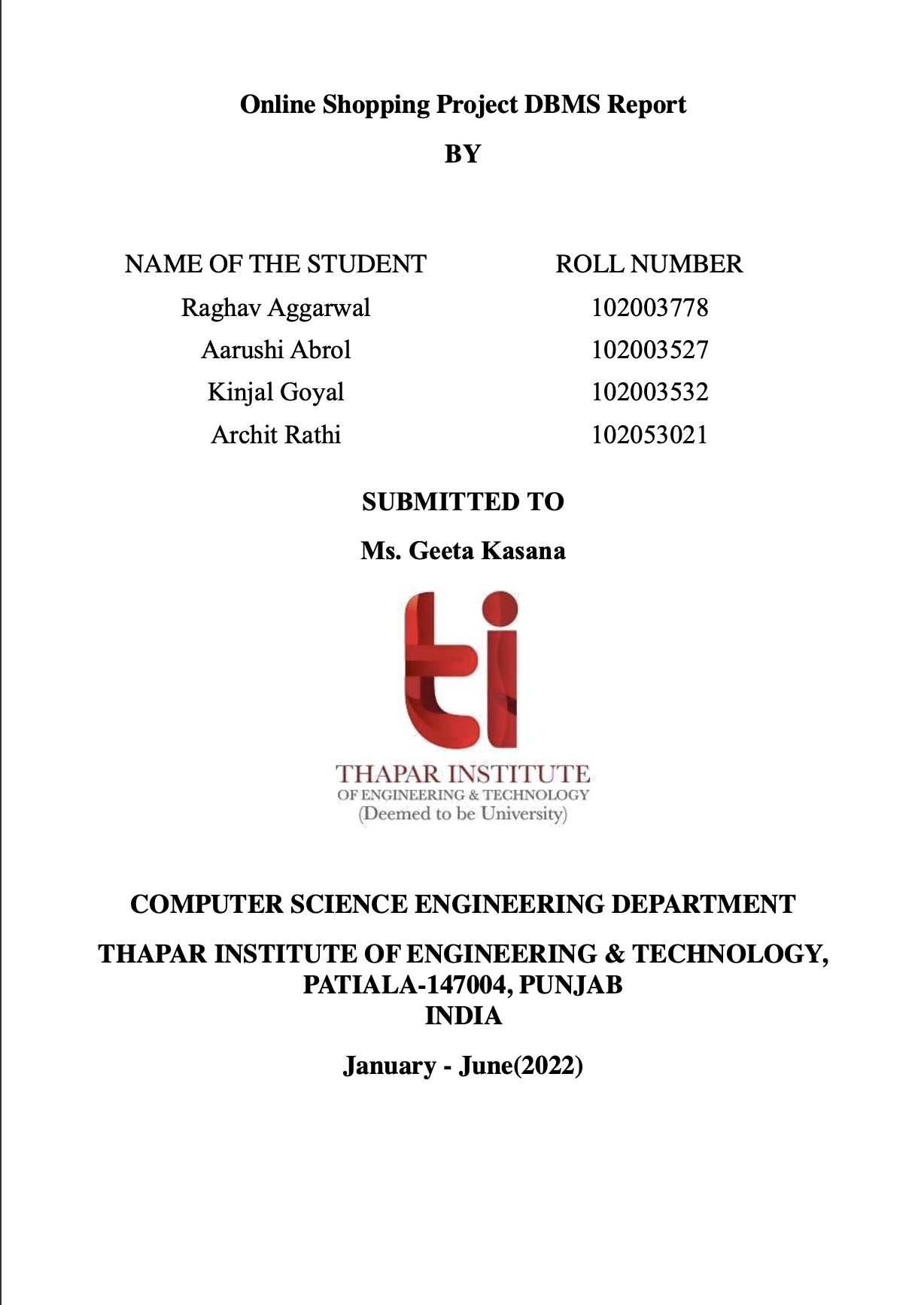
****

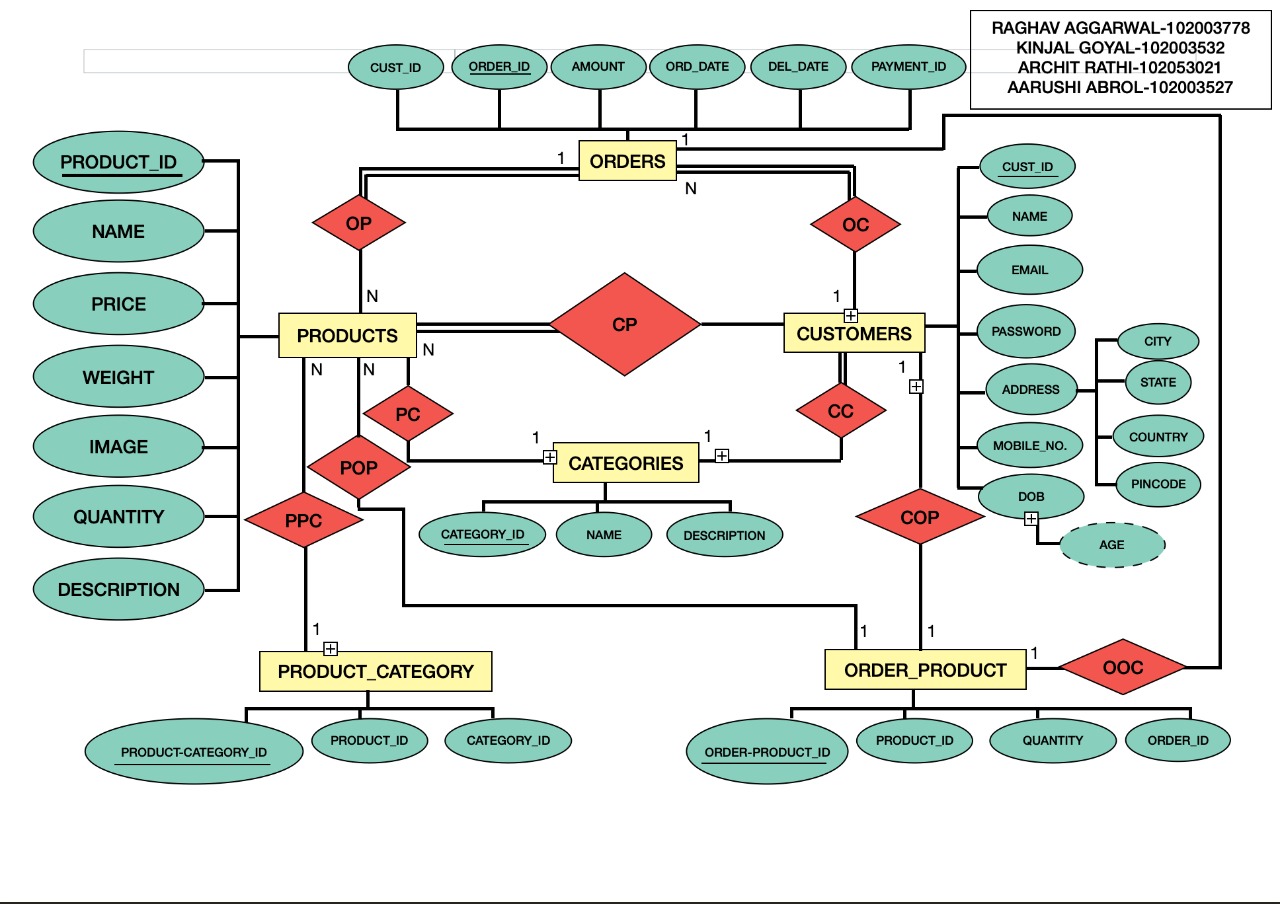
**Index**

|  |  |  |
| --- | --- | --- |
| **Sr. no** | **Title** | **Page No** |
| **1.** | **Problem Statement** | **3** |
| **2.** | **E.R. Diagram** | **4** |
| **3.** | **E.R. to Table** | **5** |
| **4.** | **Normalization** | **6-9** |
| **5.** | **SQL/PLSQL Code Implementation** | **10-14** |
| **6.** | **Output** | **15-22** |

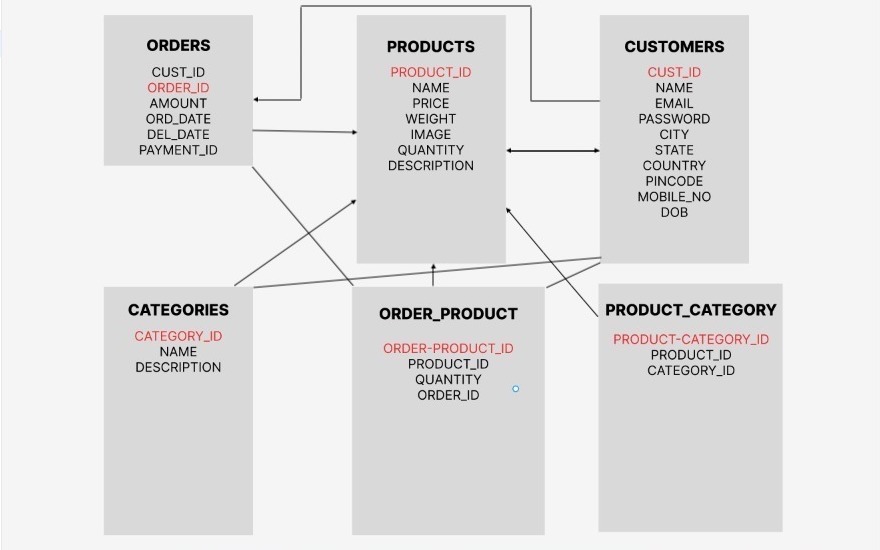
**PROBLEM STATEMENT**

In this modern era of shopping, no seller wants to be left behind, moreover due to its simplicity the shift from an offline selling model to an online selling model is witnessing rampant growth. Therefore, as an engineer, our job is to ease the path of this transition for the seller. Amongst many things that an online site requires the most important is a database system. Hence in this project, we are planning to design a database where we can manage orders placed by various customers in a Shopping Complex.

**ER DIAGRAM**

****

**ER TO TABLE**

****

**NORMALIZATION**

**1NF**

A relation is in its first normal form if every attribute in that relation is singled valued attribute.

**2NF**

A relation must be in first normal form and the relation must not contain any partial dependency i.e., no non-prime attribute (attributes that are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

**3NF**

A relation is in the third normal form if there is no transitive dependency for non-prime attributes as well as it is in the second normal form.

A relation is in 3NF if at least one of the following conditions holds in every non-trivial function dependency X –> Y

X is a super key.

Y is a prime attribute (each element of Y is part of some candidate key).

**4NF**

The fourth normal form (4NF) is a level of database normalization where there are no non-trivial multivalued dependencies other than a candidate key. It builds on the first three normal forms (1NF, 2NF, and 3NF) and the Boyce-Codd Normal Form (BCNF). It states that, in addition to a database meeting the requirements of BCNF, it must not contain more than one multivalued dependency.

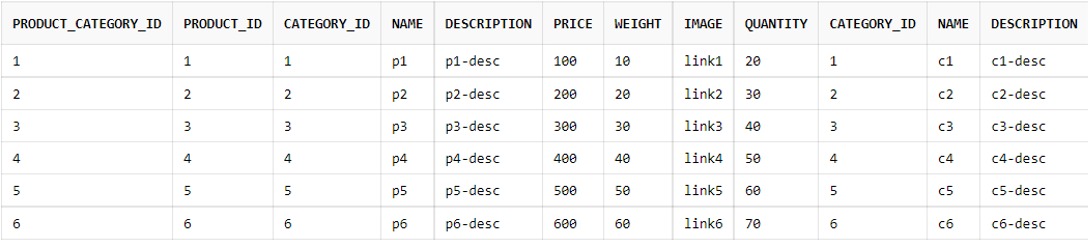
**BCNF**

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF if in every non-trivial functional dependency X –> Y, X is a super key.

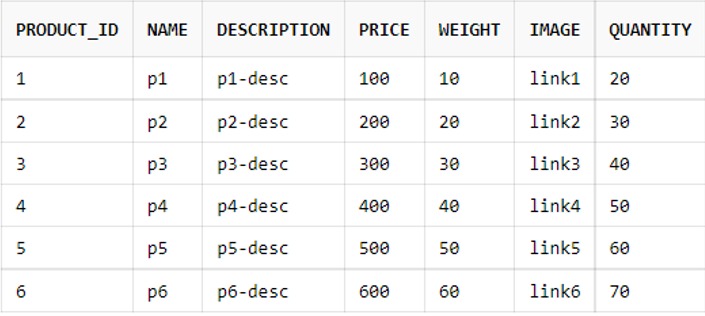
After following these rules, following are the normalized tables obtained:

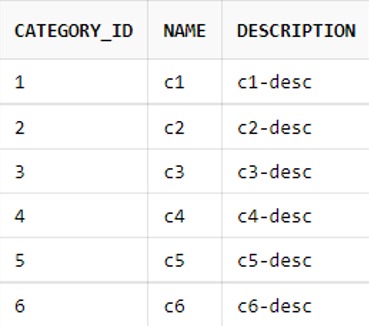
1. Products Table

From this table,

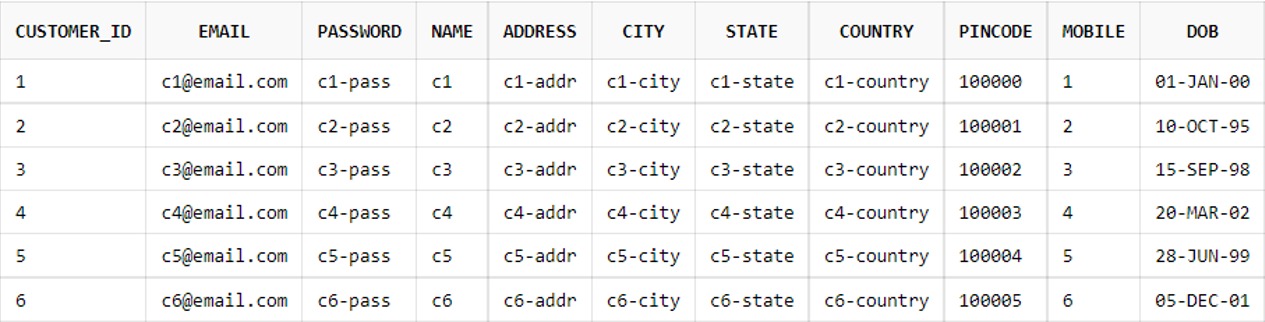


we obtained the following tables after normalization:





1. Costumer Table

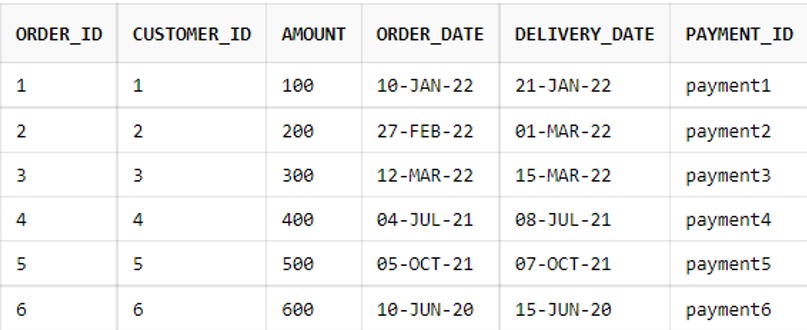


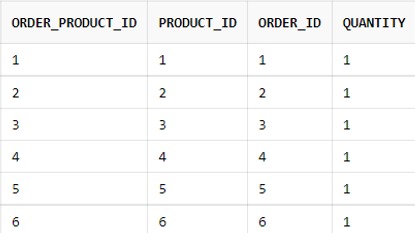
1. Orders Table

From this table,



we obtained the following tables after normalization:





**SQL IMPLEMENTATION**

CREATION OF TABLES

CREATE TABLE products (product\_id INT PRIMARY KEY, name VARCHAR2(200), description VARCHAR2(400), price INT, weight INT, image VARCHAR2(200), quantity INT);

CREATE TABLE categories (category\_id INT PRIMARY KEY, name VARCHAR2(200), description VARCHAR2(200));

CREATE TABLE customers (customer\_id INT PRIMARY KEY, email VARCHAR2(200), password VARCHAR2(200), name VARCHAR2(200), address VARCHAR2(200), city VARCHAR2(200), state VARCHAR2(200), country VARCHAR2(200), pincode INT, mobile VARCHAR2(40), DOB DATE);

CREATE TABLE orders (order\_id INT PRIMARY KEY, customer\_id INT REFERENCES customers(customer\_id), amount INT, order\_date DATE, delivery\_date DATE, payment\_id VARCHAR2(200));

CREATE TABLE product\_category (product\_category\_id INT PRIMARY KEY, product\_id INT REFERENCES products(product\_id), category\_id INT REFERENCES categories(category\_id));

CREATE TABLE order\_product (order\_product\_id INT PRIMARY KEY, product\_id INT REFERENCES products(product\_id), order\_id INT REFERENCES orders(order\_id), quantity INT);

INSERTION OF VALUES

INSERT INTO products VALUES (1, 'p1', 'p1-desc', 100, 10, 'link1', 20);

INSERT INTO products VALUES (2, 'p2', 'p2-desc', 200, 20, 'link2', 30);

INSERT INTO products VALUES (3, 'p3', 'p3-desc', 300, 30, 'link3', 40);

INSERT INTO products VALUES (4, 'p4', 'p4-desc', 400, 40, 'link4', 50);

INSERT INTO products VALUES (5, 'p5', 'p5-desc', 500, 50, 'link5', 60);

INSERT INTO products VALUES (6, 'p6', 'p6-desc', 600, 60, 'link6', 70);

SELECT \* FROM products; //Just to show the table with attributes and values

INSERT INTO categories VALUES (1, 'c1', 'c1-desc');

INSERT INTO categories VALUES (2, 'c2', 'c2-desc');

INSERT INTO categories VALUES (3, 'c3', 'c3-desc');

INSERT INTO categories VALUES (4, 'c4', 'c4-desc');

INSERT INTO categories VALUES (5, 'c5', 'c5-desc');

INSERT INTO categories VALUES (6, 'c6', 'c6-desc');

SELECT \* FROM categories; //Just to show table with attributes and values

INSERT INTO customers VALUES(1, 'c1@email.com', 'c1-pass', 'c1', 'c1-addr', 'c1-city', 'c1-state', 'c1-country', '100000', '1','01-JAN-2000');

INSERT INTO customers VALUES(2, 'c2@email.com', 'c2-pass', 'c2', 'c2-addr', 'c2-city', 'c2-state', 'c2-country','100001', '2', '10-OCT-1995');

INSERT INTO customers VALUES(3, 'c3@email.com', 'c3-pass', 'c3', 'c3-addr', 'c3-city', 'c3-state', 'c3-country','100002', '3', '15-SEP-1998');

INSERT INTO customers VALUES(4, 'c4@email.com', 'c4-pass', 'c4', 'c4-addr', 'c4-city', 'c4-state', 'c4-country','100003', '4', '20-MAR-2002');

INSERT INTO customers VALUES(5, 'c5@email.com', 'c5-pass', 'c5', 'c5-addr', 'c5-city', 'c5-state', 'c5-country','100004', '5', '28-JUN-1999');

INSERT INTO customers VALUES(6, 'c6@email.com', 'c6-pass', 'c6', 'c6-addr', 'c6-city', 'c6-state', 'c6-country','100005', '6', '05-DEC-2001');

SELECT \* FROM customers; //Just to show table with attributes and values

INSERT INTO orders(order\_id, customer\_id, amount, order\_date, delivery\_date, payment\_id) VALUES (1, 1, 100,'10-JAN-2022','21-JAN-2022', 'payment1');

INSERT INTO orders(order\_id, customer\_id, amount, order\_date, delivery\_date, payment\_id) VALUES (2, 2, 200,'27-FEB-2022','01-MAR-2022', 'payment2');

INSERT INTO orders(order\_id, customer\_id, amount, order\_date, delivery\_date, payment\_id) VALUES (3, 3, 300,'12-MAR-2022','15-MAR-2022', 'payment3');

INSERT INTO orders(order\_id, customer\_id, amount, order\_date, delivery\_date, payment\_id) VALUES (4, 4, 400,'04-JUL-2021','08-JUL-2021', 'payment4');

INSERT INTO orders(order\_id, customer\_id, amount, order\_date, delivery\_date, payment\_id) VALUES (5, 5, 500,'05-OCT-2021','07-OCT-2021', 'payment5');

INSERT INTO orders(order\_id, customer\_id, amount, order\_date, delivery\_date, payment\_id) VALUES (6, 6, 600,'10-JUN-2020','15-JUN-2020', 'payment6');

SELECT \* FROM orders; //Just to show the table with attributes and values

INSERT INTO product\_category VALUES (1, 1, 1);

INSERT INTO product\_category VALUES (2, 2, 2);

INSERT INTO product\_category VALUES (3, 3, 3);

INSERT INTO product\_category VALUES (4, 4, 4);

INSERT INTO product\_category VALUES (5, 5, 5);

INSERT INTO product\_category VALUES (6, 6, 6);

SELECT \* FROM product\_category

// If you wish to get details of entire products information including categories use the following command:

SELECT \* FROM product\_category pc JOIN products p ON pc.product\_id=p.product\_id JOIN categories c ON pc.category\_id=c.category\_id;

INSERT INTO order\_product VALUES (1, 1, 1, 1);

INSERT INTO order\_product VALUES (2, 2, 2, 1);

INSERT INTO order\_product VALUES (3, 3, 3, 1);

INSERT INTO order\_product VALUES (4, 4, 4, 1);

INSERT INTO order\_product VALUES (5, 5, 5, 1);

INSERT INTO order\_product VALUES (6, 6, 6, 1);

SELECT \* FROM order\_product

// If you wish to get details of entire orders information including categories use the following command:

SELECT \* FROM order\_product op JOIN orders o ON o.order\_id=op.order\_id JOIN products p ON op.product\_id=p.product\_id;

BASIC QUERIES

**Total Number of Products Sold**

SELECT SUM(quantity) FROM order\_product;

**Total Number of a specific Products Sold**

SELECT SUM(quantity) FROM order\_product WHERE product\_id=(SELECT product\_id FROM products WHERE name='p1');

**Total cost earned by specific product**

SELECT SUM(op.quantity\*p.price) FROM order\_product op JOIN products p ON op.product\_id=p.product\_id WHERE p.name='p4';

**Customers Buying More than a specific amount in a single order**

SELECT \* FROM customers c JOIN orders o ON c.customer\_id=o.customer\_id where ocu.amount > 300;

**Users who spent greater than some amount**

SELECT o.customer\_id FROM customers c JOIN orders o ON c.customer\_id=o.customer\_id GROUP BY c.customer\_id HAVING SUM(o.amount) > 300;

**Total Amount spent by each user**

SELECT o.customer\_id,SUM(o.amount) FROM customers c JOIN orders o ON c.customer\_id=o.customer\_id GROUP BY c.customer\_id;

**PL/SQL IMPLEMENTATION**

**To print total no. of orders**

create or replace procedure tot\_orders

is

quan number(5);

begin

select count(\*) into quan from orders;

dbms\_output.put\_line('total ' || quan);

end;

**To print total ammount of value present in your orders**

create or replace procedure tot\_cost

is

costs number(5);

begin

select sum(amount) into costs from orders;

dbms\_output.put\_line('total ' || costs);

end;

**Procedure to automatically apply discount based on order value**

create or replace procedure discount

is

costs number(5);

begin

select sum(amount) into costs from orders;

if (costs<=500) then

costs := costs-0;

elsif (costs>500 and costs <=1000) then

costs := 0.9\*costs;

else

costs := 0.8\*costs;

end if;

dbms\_output.put\_line('Total cost after discount ' || costs);

end;

**This takes coupon code to apply discount based on coupon code**

create or replace procedure coupon(code in number)

is

costs number(5);

coup\_10 number(9):=1032;

coup\_5 number(9):=8532;

begin

select sum(amount) into costs from orders;

if (code = coup\_10) then

costs:=costs\*0.9;

dbms\_output.put\_line('Discount applied: '||10||' percent');

elsif (code = coup\_5) then

costs:=costs\*0.95;

dbms\_output.put\_line('Discount applied: '||5||' percent');

end if;

dbms\_output.put\_line('Total checkout value '||costs);

end;

**To filter products based on cost**

create or replace procedure cost\_filter(c in number,t in varchar)

is

cs products.price%type;

ty products.name%type;

id products.product\_id%type;

cursor cf is

select product\_id,price,name from product where price<c;

begin

open cf;

loop

fetch cf into id,cs,ty;

exit when cf%notfound;

dbms\_output.put\_line('Product' || id || 'has cost ' || cs || ' and the product is ' || ty);

end loop;

close cf;

exception

when no\_data\_found then

dbms\_output.put\_line('Sorry no such products exist');

end;

**Trigger**

**If any product is present on your order’s page, then it will display a proper notification for your discounted price**

CREATE OR REPLACE TRIGGER display\_reduced\_price

BEFORE UPDATE ON orders

FOR EACH ROW

WHEN (NEW.order\_id > 0)

DECLARE

reduction number;

BEGIN

reduction := :OLD.amount - :NEW.amount;

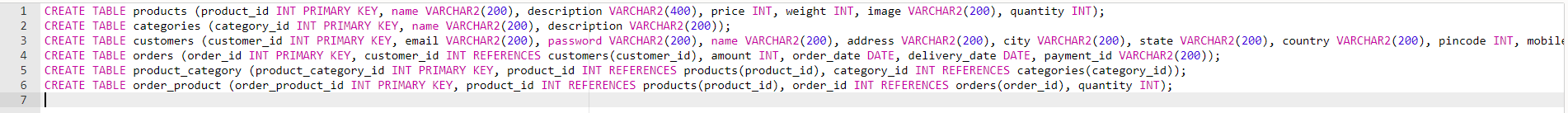
if (reduction>0) then

dbms\_output.put\_line('Congratulations , You have just saved RS. ' || reduction);

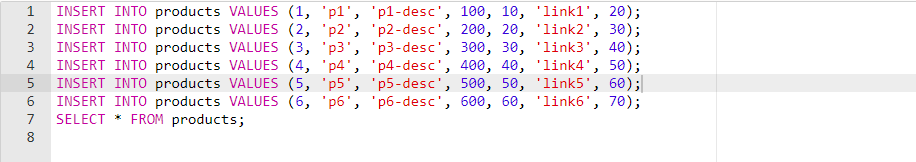
END IF;

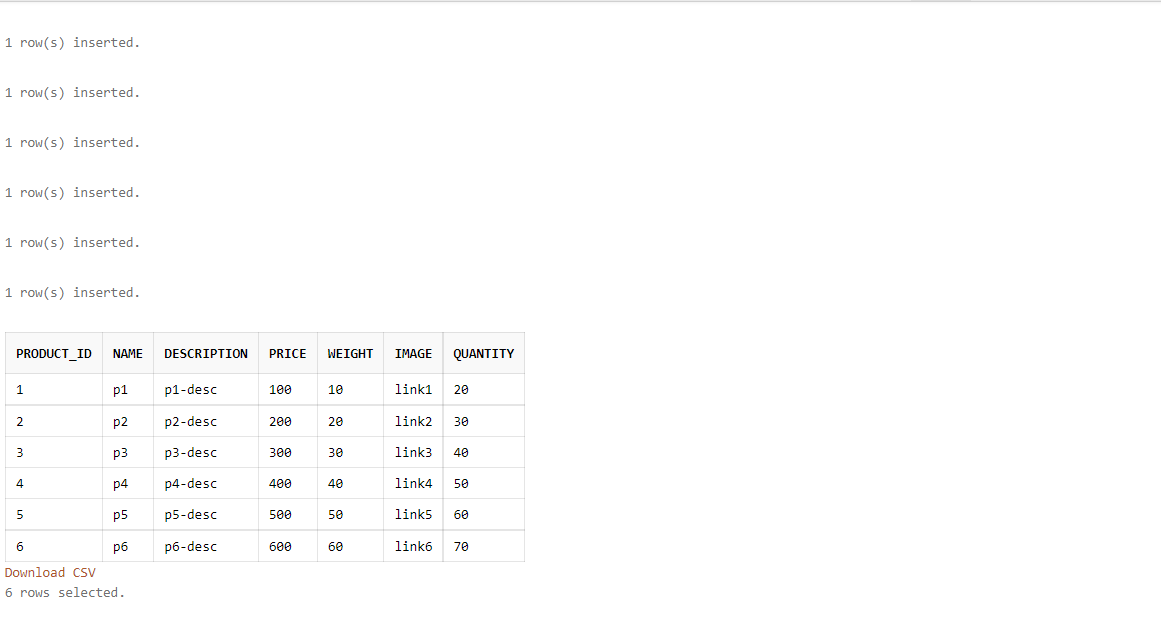
END;

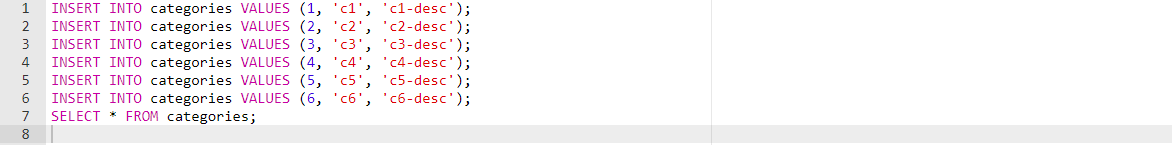
**OUTPUT SCREENSHOTS**

****

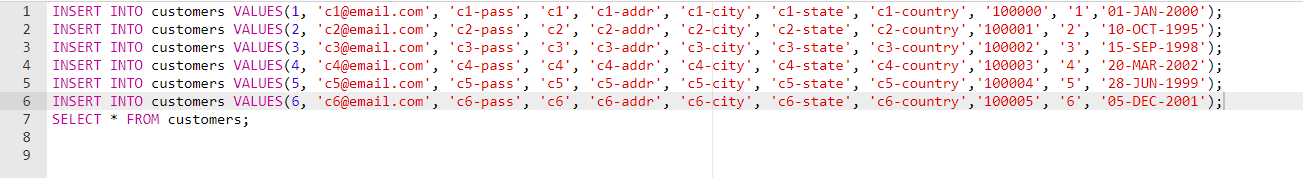
****

****

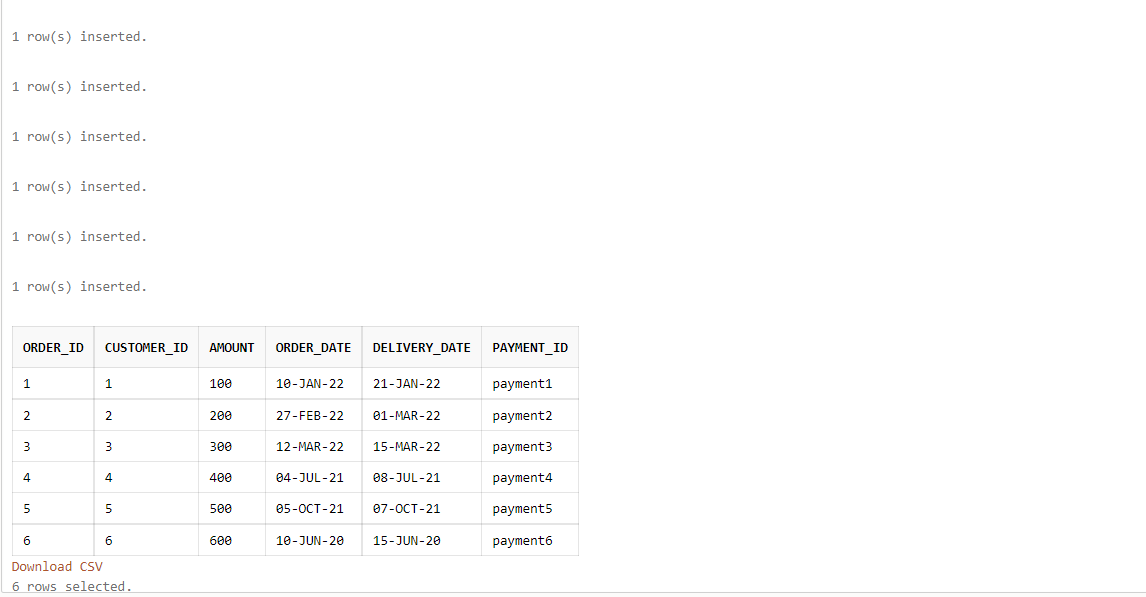
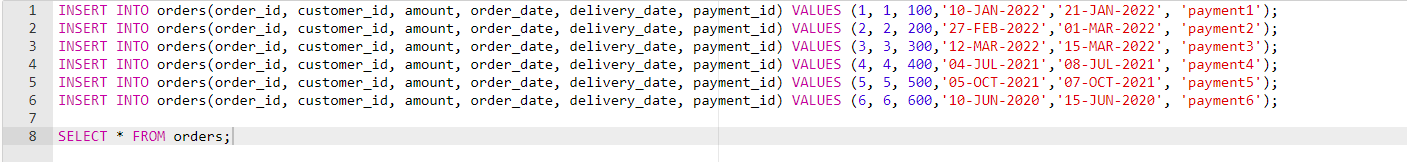
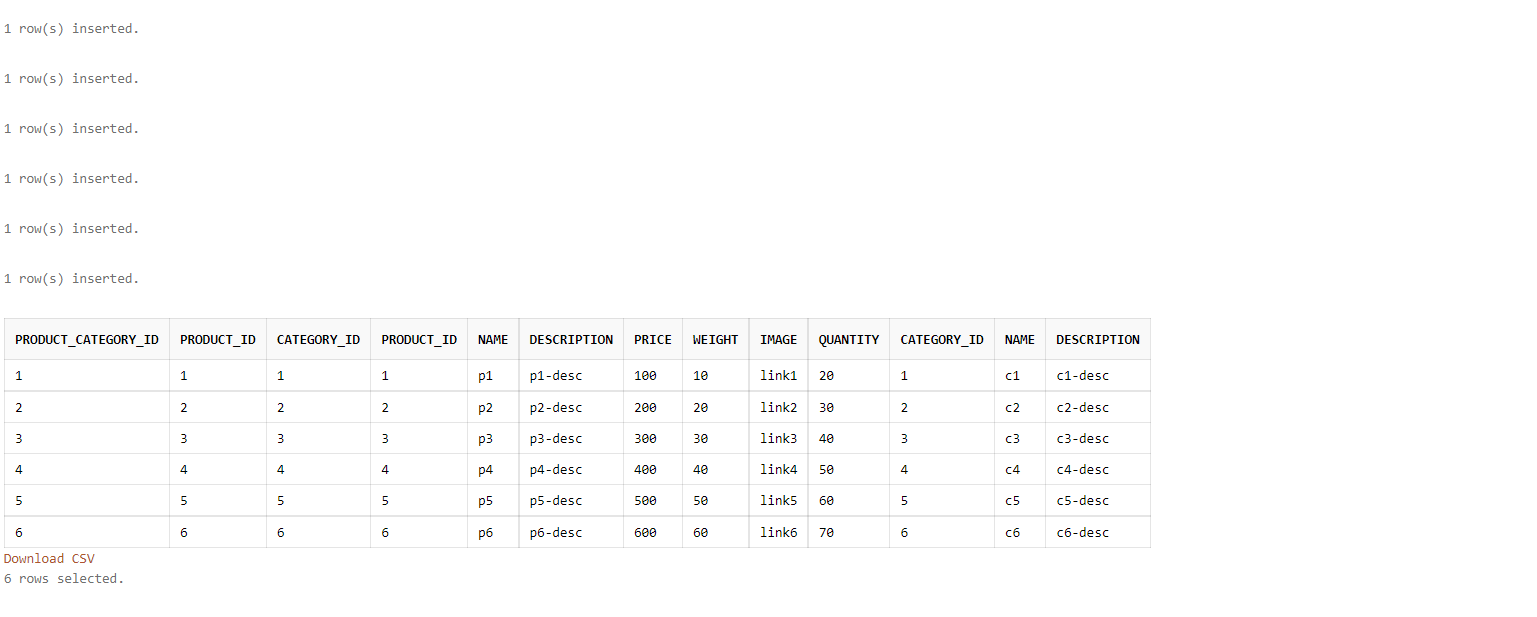
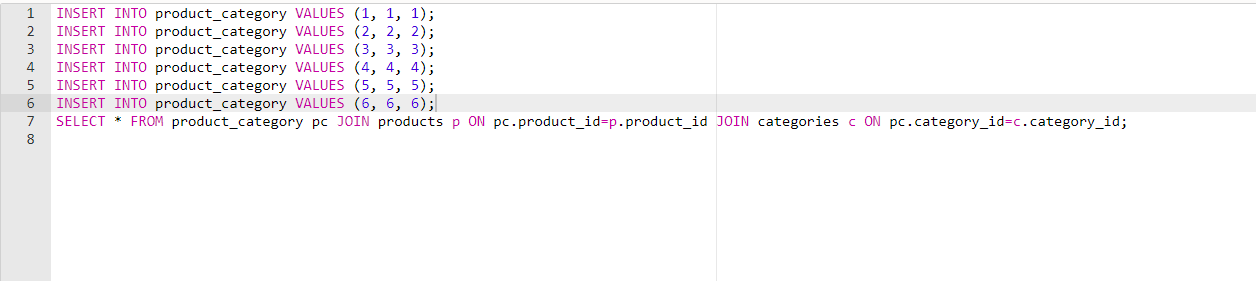
****

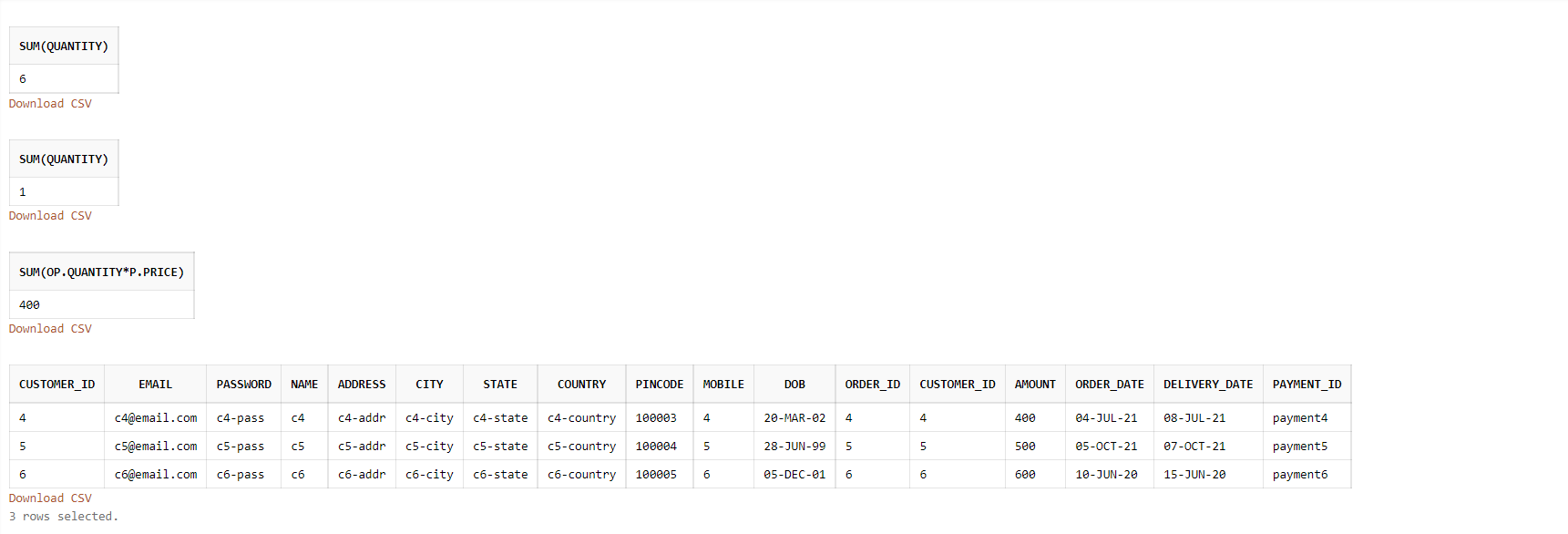
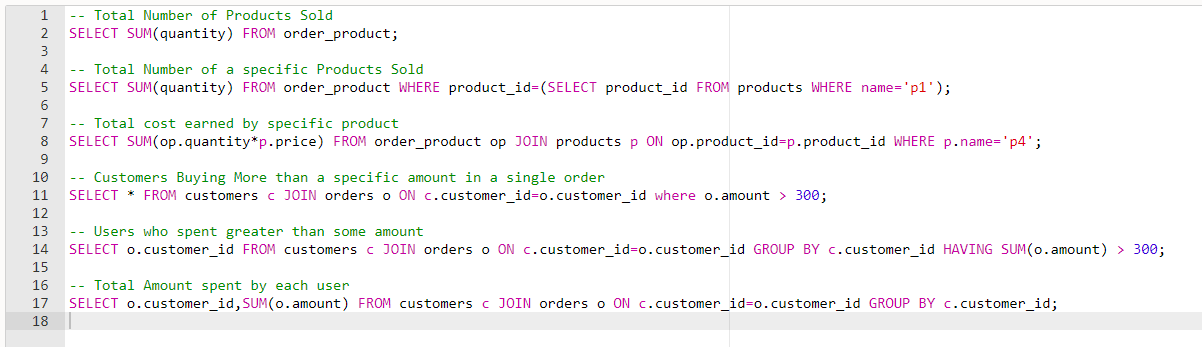
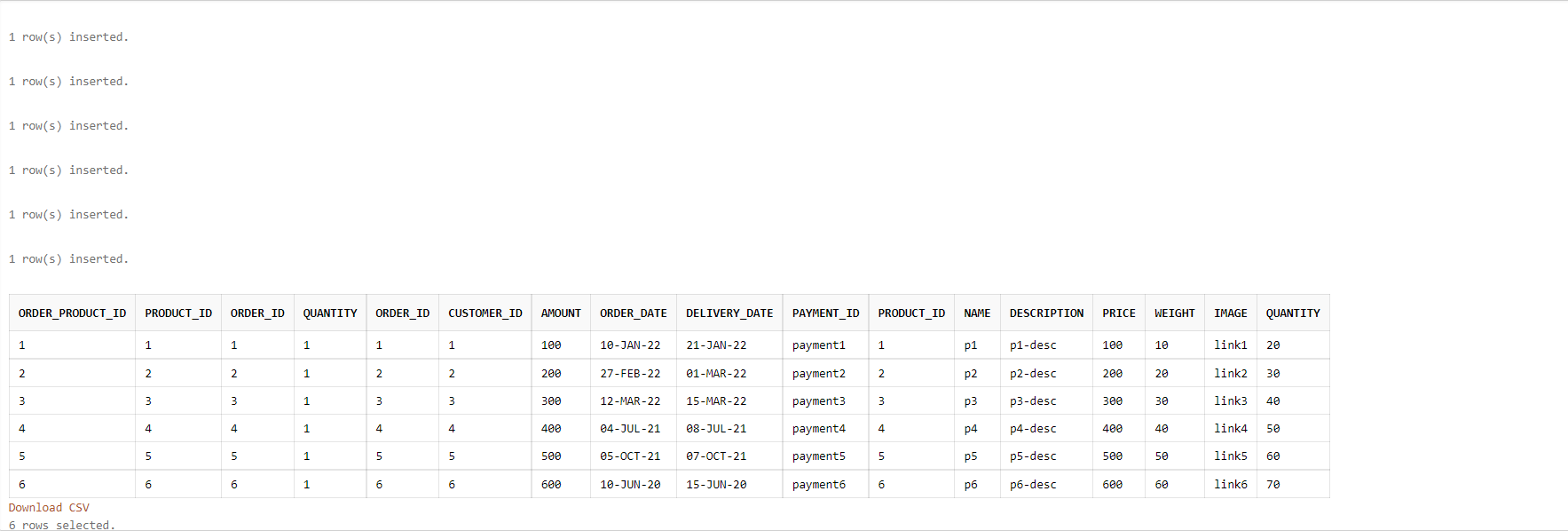
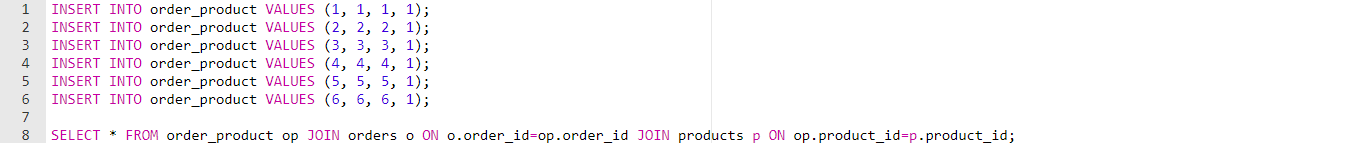
****

****

****

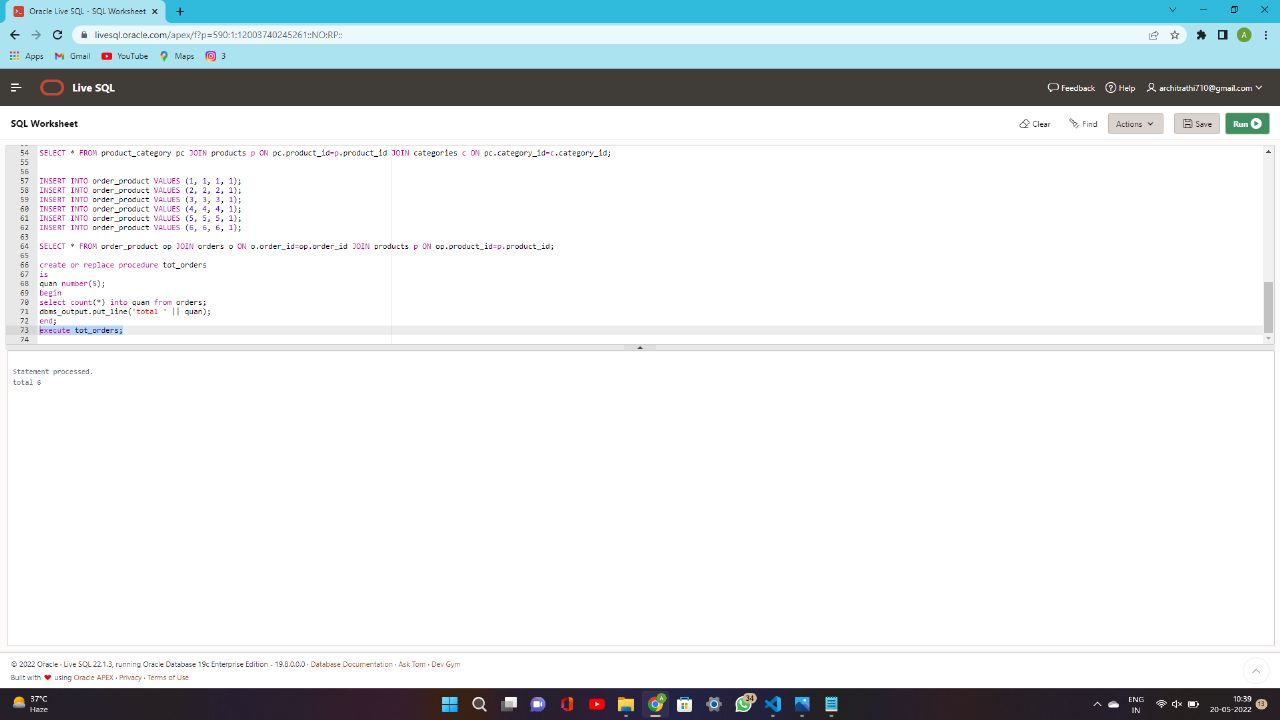
****

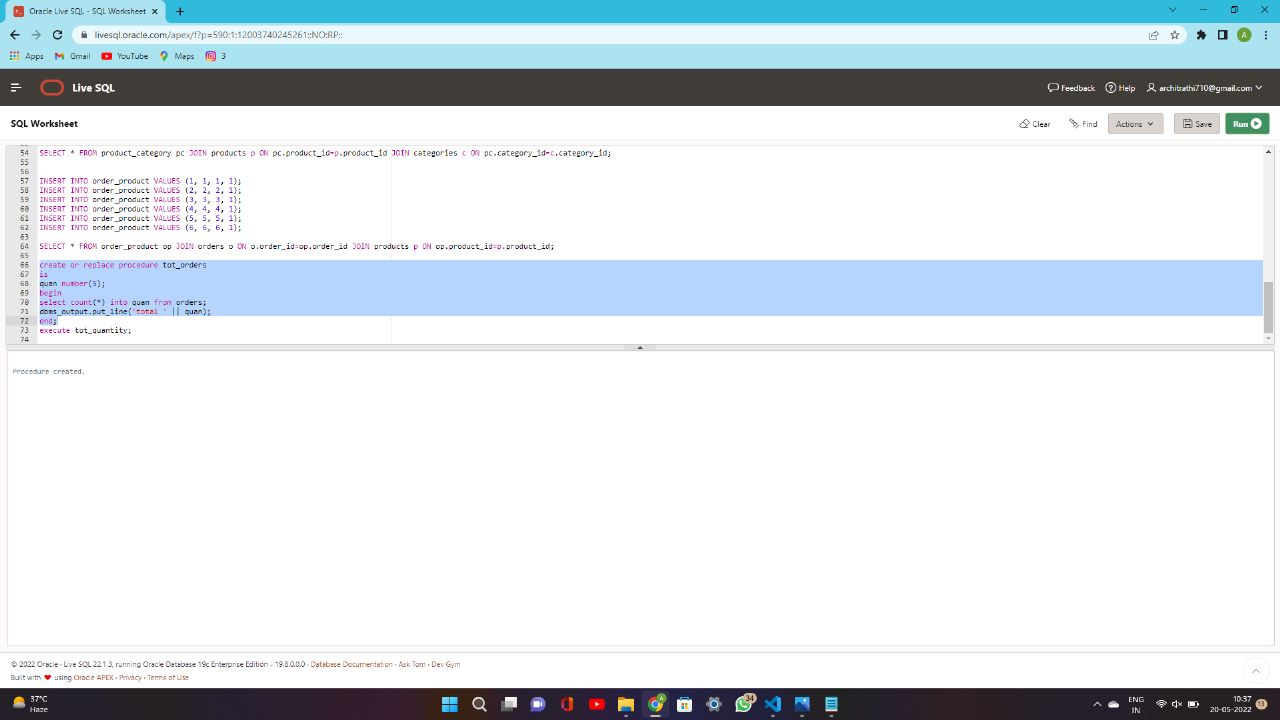
****

****

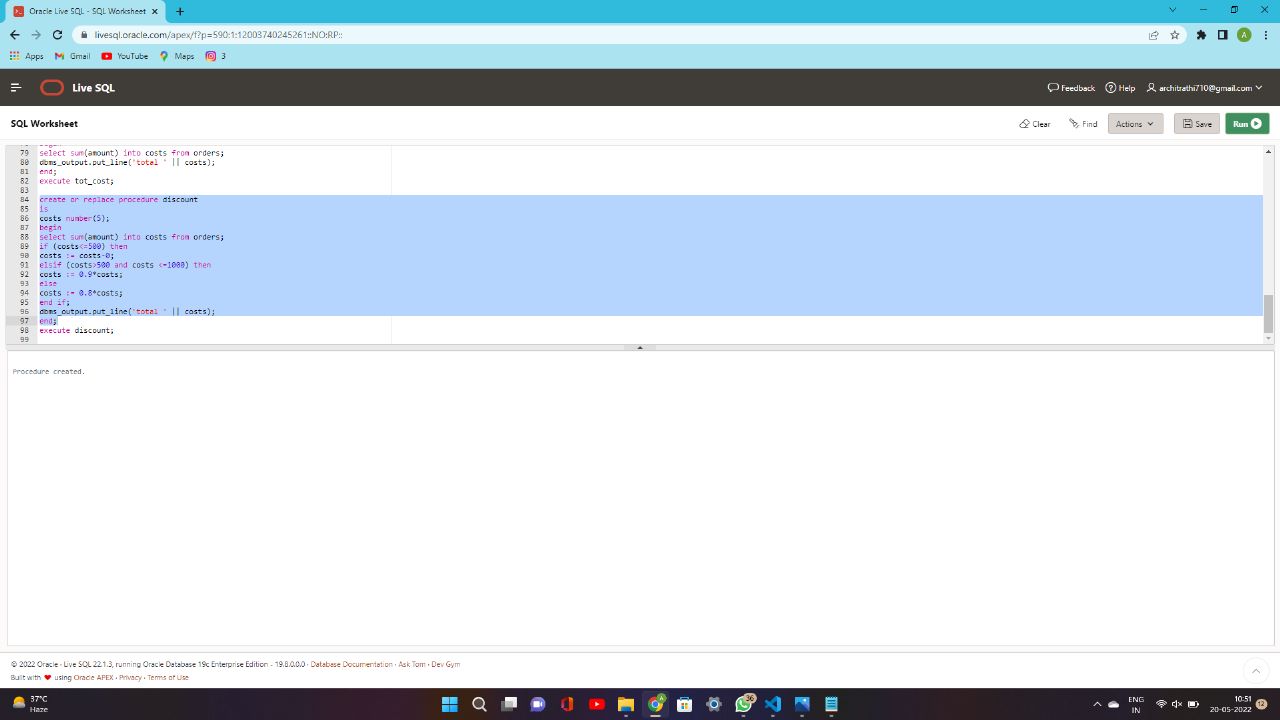
**OUTPUT SCREENSHOTS FOR PROCEDURES**

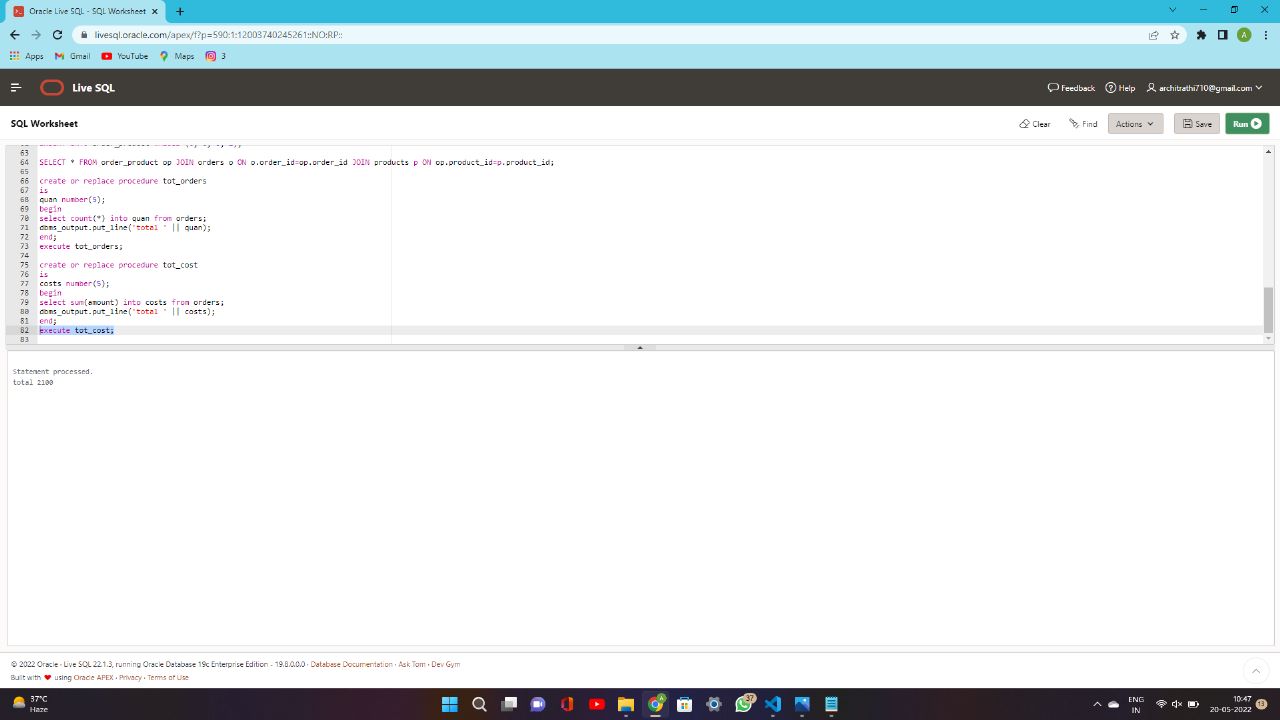
**1.**

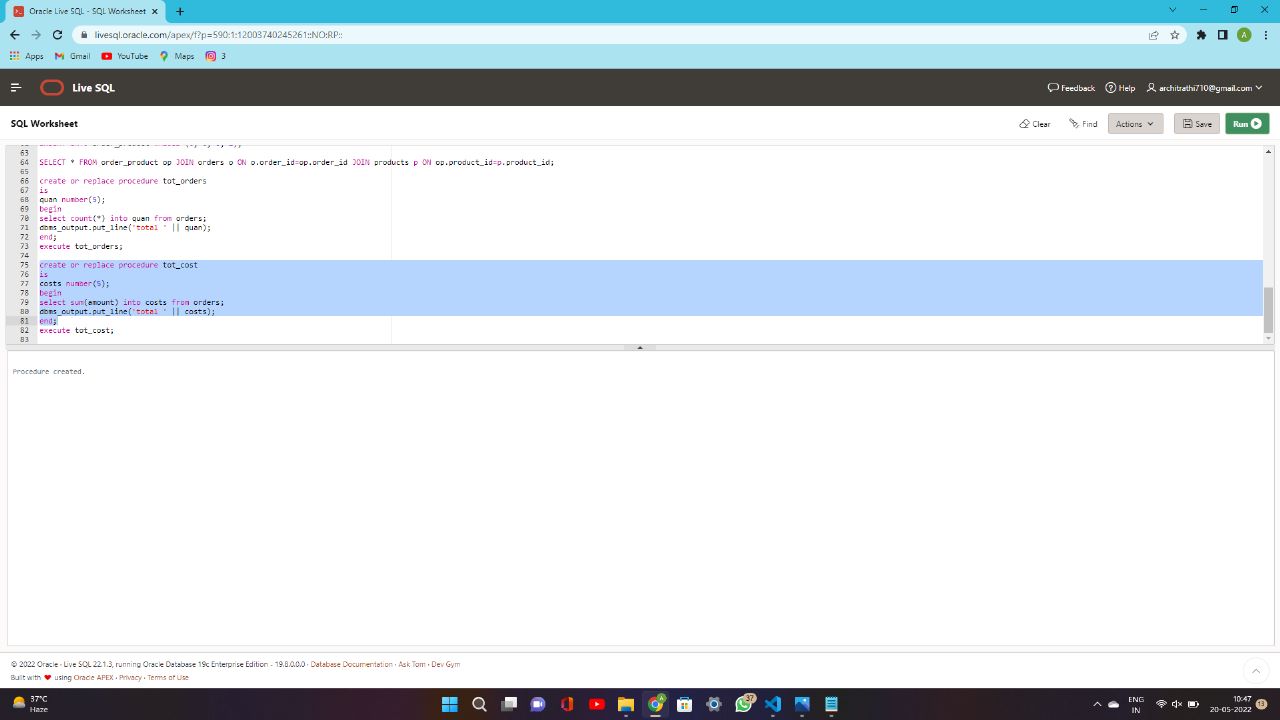
****

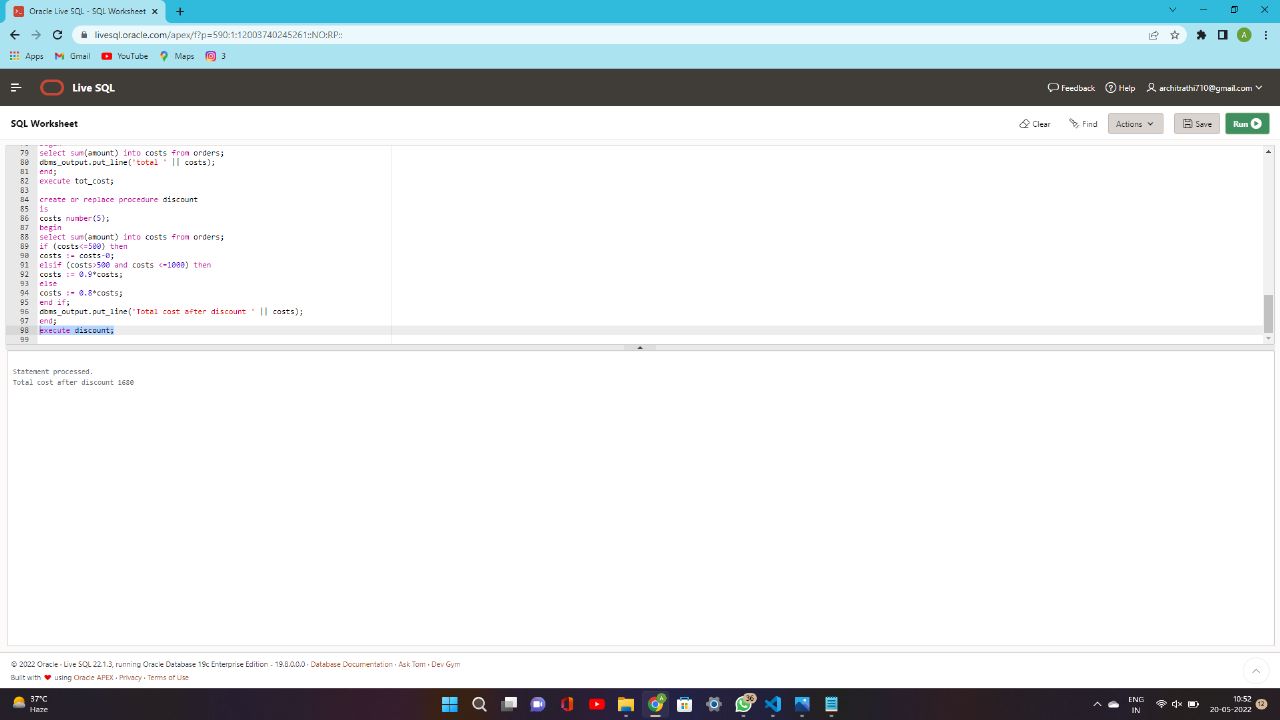
****

2.

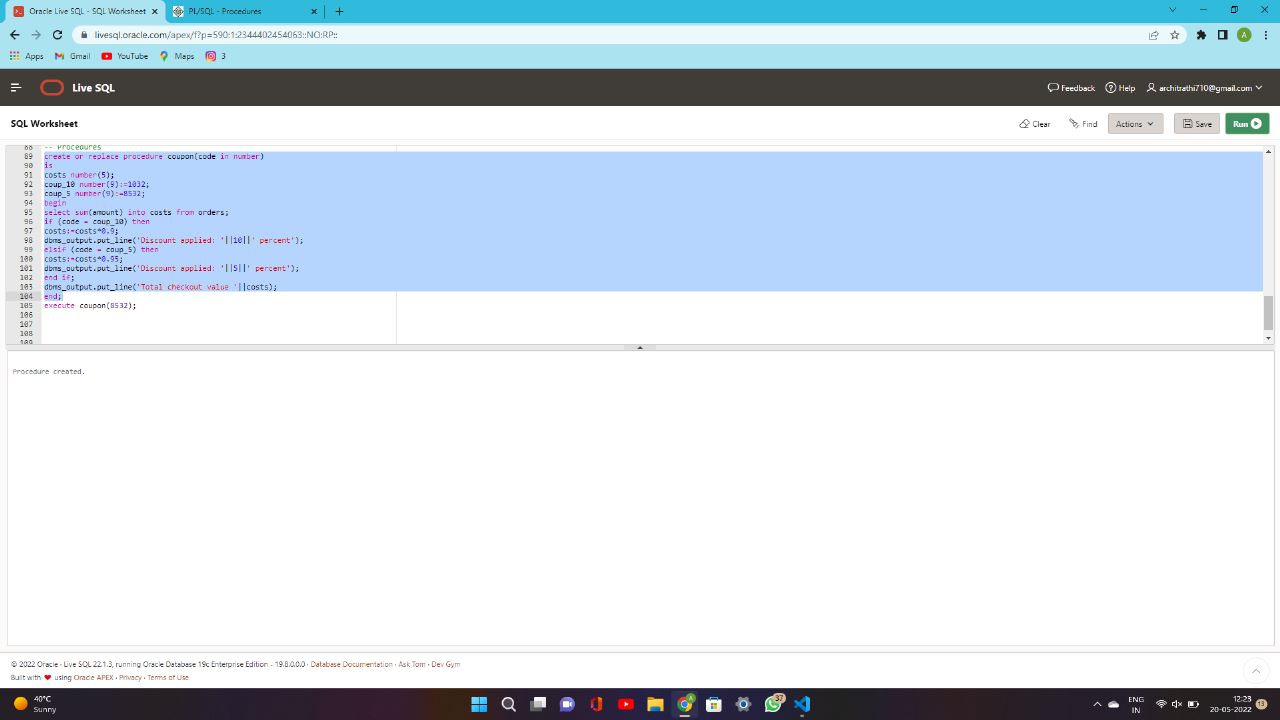


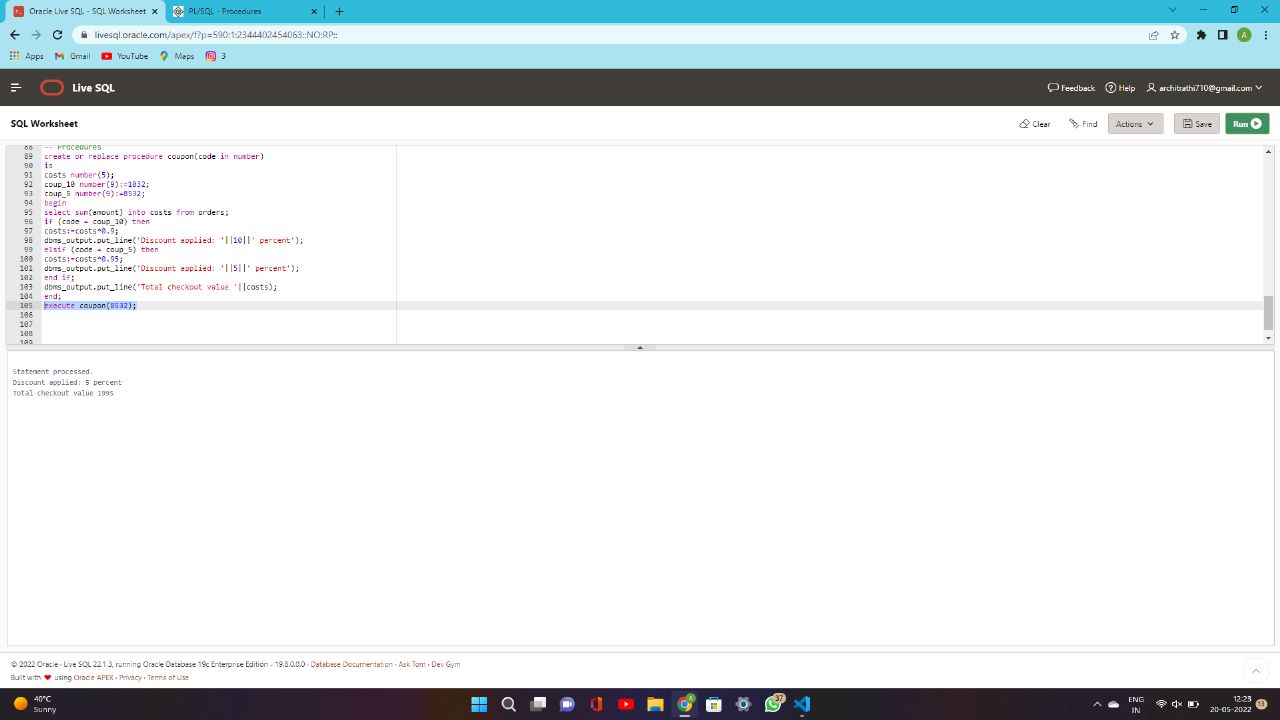


3.



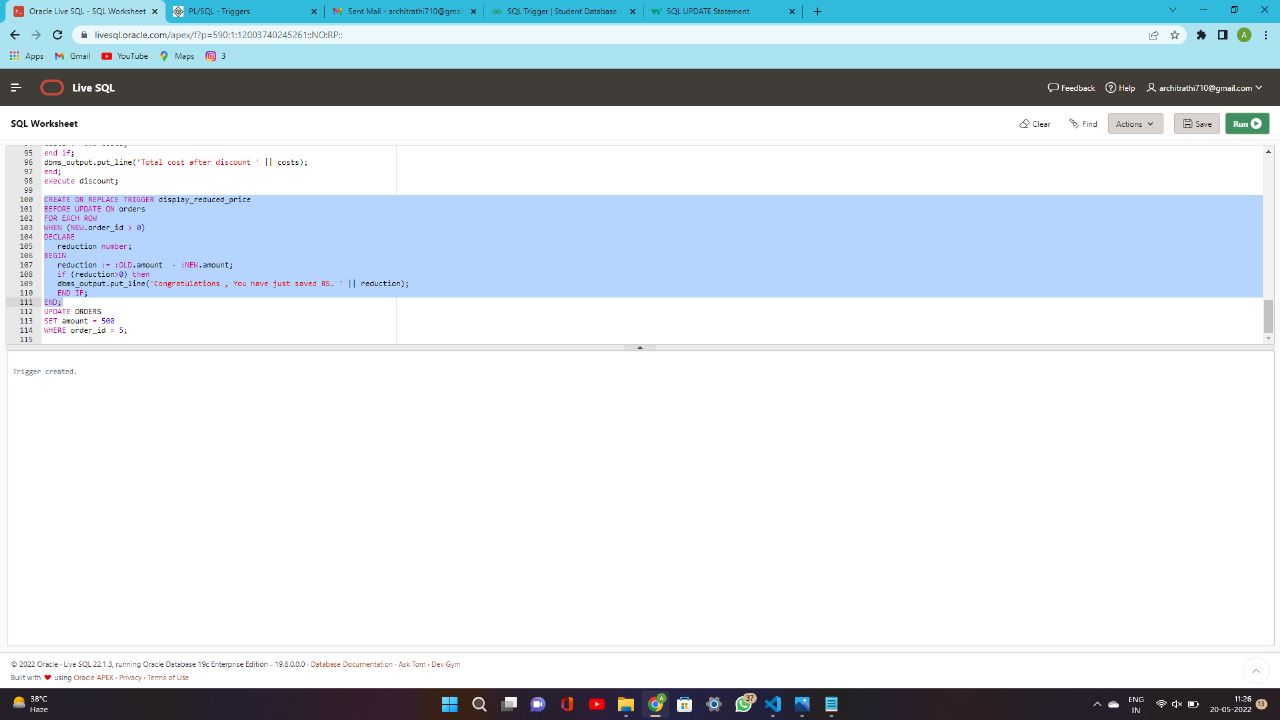
4.

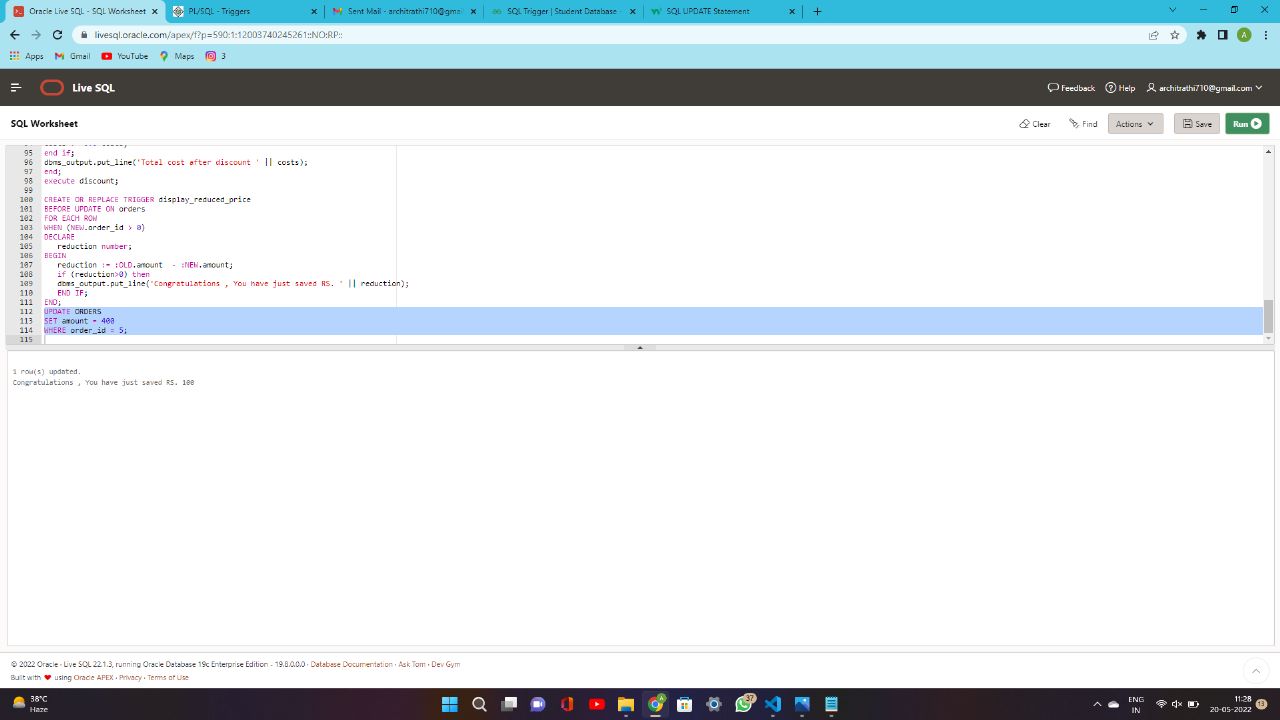




**Trigger**

**If any product is present on your order’s page, then it will display a proper notification for your discounted price**

****

****