<u>PROJECT - 2 REPORT</u>

RETAIL BUSINESS MANAGEMENT SYSTEM

(CS-558-02)

BY

Sri Harsha Tanamala – B00979277
Akhil Madiri – B00978616

Package Description and Objectives:

Our PL/SQL package is designed to help a retail business manage its database. It includes procedures and functions for managing employees, customers, products, purchases, and logs. The main goals of our package are:

- Make it easy for users to generate unique values for pur# and log# using sequences.
- Allow users to view information from each table in the database.
- Help users report monthly sales activities for any given employee.
- Help users add new employees and automatically log these changes.
- Help users add new purchases and automatically adjust the qoh attribute of the Products table and log these changes.
- Use triggers to log changes to the database automatically.
- Provide clear error messages for any issues that arise.

Procedure and Function Descriptions:

Our package includes the following procedures to help manage the database:

- **show_employees():** Displays all tuples in the Employees table.
- **show_customers():** Displays all tuples in the Customers table.
- **show_products():** Displays all tuples in the Products table.
- **show_prod_discnt():** Displays all tuples in the Product_discnt table
- **show_purchases():** Displays all tuples in the Purchases table.
- **show_logs():** Displays all tuples in the Logs table.
- **monthly_sale_activities(employee_id):** Reports monthly sales activities for a given employee.
- **add_employee(e_id , e_name , e_telephone#, e_email):** Adds a new tuple to the Employees table.
- **add_purchase(e_id, p_id, c_id, pur_qty, pur_unit_price):** Adds a new tuple to the Purchases table, adjusts the qoh attribute of the Products table.

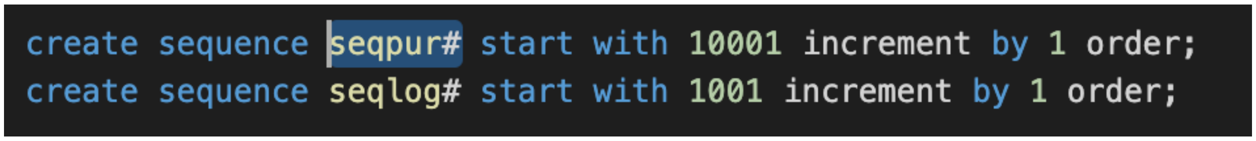
Code Description:

As per the questions given, we have implemented the package procedures and triggers.

The initial script given for project 1 consists of DDL of five tables and data for it.
We created log table with the given ddl in questions.

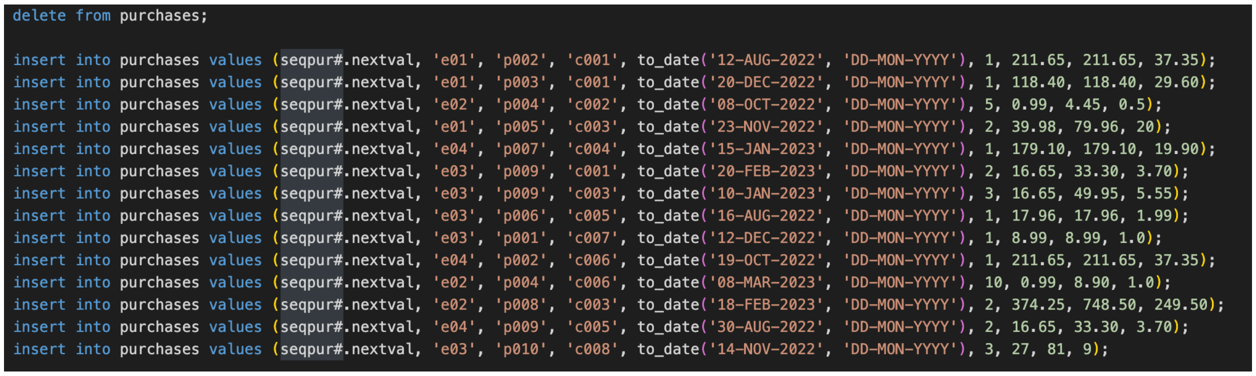
1) For the first question,

We have created two sequences.
Seqpur# for pur# column in purchases table
Seqlog# for log# column in logs table



```
create sequence seqpur# start with 10001 increment by 1 order;
create sequence seqlog# start with 1001 increment by 1 order;
```

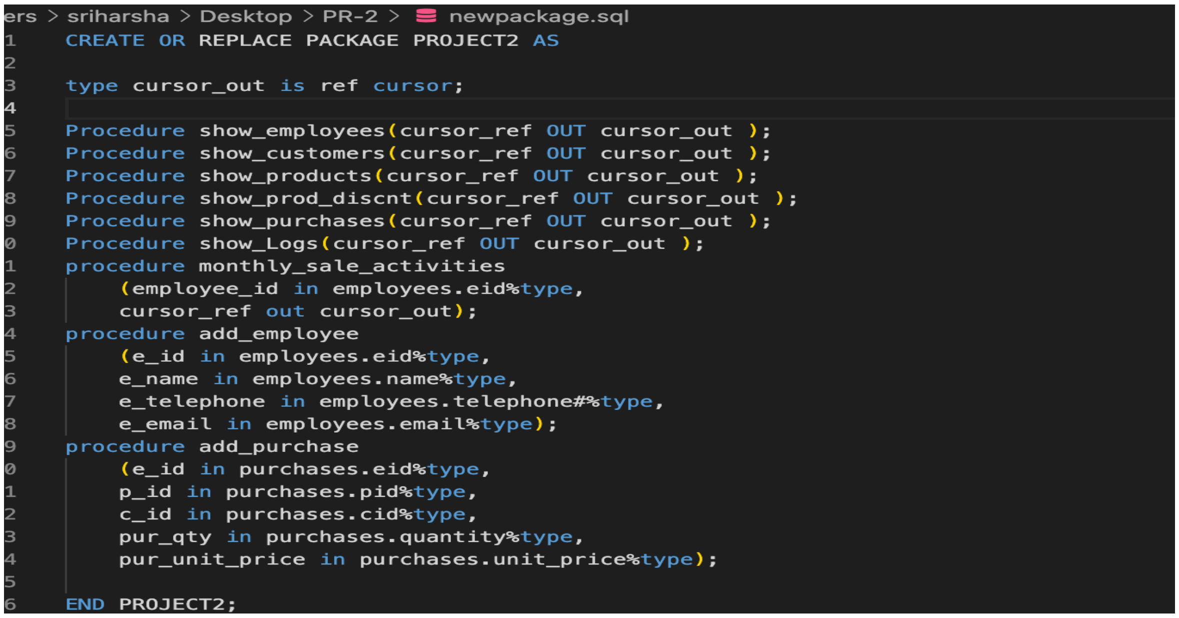
Deleted the data from purchases table and reloaded with same data with no pur#(it will generate from the sequence created.



```
delete from purchases;

insert into purchases values (seqpur#.nextval, 'e01', 'p002', 'c001', to_date('12-AUG-2022', 'DD-MON-YYYY'), 1, 211.65, 211.65, 37.35);
insert into purchases values (seqpur#.nextval, 'e01', 'p003', 'c001', to_date('20-DEC-2022', 'DD-MON-YYYY'), 1, 118.40, 118.40, 29.60);
insert into purchases values (seqpur#.nextval, 'e02', 'p004', 'c002', to_date('08-OCT-2022', 'DD-MON-YYYY'), 5, 0.99, 4.45, 0.5);
insert into purchases values (seqpur#.nextval, 'e01', 'p005', 'c003', to_date('23-NOV-2022', 'DD-MON-YYYY'), 2, 39.98, 79.96, 20);
insert into purchases values (seqpur#.nextval, 'e04', 'p007', 'c004', to_date('15-JAN-2023', 'DD-MON-YYYY'), 1, 179.10, 179.10, 19.90);
insert into purchases values (seqpur#.nextval, 'e03', 'p009', 'c001', to_date('20-FEB-2023', 'DD-MON-YYYY'), 2, 16.65, 33.30, 3.70);
insert into purchases values (seqpur#.nextval, 'e03', 'p009', 'c003', to_date('10-JAN-2023', 'DD-MON-YYYY'), 3, 16.65, 49.95, 5.55);
insert into purchases values (seqpur#.nextval, 'e03', 'p006', 'c005', to_date('16-AUG-2022', 'DD-MON-YYYY'), 1, 17.96, 17.96, 1.99);
insert into purchases values (seqpur#.nextval, 'e03', 'p001', 'c007', to_date('12-DEC-2022', 'DD-MON-YYYY'), 1, 8.99, 8.99, 1.0);
insert into purchases values (seqpur#.nextval, 'e04', 'p002', 'c006', to_date('19-OCT-2022', 'DD-MON-YYYY'), 1, 211.65, 211.65, 37.35);
insert into purchases values (seqpur#.nextval, 'e02', 'p004', 'c006', to_date('08-MAR-2023', 'DD-MON-YYYY'), 10, 0.99, 8.90, 1.0);
insert into purchases values (seqpur#.nextval, 'e02', 'p008', 'c003', to_date('18-FEB-2023', 'DD-MON-YYYY'), 2, 374.25, 748.50, 249.50);
insert into purchases values (seqpur#.nextval, 'e04', 'p009', 'c005', to_date('30-AUG-2022', 'DD-MON-YYYY'), 2, 16.65, 33.30, 3.70);
insert into purchases values (seqpur#.nextval, 'e03', 'p010', 'c008', to_date('14-NOV-2022', 'DD-MON-YYYY'), 3, 27, 81, 9);
```

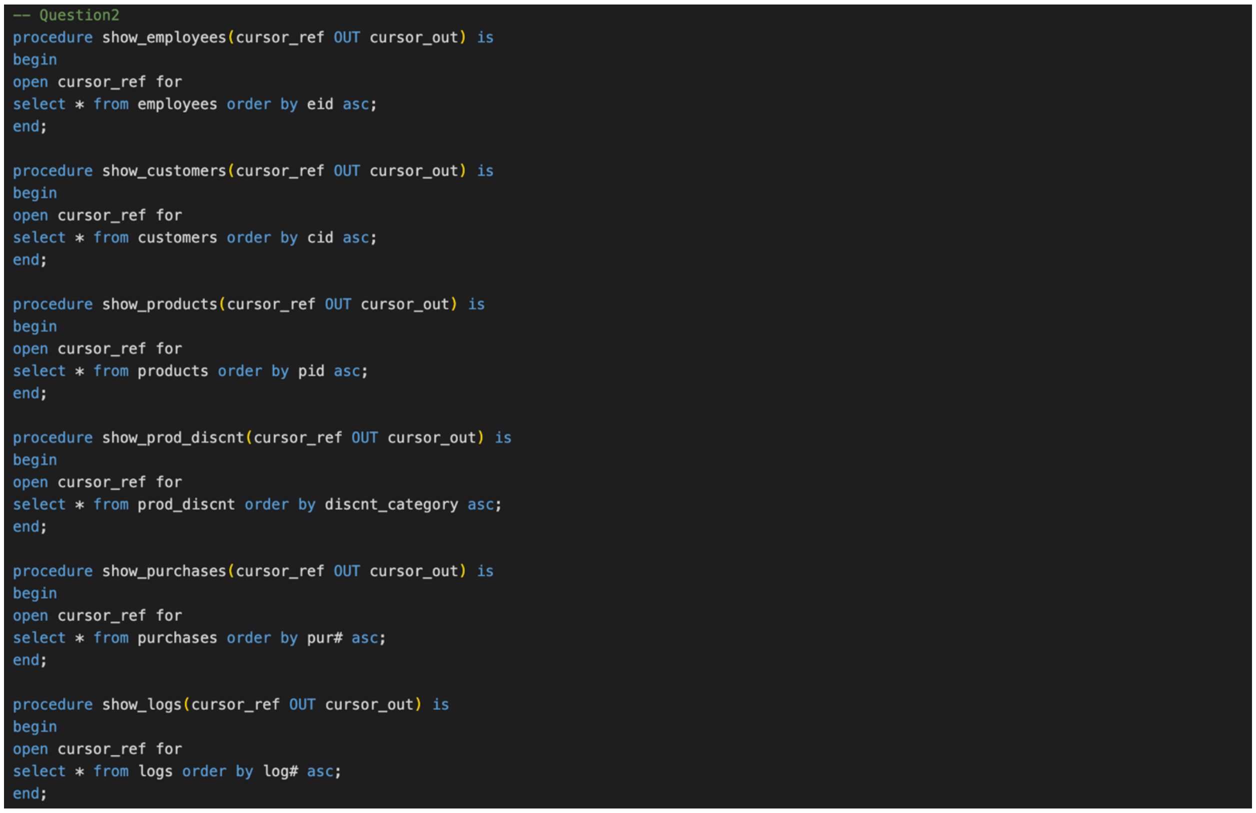
2) For the second question,

Created the package and the procedures



```
ers > sriharsha > Desktop > PR-2 > newpackage.sql
CREATE OR REPLACE PACKAGE PROJECT2 AS

type cursor_out is ref cursor;

Procedure show_employees(cursor_ref OUT cursor_out );
Procedure show_customers(cursor_ref OUT cursor_out );
Procedure show_products(cursor_ref OUT cursor_out );
Procedure show_prod_discnt(cursor_ref OUT cursor_out );
Procedure show_purchases(cursor_ref OUT cursor_out );
Procedure show_Logs(cursor_ref OUT cursor_out );
procedure monthly_sale_activities
    (employee_id in employees.eid%type,
    cursor_ref out cursor_out);
procedure add_employee
    (e_id in employees.eid%type,
    e_name in employees.name%type,
    e_telephone in employees.telephone#%type,
    e_email in employees.email%type);
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.quantity%type,
    pur_unit_price in purchases.unit_price%type);

END PROJECT2;
```

Firstly, we have created the 6 procedures to show the data present in the 6 tables respectively.

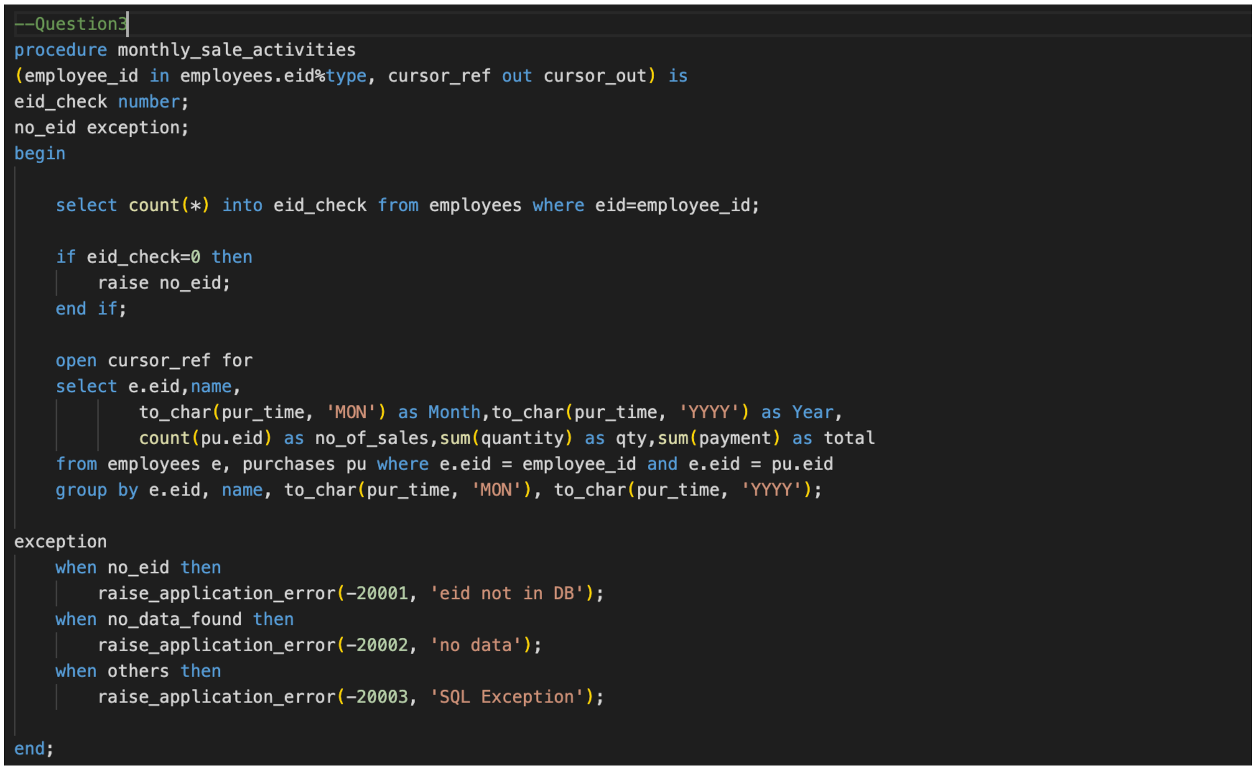


```
-- Question2
procedure show_employees(cursor_ref OUT cursor_out) is
begin
open cursor_ref for
select * from employees order by eid asc;
end;

procedure show_customers(cursor_ref OUT cursor_out) is
begin
open cursor_ref for
select * from customers order by cid asc;
end;

procedure show_products(cursor_ref OUT cursor_out) is
begin
open cursor_ref for
select * from products order by pid asc;
end;

procedure show_prod_discnt(cursor_ref OUT cursor_out) is
begin
open cursor_ref for
select * from prod_discnt order by discnt_category asc;
end;

procedure show_purchases(cursor_ref OUT cursor_out) is
begin
open cursor_ref for
select * from purchases order by pur# asc;
end;

procedure show_logs(cursor_ref OUT cursor_out) is
begin
open cursor_ref for
select * from logs order by log# asc;
end;
```

3) For the third question,

We have created a procedure **monthly_sale_activities** to show the information sales made by employee month wise.

For this procedure we take **employee_id** as input and return the sales done by the employee.

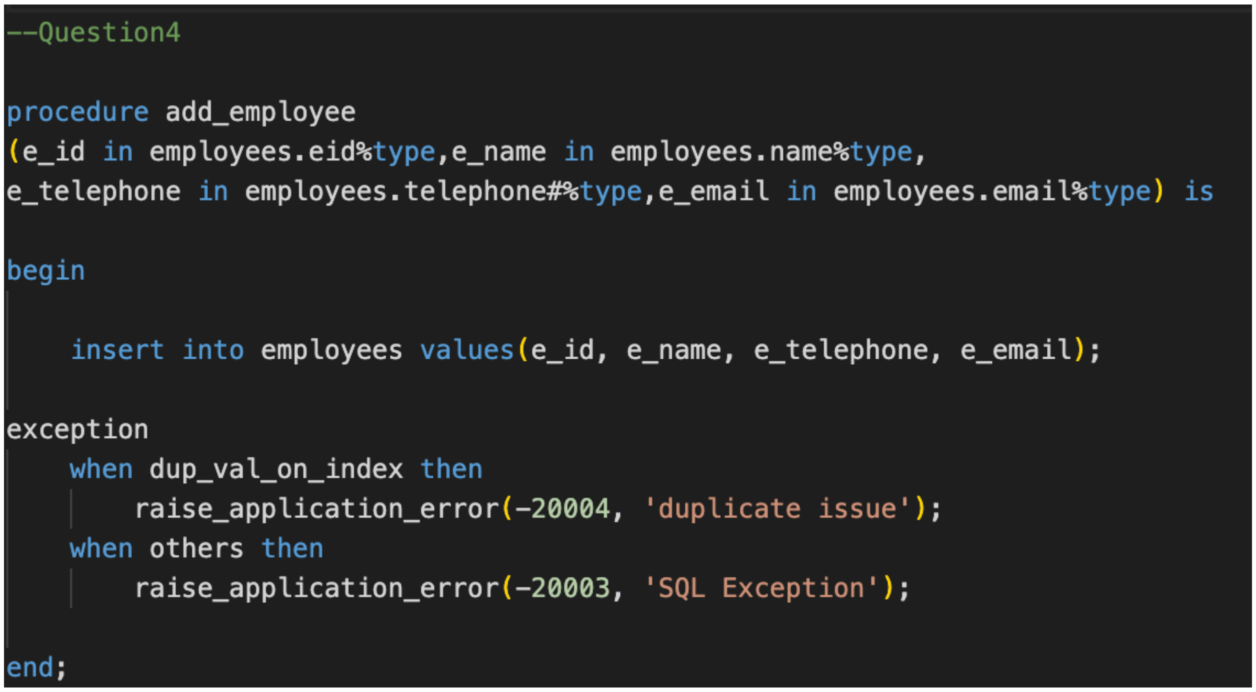


```
--Question3
procedure monthly_sale_activities
(employee_id in employees.eid%type, cursor_ref out cursor_out) is
eid_check number;
no_eid exception;
begin

    select count(*) into eid_check from employees where eid=employee_id;

    if eid_check=0 then
        raise no_eid;
    end if;

    open cursor_ref for
    select e.eid,name,
            to_char(pur_time, 'MON') as Month,to_char(pur_time, 'YYYY') as Year,
            count(pu.eid) as no_of_sales,sum(quantity) as qty,sum(payment) as total
    from employees e, purchases pu where e.eid = employee_id and e.eid = pu.eid
    group by e.eid, name, to_char(pur_time, 'MON'), to_char(pur_time, 'YYYY');

exception
    when no_eid then
        raise_application_error(-20001, 'eid not in DB');
    when no_data_found then
        raise_application_error(-20002, 'no data');
    when others then
        raise_application_error(-20003, 'SQL Exception');

end;
```

We have implemented the exception cases to check whether the e_id exists in employee table or not. If not, it just pops out as eid not in DB as exception.
If eid in the table and no sales has done by employee, it says no data and for all other errors as sql exception.

4) For the fourth question,

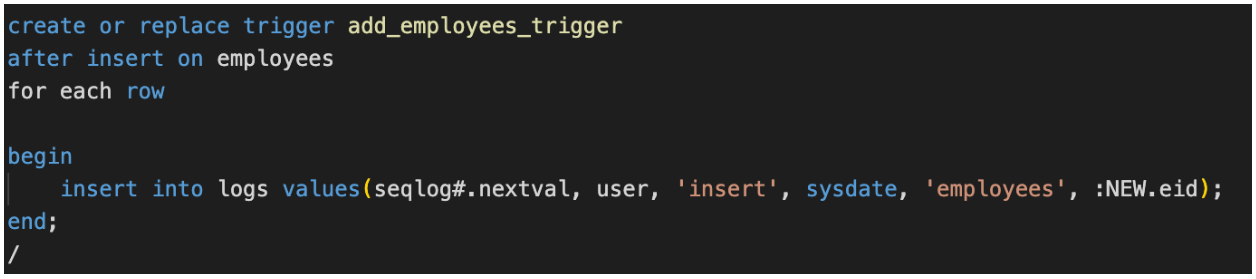
We have created a procedure **add_employee** to insert records into the employee table, this takes **e_id, e_name, e_telephone, e_email** as input.



```
--Question4

procedure add_employee
(e_id in employees.eid%type,e_name in employees.name%type,
e_telephone in employees.telephone#%type,e_email in employees.email%type) is

begin

    insert into employees values(e_id, e_name, e_telephone, e_email);

exception
    when dup_val_on_index then
        raise_application_error(-20004, 'duplicate issue');
    when others then
        raise_application_error(-20003, 'SQL Exception');

end;
```

It just inserts data, the exception part checks for the duplicate records on the eid column.

We have created a trigger(**add_employees_trigger**) whenever a new employee is added to the Employees table. It will insert a record into logs table which user has done the operation and what time.



```
create or replace trigger add_employees_trigger
after insert on employees
for each row

begin
    insert into logs values(seqlog#.nextval, user, 'insert', sysdate, 'employees', :NEW.eid);
end;
/
```

5) For the fifth question,

We have created a procedure to **add_purchases** to deal with the transactions. It inserts records into the purchases table. This takes **e_id, p_id, c_id, pur_qty, pur_unit_price** as input.

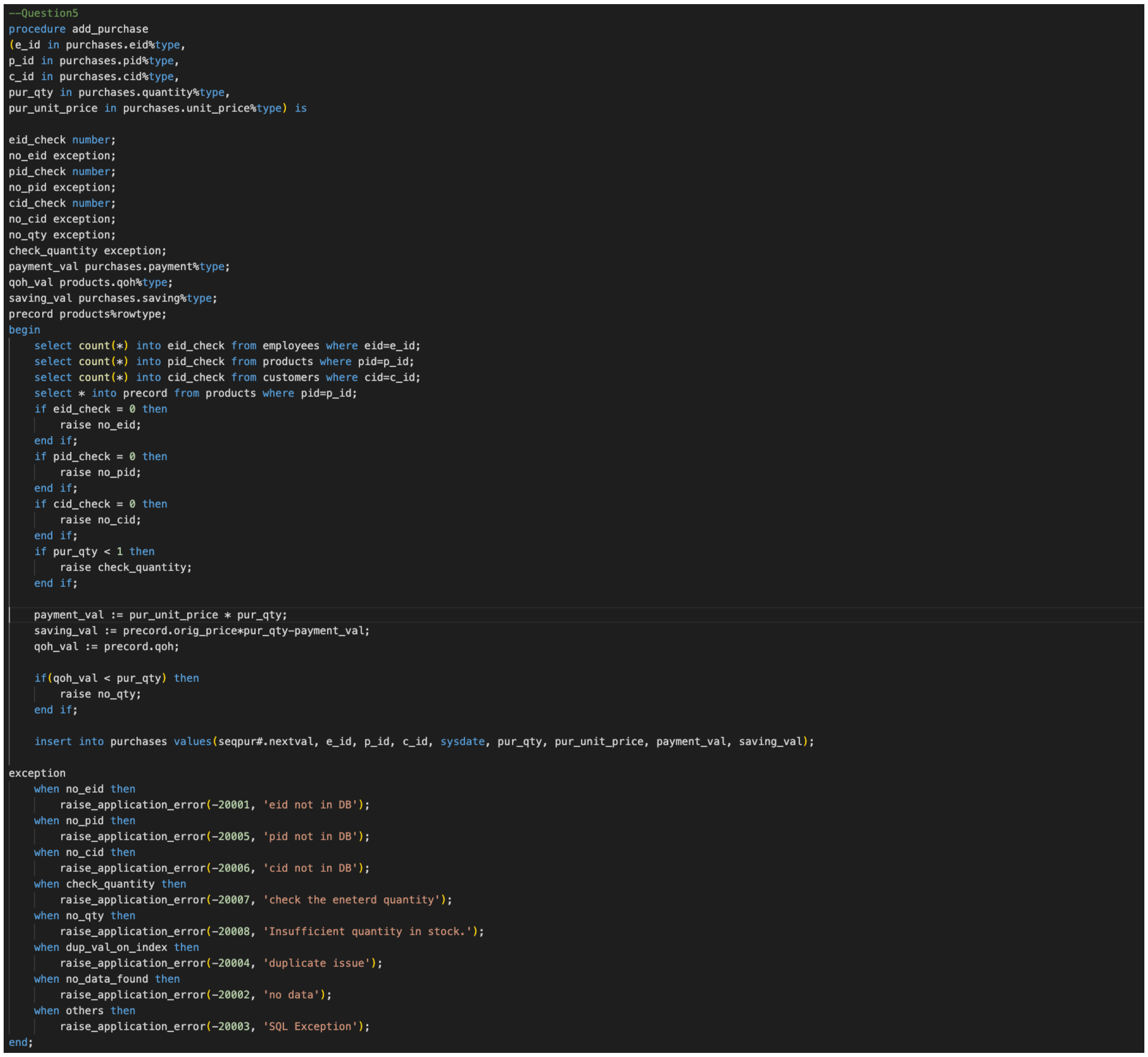
1. Pur# will be generated from the sequence seqpur#.
2. Payment is calculated from pur_qty and pur_unit_price, just multiplying both of them.
3. It also calculates the savings. First, we get the original_price of the product from the products table with p_id, next we multiply with the pur_qty and subract that with the payment.

4. Pur_time will be taken as default time when the record is inserted.

Before even inserting, we check whether we have enough quantity in products tables with the p_id.

If we have less quantity, we just raise Insufficient quantity in stock.

If we have enough quantity, we just proceed with inserting.



```
--Question5
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.quantity%type,
pur_unit_price in purchases.unit_price%type) is

eid_check number;
no_eid exception;
pid_check number;
no_pid exception;
cid_check number;
no_cid exception;
no_qty exception;
check_quantity exception;
payment_val purchases.payment%type;
qoh_val products.qoh%type;
saving_val purchases.saving%type;
precord products%rowtype;
begin
    select count(*) into eid_check from employees where eid=e_id;
    select count(*) into pid_check from products where pid=p_id;
    select count(*) into cid_check from customers where cid=c_id;
    select * into precord from products where pid=p_id;
    if eid_check = 0 then
        raise no_eid;
    end if;
    if pid_check = 0 then
        raise no_pid;
    end if;
    if cid_check = 0 then
        raise no_cid;
    end if;
    if pur_qty < 1 then
        raise check_quantity;
    end if;

    payment_val := pur_unit_price * pur_qty;
    saving_val := precord.orig_price*pur_qty-payment_val;
    qoh_val := precord.qoh;

    if(qoh_val < pur_qty) then
        raise no_qty;
    end if;

    insert into purchases values(seqpur#.nextval, e_id, p_id, c_id, sysdate, pur_qty, pur_unit_price, payment_val, saving_val);

exception
    when no_eid then
        raise_application_error(-20001, 'eid not in DB');
    when no_pid then
        raise_application_error(-20005, 'pid not in DB');
    when no_cid then
        raise_application_error(-20006, 'cid not in DB');
    when check_quantity then
        raise_application_error(-20007, 'check the eneterd quantity');
    when no_qty then
        raise_application_error(-20008, 'Insufficient quantity in stock.');
    when dup_val_on_index then
        raise_application_error(-20004, 'duplicate issue');
    when no_data_found then
        raise_application_error(-20002, 'no data');
    when others then
        raise_application_error(-20003, 'SQL Exception');
end;
```

The necessary exceptions are added to check whether the e_id, p_id and c_id present in the employee, products and customers table respectively.

If zero or negative values are entered in pur_qty, it will ask to recheck the entered pur_qty.

We have created a trigger(**add_purchases_trigger**) that implements the requirements after inserting the purchase, it will check if the new quantity(after subtracting the pur_qty from the products qoh) of product is going to be less than or greater than the qoh_threshold, if it is greater it just updates the qoh value in products table or else, it will print the message “The current qoh of the product is below the required threshold and new supply is required” and updates the qoh value to adding 20 with qoh_threshold and finally it prints the new qoh value of the product.

It also updates the last_visit_made and visits_made in the customer table for the particular cid.

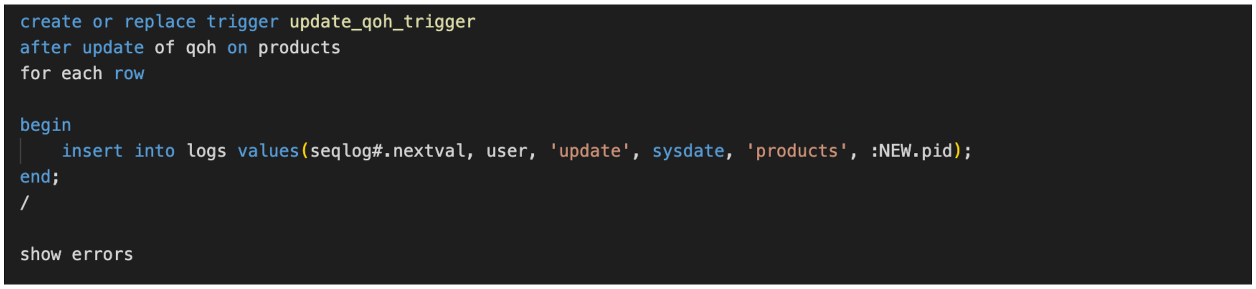
Finally, it inserts a record to logs as which user made the operation and the time it is made and p_key of purchases.

6) For the sixth question,

We have created two more triggers **update_last_visit_date_trigger** and **update_vists_made_trigger** which basically looks at the respective column in customers table and when the update happens on the particular column, it will insert a record into logs as which user made the operation and what time it is made and p_key of customers.

```
create or replace trigger update_last_visit_date_trigger
after update of last_visit_date on customers
for each row

begin
    insert into logs values(seqlog#.nextval, user, 'update', sysdate, 'customers', :NEW.cid);
end;
/

create or replace trigger update_visits_made_trigger
after update of visits_made on customers
for each row

begin
    insert into logs values(seqlog#.nextval, user, 'update', sysdate, 'customers', :NEW.cid);
end;
/
```

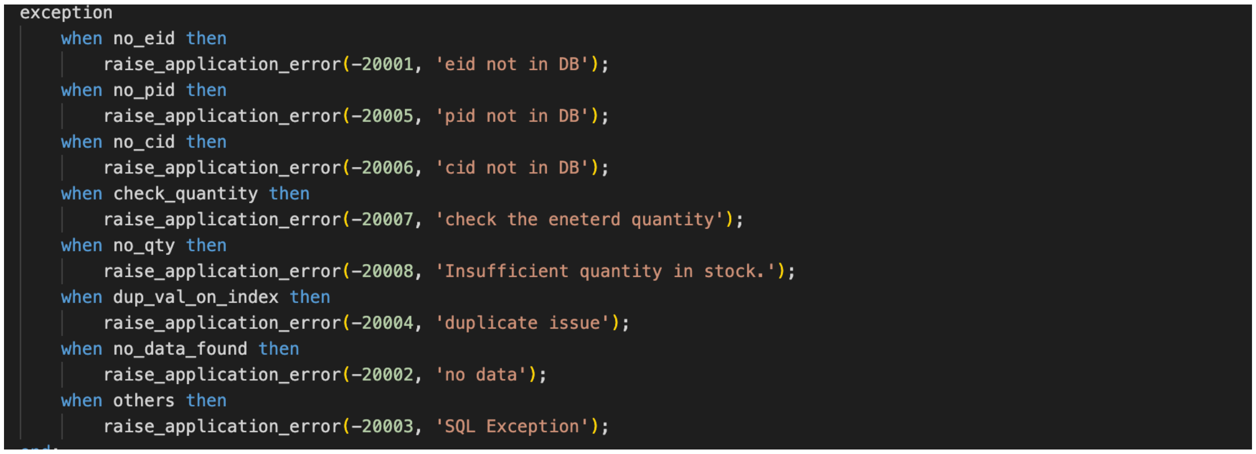
The last trigger **update_qoh_trigger** looks at the qoh column in products. Whenever an update happens on that column, it will insert a record into logs specifying which user made the operation and what time it is made and p_key of products.



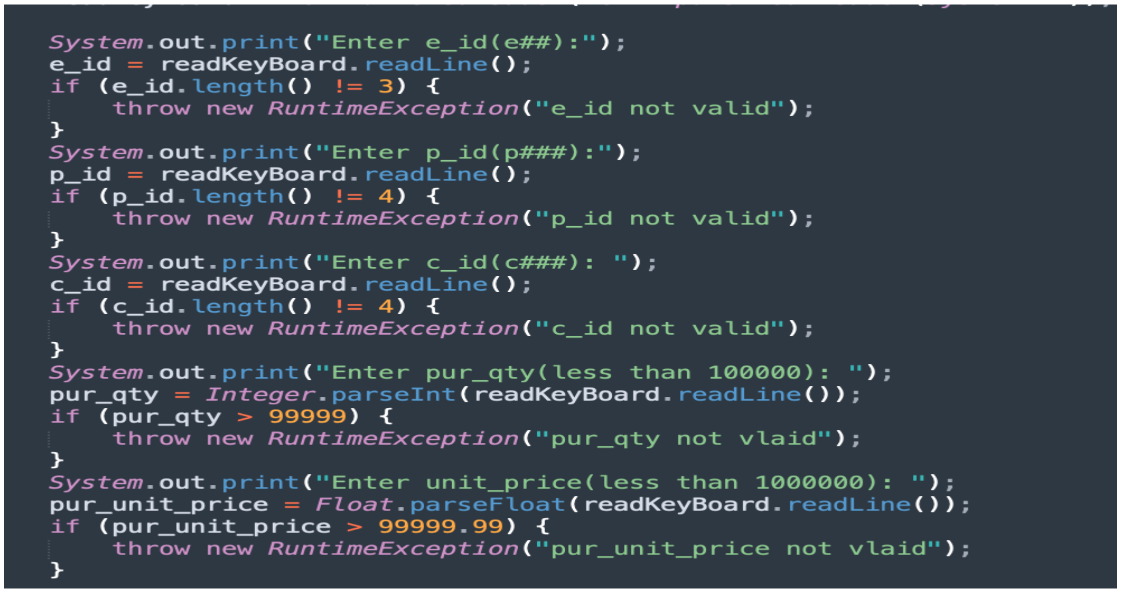
```
create or replace trigger update_qoh_trigger
after update of qoh on products
for each row

begin
    insert into logs values(seqlog#.nextval, user, 'update', sysdate, 'products', :NEW.pid);
end;
/

show errors
```

7) For the seventh question,

We have added exceptions in each procedure to show errors. Whenever the wrong data comes in, it will populate specific exceptions and cases such as whenever the user gives the wrong length of data etc.., are taken care of in the java program.



```
exception
    when no_eid then
        raise_application_error(-20001, 'eid not in DB');
    when no_pid then
        raise_application_error(-20005, 'pid not in DB');
    when no_cid then
        raise_application_error(-20006, 'cid not in DB');
    when check_quantity then
        raise_application_error(-20007, 'check the eneterd quantity');
    when no_qty then
        raise_application_error(-20008, 'Insufficient quantity in stock.');
    when dup_val_on_index then
        raise_application_error(-20004, 'duplicate issue');
    when no_data_found then
        raise_application_error(-20002, 'no data');
    when others then
        raise_application_error(-20003, 'SQL Exception');
```



```
System.out.print("Enter e_id(e##):");
e_id = readKeyBoard.readLine();
if (e_id.length() != 3) {
    throw new RuntimeException("e_id not valid");
}
System.out.print("Enter p_id(p###):");
p_id = readKeyBoard.readLine();
if (p_id.length() != 4) {
    throw new RuntimeException("p_id not valid");
}
System.out.print("Enter c_id(c###): ");
c_id = readKeyBoard.readLine();
if (c_id.length() != 4) {
    throw new RuntimeException("c_id not valid");
}
System.out.print("Enter pur_qty(less than 100000): ");
pur_qty = Integer.parseInt(readKeyBoard.readLine());
if (pur_qty > 99999) {
    throw new RuntimeException("pur_qty not vlaid");
}
System.out.print("Enter unit_price(less than 1000000): ");
pur_unit_price = Float.parseFloat(readKeyBoard.readLine());
if (pur_unit_price > 99999.99) {
    throw new RuntimeException("pur_unit_price not vlaid");
}
```

Collaboration Report:

Meetings: We had our first meeting regarding the project on 31st March. During our first meeting, we briefly discussed how to implement the java code and GUI. We also divided the questions among ourselves. Akhil was in-charge of questions 1,2,3,6 and Harsha was in-charge of questions 4,5,7.

- After our first meeting, we met every week on Thursdays at 8PM to discuss our progress and solve the issues we had.
- At our second meeting (April 6), we discussed the challenges in the JDBC part and shared suggestions with each other.
- At our Third meeting (April 13), we discussed the exception handling part and ensured that all the functions and triggers are working as expected.
- At our fourth meeting (April 20), we tested our code end to end and made few changes about printing trigger messages and planned the demo and documentation part.

Plans: We planned to complete the project within 4 weeks, and we managed to do so successfully. At each meeting, we both reviewed each other's progress and updated plans accordingly.

Responsibilities: Harsha focused on creating the add_employees, add_purchase procedures, insert_purchase_trigger and the menu-driven interface using Java and JDBC. Akhil worked on creating the show_* procedures, the monthly_sale_activities procedure, and the trigger for updating qoh, customer_last_visit_date and visits_made . However, we both reviewed and contributed to each other's work.

Self-assessment: We worked really well together as a team. We communicated clearly and helped each other out when needed.