the sequences that have been created for this project.

log_seq - This sequence is used to generate log# values for the Logs table when a new entry is made into the table. It will begin from 1 , increment by 1 and Max value will be 100. Below is the code.

```
create sequence log_seq
increment by 1
start with 1
 maxvalue 100
cycle
order;
```

pur_sequence - This sequence is used to generate pur# values for the purchases table when a new entry is made into the table. It will begin from 100001 , increment by 1 and Max value will be 999999. Below is the code.

```
create sequence pur_sequence
increment by 1
start with 100001
maxvalue 999999
cycle
order;
  /
```

sup_sequence - This sequence is used to generate sup# values for the supplies table when a new entry is made into the table. It will begin from 1 , increment by 1 and Max value will be 100. Below is the code.

```
create sequence sup_sequence
        increment by 1
        start with 1
        maxvalue 100
        nocache
        cycle;
        /
```

PACKAGE CREATED FOR THE PROJECT

The Package named project2 has been created for this project which contains all the procedures and functions used by the project. This package is created as follows.

1) Creating the package with the declaration of procedures and functions. Below is the code for the same.

```
create or replace package project2 as
type ref_cursor is ref cursor;

function getproducts
    return ref_cursor;

function getcustomers
    return ref_cursor;

function getpurchases
    return ref_cursor;

function getemployees
    return ref_cursor;

function getsuppliers
    return ref_cursor;

function getsupply
    return ref_cursor;

function getlogs
    return ref_cursor;

PROCEDURE report_monthly_sale(prod_id IN purchases.pid%type,Invaliderror OUT
varchar2,c1 OUT sys_refcursor);
```

```
procedure add_product(p_id in products.pid%type,
pname in products.pname%type,
qoh in products.qoh%type,
qoh_threshold in products.qoh_threshold%type,
original_price in products.original_price%type,
discnt_rate in products.discnt_rate%type,
error out varchar2);


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_output out varchar,
error out varchar2);


end project2;
/
```

PROCEDURES AND FUNCTIONS USED IN THE PACKAGE

PROCEDURES DEFINED IN THIS PACKAGE

```
set serveroutput on
create or replace package body project2 as

function getproducts
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from products;
   return rc;
end;

function getcustomers
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from customers;
   return rc;
end;

function getpurchases
```

```
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from purchases;
   return rc;
end;


function getemployees
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from employees;
   return rc;
end;


function getsuppliers
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from suppliers;
   return rc;
end;


function getsupply
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from supply;
   return rc;
end;


function getlogs
return ref_cursor as
rc ref_cursor;
begin
   open rc for
   select * from logs;
   return rc;
end;

PROCEDURE report_monthly_sale(prod_id IN purchases.pid%type,Invaliderror OUT
varchar2,c1 OUT sys_refcursor)
IS

Invalidpid exception;
count_values number;
```

```
BEGIN
select count(*) INTO count_values FROM purchases where pid = prod_id;
if(count_values = 0) THEN
        raise Invalidpid;
else
        OPEN c1 FOR
SELECT pur.pid,prod.pname,to_char(pur.ptime,'MON-
YYYY')"Month",sum(pur.qty)"Quantity",sum(pur.total_price)"total_price",sum(pur.total_p
rice)/sum(pur.qty)"Average" FROM purchases pur,products prod where pur.pid=prod.pid
and pur.pid = prod_id group by pur.pid,prod.pname,to_char(pur.ptime, 'MON-YYYY');
end if;
exception
        when Invalidpid then
        Invaliderror:='Does not exist';
end report_monthly_sale;
```

```
procedure add_product(p_id in products.pid%type,
pname in products.pname%type,
qoh in products.qoh%type,
qoh_threshold in products.qoh_threshold%type,
original_price in products.original_price%type,
discnt_rate in products.discnt_rate%type,
error out varchar2) is

pid_error exception;
pid_count number;

BEGIN
select count(*) into pid_count from products where pid = p_id;
if(pid_count=0) then
raise pid_error;
else
insert into products (pid,pname,qoh,qoh_threshold,original_price,discnt_rate) values
(p_id,pname,qoh,qoh_threshold,original_price, discnt_rate);
dbms_output.put_line('Added Successfully');

end if;

exception
when pid_error then
error:='Product does not exists';

END add_product;
```

```
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_output out varchar,
error out varchar2) is

pid_error exception;
eid_error exception;
cid_error exception;
pur_date date;
pur_total_price number(7,2);
next_pur# number(6);
og_price number(6,2);
pid_count number;
eid_count number;
cid_count number;

BEGIN
pur_date:=SYSDATE;
select count(*) into pid_count from products where pid = p_id;
select count(*) into eid_count from employees where eid = e_id;
select count(*) into cid_count from customers where cid = c_id;
SELECT prod.original_price into og_price from products prod where prod.pid = p_id;
pur_total_price := og_price * pur_qty;

if(eid_count=0) then
raise eid_error;

elsif(pid_count=0) then
raise pid_error;

elsif(cid_count=0) then
raise cid_error;

else


next_pur#:=pur_sequence.nextval;
insert into purchases values (next_pur#,e_id,p_id,c_id,
pur_qty, pur_date, pur_total_price);
dbms_output.put_line('Purchase Successful');

end if;

exception
when eid_error then
error:='Employee does not exists';

when pid_error then
error:='Product does not exists';

when cid_error then
```

```
error:='Customer does not exists';

END add_purchase;
```

TRIGGERS CREATED FOR THE PROJECT

1. **LOG_TRIGGER_INSERT_PURCHASES** – This trigger will be fired when the user performs an insert on the purcases table. This trigger will insert, When a tuple is added to the logs table due to the 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. Below is the code.

```
CREATE OR REPLACE TRIGGER LOG_TRIGGER_INSERT_PURCHASES
AFTER INSERT ON PURCHASES
FOR EACH ROW
BEGIN
INSERT INTO LOGS
(LOG#,WHO,OTIME,TABLE_NAME,OPERATION,KEY_VALUE)
VALUES(LOG_SEQ.NEXTVAL,user,sysdate,'PURCHASES','INSERT',:NEW.pur#);
END;
 /
```

2. **LOG__TRIGGER_UPDATE_PRODUCTS** – This trigger will be fired when update the qoh attribute of the products table and will entry into LOGS table_name should be "products", the operation should be "update" and the key_value should be the pid of the affected product e.

```
CREATE OR REPLACE TRIGGER LOG__TRIGGER_UPDATE_PRODUCTS
AFTER UPDATE OF QOH ON PRODUCTS
FOR EACH ROW
BEGIN
INSERT INTO LOGS
(LOG#,WHO,OTIME,TABLE_NAME,OPERATION,KEY_VALUE)
VALUES(LOG_SEQ.NEXTVAL,user,sysdate,'PRODUCTS','UPDATE',:NEW.pid);
END;
 /
```

3. **LOG__TRIGGER_UPDATE_CUSTOMERS** – This trigger will be fired when update the visits_made attribute of the customers table and then entry into LOGS will be like, 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 LOG__TRIGGER_UPDATE_CUSTOMERS
AFTER UPDATE OF VISITS_MADE ON CUSTOMERS
FOR EACH ROW
BEGIN
INSERT INTO LOGS
(LOG#,WHO,OTIME,TABLE_NAME,OPERATION,KEY_VALUE)
VALUES(LOG_SEQ.NEXTVAL,user,sysdate,'CUSTOMERS','UPDATE',:NEW.cid);
END;
/
```

4. **LOG__TRIGGER_UPDATE_SUPPLY** – This trigger will be fired when the user performs an insert on the supplies table. This trigger will insert a entry in to logs table the table_name should be “supply”, the operation should be “insert” and the key_value should be the sup# of the newly inserted supply . Below is the code.

```
CREATE OR REPLACE TRIGGER LOG__TRIGGER_UPDATE_SUPPLY
AFTER INSERT ON SUPPLY
FOR EACH ROW
BEGIN
INSERT INTO LOGS
(LOG#,WHO,OTIME,TABLE_NAME,OPERATION,KEY_VALUE)
VALUES(LOG_SEQ.NEXTVAL,user,sysdate,'SUPPLY','INSERT',:NEW.sup#);
END;
/
```

5. **update_QOH_Check -** Before a purchase is made (i.e., before a tuple is added into the purchases table), your 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 (e.g., “Insufficient quantity in stock.”) and the purchase request should be rejected.

```
create or replace trigger update_QOH_Check
Before insert on purchases
FOR EACH ROW
declare qoh_Insufficient exception;
        qoh_p number;
BEGIN
select qoh into qoh_p from products pr where pr.pid = :new.pid;
 if (:new.qty > qoh_p) then
        raise qoh_Insufficient;
 end if;
exception
when qoh_Insufficient then
      raise_application_error(-20003,'qoh_Insufficient');
end;
/
```

6. **UPDATE_QOHT** – 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:

    1. print a message saying that the current qoh of the product is below the required threshold and new supply is required;
    2. 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 M + qoh + 5, where M is the minimum value for quantity such that M + qoh > qoh_threshold, and use sysdate for sdate;
    3. increase qoh of the product by the quantity ordered;
    4. print another message showing the new value of the qoh of the product; and

```
create or replace trigger UPDATE_QOHT
after insert on purchases
declare
pur#_id purchases.pur#%type;
p_id purchases.pid%type;
c_id purchases.cid%type;
pur_qty purchases.qty%type;
sup#_id supply.sup#%type;
sup_qty supply.quantity%type;
temp_qoh_threshold products.qoh_threshold%type;
new_qoh products.qoh%type;
temp_visits_made customers.visits_made%type;
s_sid supply.sid%type;
sup_date date;
pdate date;
last_visit date;

BEGIN
Select sysdate into sup_date from dual;
Select sysdate into pdate from dual;
select pur#,pid,cid,qty,ptime into pur#_id,p_id,c_id,pur_qty,last_visit from purchases group by
pur#,pid,cid,qty,ptime having pur#=(select max(pur#) from purchases);
update products set qoh=qoh-pur_qty where pid=p_id;
select qoh, qoh_threshold into new_qoh, temp_qoh_threshold from products pr where pr.pid =
p_id;
select visits_made INTO temp_visits_made from customers where cid=c_id;
if(last_visit_date !=: pdate)
update customers set visits_made = temp_visits_made+1 , last_visit_date = last_visit where
cid=c_id;

if (new_qoh < temp_qoh_threshold) then
   dbms_output.put_line('Quantity on hand(qoh) is below the required threshold and new
supply is required');
  sup_qty:=10+temp_qoh_threshold+1;
   select sid into s_sid from (select sid from supply where pid=p_id order by sid asc) where
rownum = 1;
```

```
    insert into supply values (sup_sequence.nextval, p_id, s_sid, sup_date, sup_qty);
    update products set qoh=(qoh+sup_qty) where pid=p_id;
    dbms_output.put_line('New QOH: ' || (new_qoh+sup_qty));
  end if;
  end;
/
```

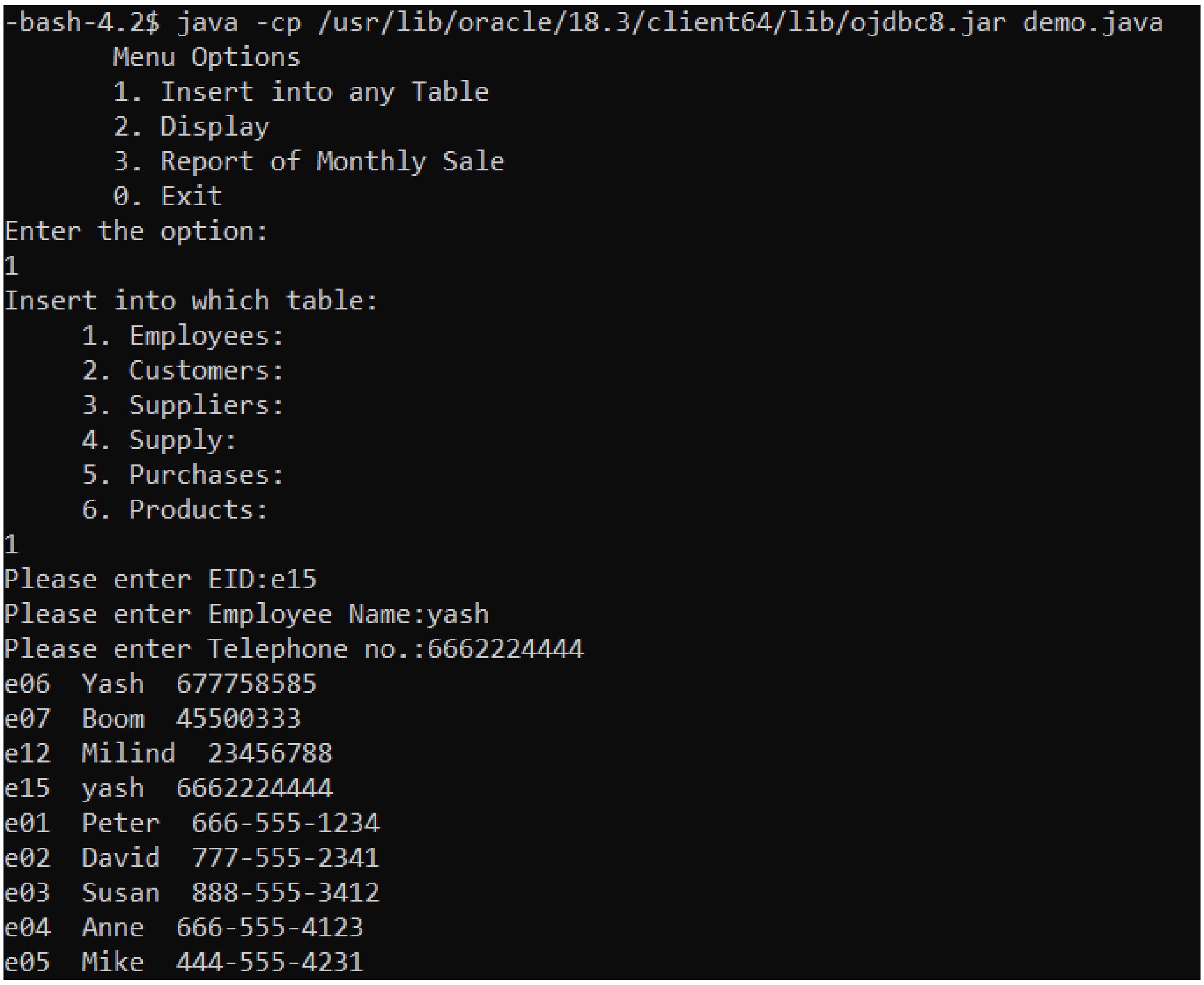
7. **PRODUCT_TRIGGER** – This trigger will be fired when a tuple in the purchases table has been deleted. It will increment the qoh value of that product in the products table by the qoh value recently deleted. It will also increment the visits_made in the customers table for that customer by 1. Also, the last_visit_date will be the current date. Below is the code.

```
 CREATE OR REPLACE TRIGGER PRODUCT_TRIGGER
AFTER DELETE ON PURCHASES
FOR EACH ROW
DECLARE
PROD_ID PURCHASES.PID%TYPE;
LAST_DATE PURCHASES.PTIME%TYPE;
BEGIN
UPDATE PRODUCTS SET PRODUCTS.QOH=PRODUCTS.QOH+:old.qty
WHERE PRODUCTS.PID=:old.pid;
UPDATE CUSTOMERS SET VISITS_MADE=VISITS_MADE+1,
LAST_VISIT_DATE=sysdate
WHERE CID=:old.cid;
END;
/
```

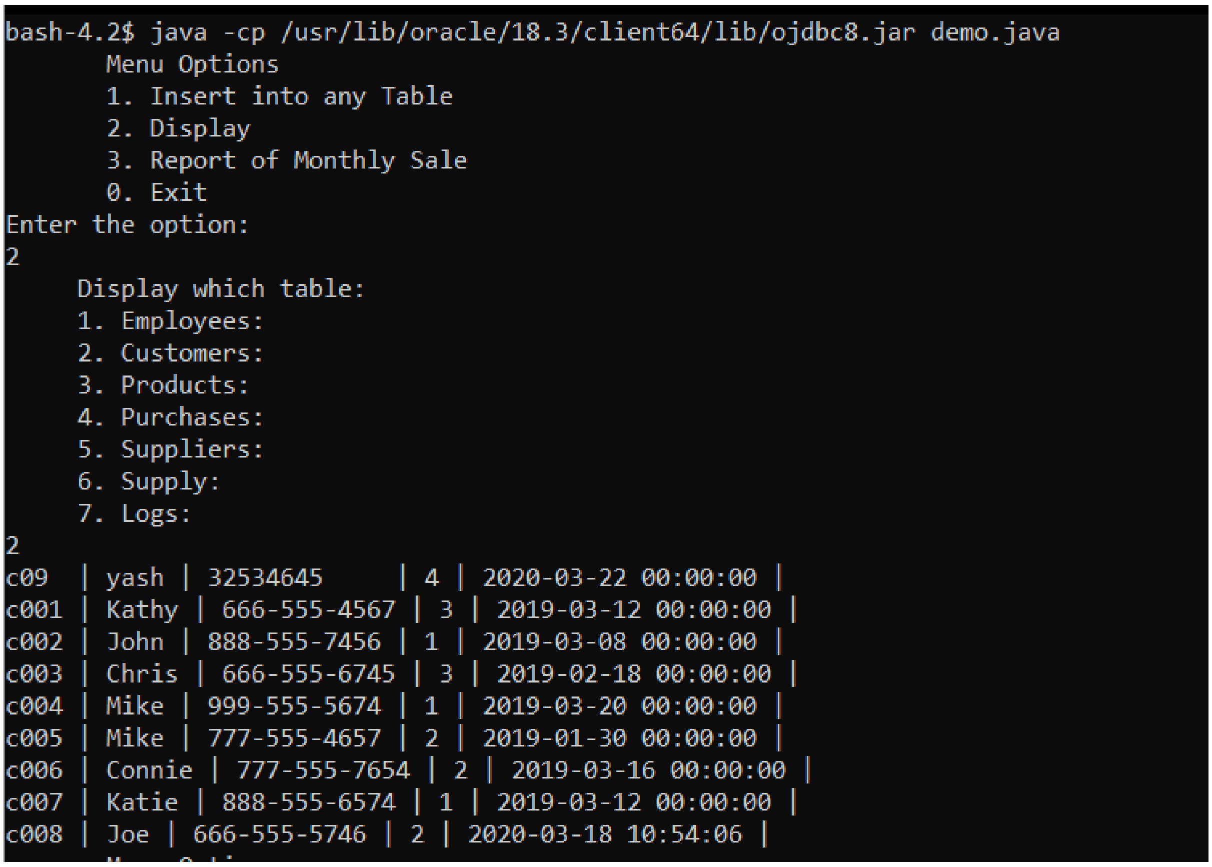
INTERFACE IMPLEMENTATION

All operations on the database like insert, update, view tables for this project have been implemented using PL/SQL along with functions, procedures, sequences, triggers which have then been implemented in the Menu driven Java Interface

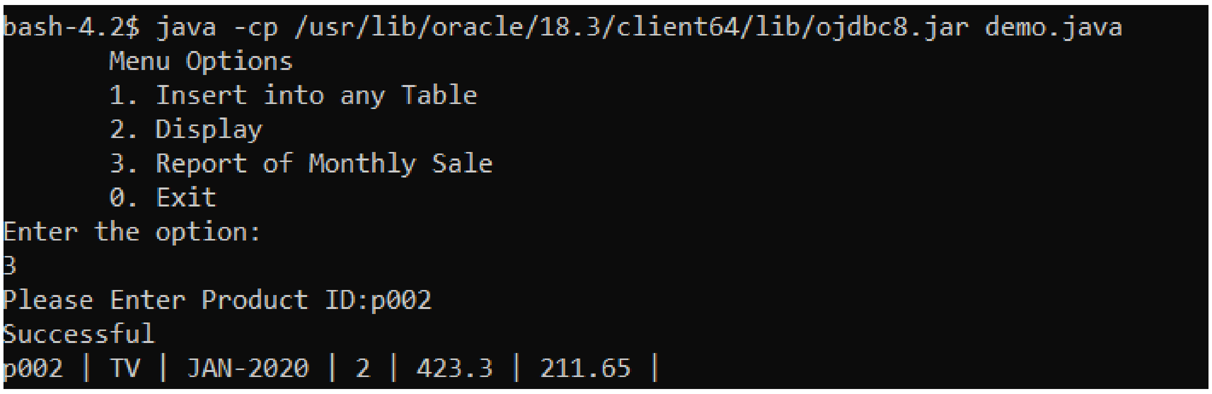
Result



```
-bash-4.2$ java -cp /usr/lib/oracle/18.3/client64/lib/ojdbc8.jar demo.java
       Menu Options
       1. Insert into any Table
       2. Display
       3. Report of Monthly Sale
       0. Exit
Enter the option:
1
Insert into which table:
     1. Employees:
     2. Customers:
     3. Suppliers:
     4. Supply:
     5. Purchases:
     6. Products:
1
Please enter EID:e15
Please enter Employee Name:yash
Please enter Telephone no.:6662224444
e06  Yash  677758585
e07  Boom  45500333
e12  Milind  23456788
e15  yash  6662224444
e01  Peter  666-555-1234
e02  David  777-555-2341
e03  Susan  888-555-3412
e04  Anne  666-555-4123
e05  Mike  444-555-4231
```



```
bash-4.2$ java -cp /usr/lib/oracle/18.3/client64/lib/ojdbc8.jar demo.java
       Menu Options
       1. Insert into any Table
       2. Display
       3. Report of Monthly Sale
       0. Exit
Enter the option:
2
     Display which table:
     1. Employees:
     2. Customers:
     3. Products:
     4. Purchases:
     5. Suppliers:
     6. Supply:
     7. Logs:
2
c09  | yash | 32534645      | 4 | 2020-03-22 00:00:00 |
c001 | Kathy | 666-555-4567 | 3 | 2019-03-12 00:00:00 |
c002 | John | 888-555-7456 | 1 | 2019-03-08 00:00:00 |
c003 | Chris | 666-555-6745 | 3 | 2019-02-18 00:00:00 |
c004 | Mike | 999-555-5674 | 1 | 2019-03-20 00:00:00 |
c005 | Mike | 777-555-4657 | 2 | 2019-01-30 00:00:00 |
c006 | Connie | 777-555-7654 | 2 | 2019-03-16 00:00:00 |
c007 | Katie | 888-555-6574 | 1 | 2019-03-12 00:00:00 |
c008 | Joe | 666-555-5746 | 2 | 2020-03-18 10:54:06 |
```



```
bash-4.2$ java -cp /usr/lib/oracle/18.3/client64/lib/ojdbc8.jar demo.java
       Menu Options
       1. Insert into any Table
       2. Display
       3. Report of Monthly Sale
       0. Exit
Enter the option:
3
Please Enter Product ID:p002
Successful
p002 | TV | JAN-2020 | 2 | 423.3 | 211.65 |
```

RESULT

All queries were successfully executed and implemented using PL/SQL and JDBC connectivity. Also, we have created a Menu driven Java Interface.

COLLABORATION REPORT

TEAM MEMBERS

Yash Jagda

Shrijeet Rupnar

10th March 2020
During our first meeting we had a brief discussion about the project. We also discussed how to design the GUI and what should be displayed on it.

15th March 2020
The 8 questions were divided amongst us.
Yash (1,2,3,4) Shrijeet (5,6,7).

16th March 2020 to 20th March 2020
We finished the PL/SQL.

25th March 2020
We met to discuss various errors that were encountered and resolved them. Also, we tried different test cases. On successful implementation of the procedures and functions we created a package with the procedures and functions while the triggers and sequences were outside the package.

26th March 2020 to 29th March 2020
The GUI was then developed by Yash. The JDBC connections were done by Yash. It was then combined, and the final GUI was made functional.

30th March 2020
Once the GUI was developed, we began connecting the PL/SQL code with the database.

31st March 2020
Shrijeet created the report for the project. Shrijeet executed various test cases.

It was a good experience working on this project as we got to learn new concepts as well as it improved our knowledge base.