Online Cosmetic Store

Content

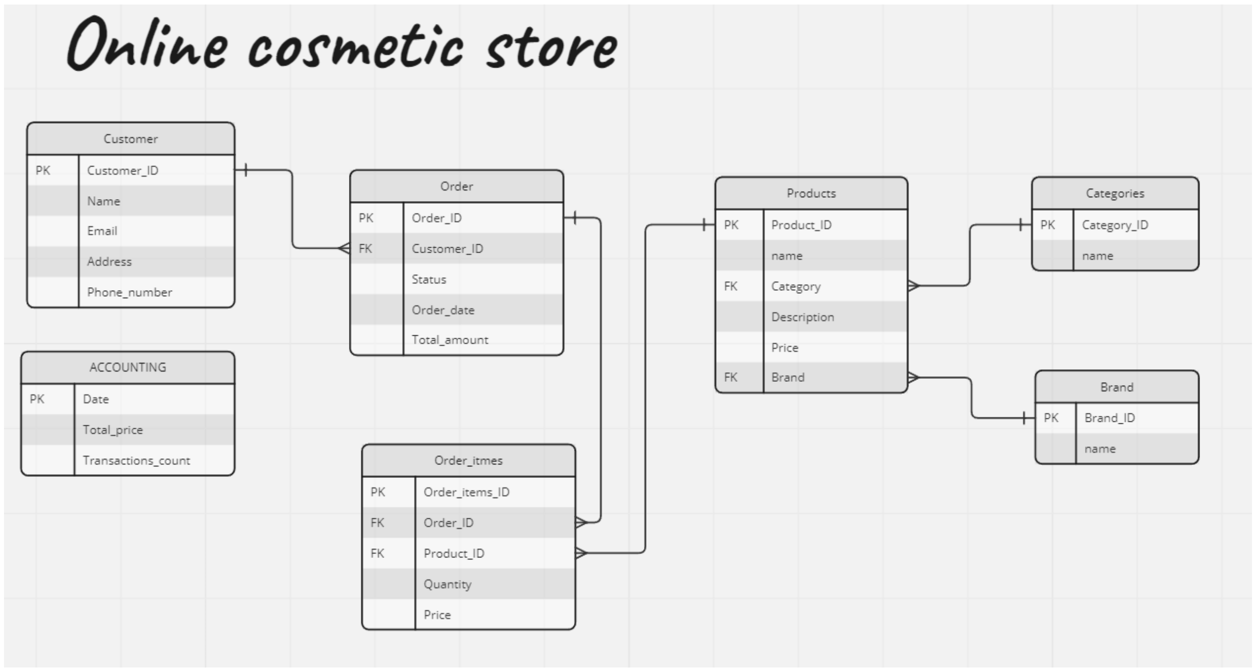
1. Introduction to the System
2. ER diagram
3. Why normal form?
4. Explanation and coding part of each item
    4.1 Triggers
    4.2 Functions
    4.3 Procedure
    4.4 Exception
5. Conclusion

Introduction to the System

The main idea of the project is to create a database that will meet the requirements of an online cosmetics store, fully exploiting the use, featuring high reliability and data efficiency.

The online cosmetic store database system is designed to manage various aspects of the store, such as products, customers, orders, categories, accounting and brands. The system includes seven main tables, each with its own set of attributes and relationships, and is designed to follow the normal forms for efficient and accurate data storage. Triggers, procedures, and user-defined exceptions are also implemented to ensure data integrity and enhance the user experience. The system enables the store to operate smoothly and efficiently, allowing customers to easily browse and purchase products online.

ER diagram



The ER diagram for the above database would have seven tables - products, customers, orders, order_items, category, brand and accounting. The products table would have a many-to-one relationship with the category table and the brand table, with foreign keys category and brand, respectively. The orders table would have a many-to-one relationship with the customers table, with a foreign key customer_id. The order_items table would have a many-to-one relationship with both the orders table and the products table, with foreign keys order_id and product_id, respectively. Accounting table would have a relationship with the orders and order_iterms tables by triggers.

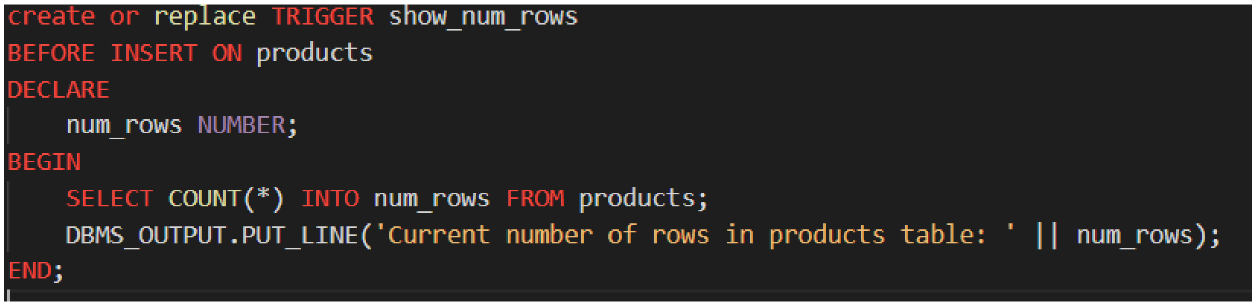
Why db in normal form?

The data structure conforms to the most regular form (3NF) since the table has a primary key, and each key has no attributes that depend only on the direct key. Each table also does not contain duplicate group attributes.

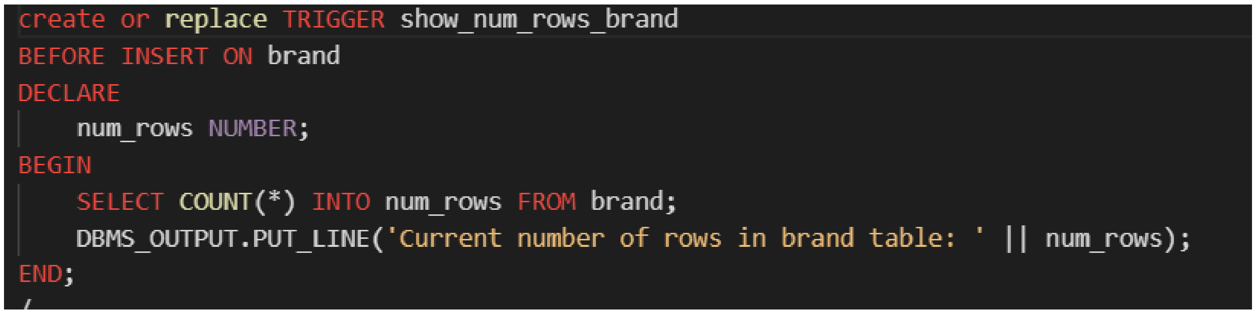
Coding parts

Triggers

1-7 triggers for show the current number of rows in the table

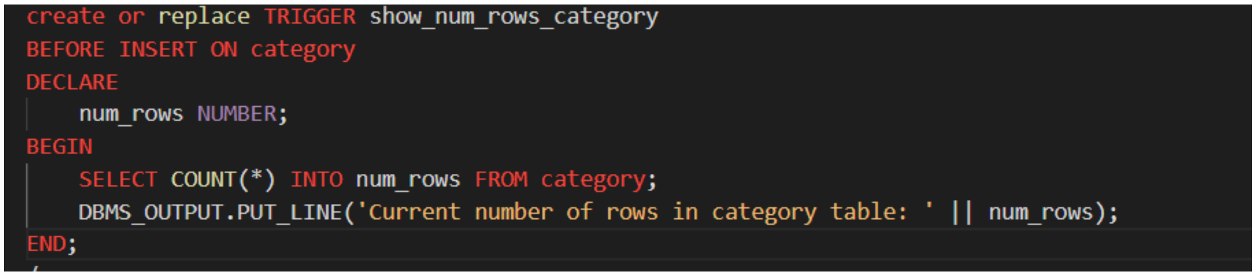
1) 

```
create or replace TRIGGER show_num_rows
BEFORE INSERT ON products
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM products;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in products table: ' || num_rows);
END;
```

2) 

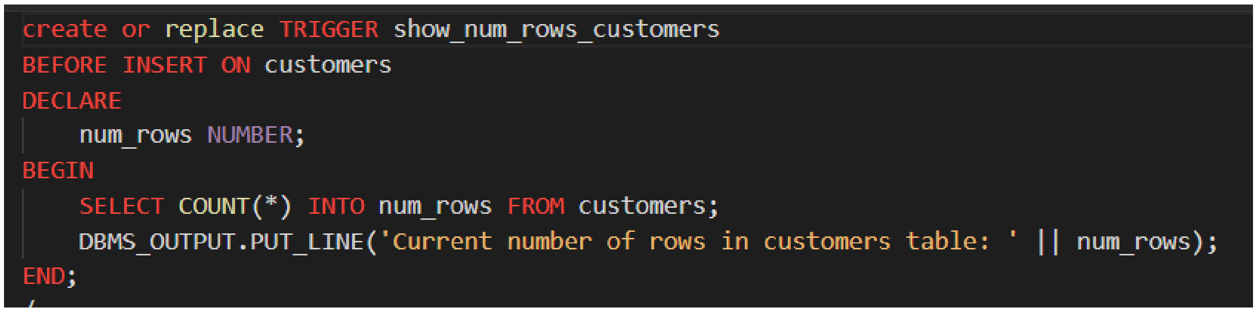
```
create or replace TRIGGER show_num_rows_brand
BEFORE INSERT ON brand
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM brand;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in brand table: ' || num_rows);
END;
```

3)



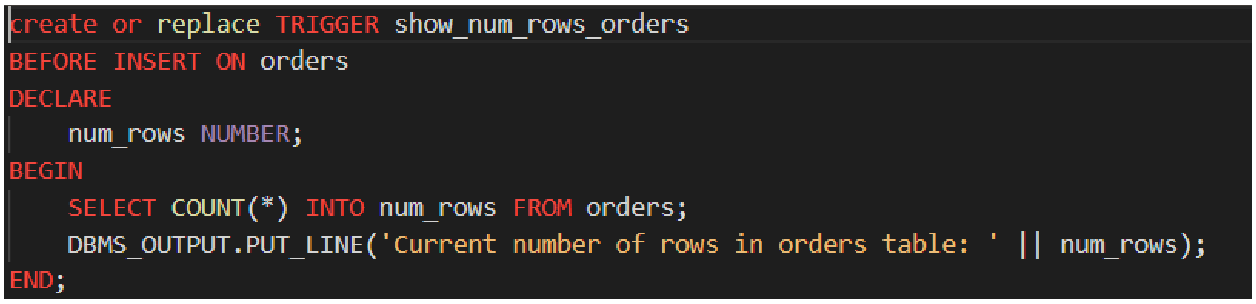
```
create or replace TRIGGER show_num_rows_category
BEFORE INSERT ON category
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM category;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in category table: ' || num_rows);
END;
```

4)



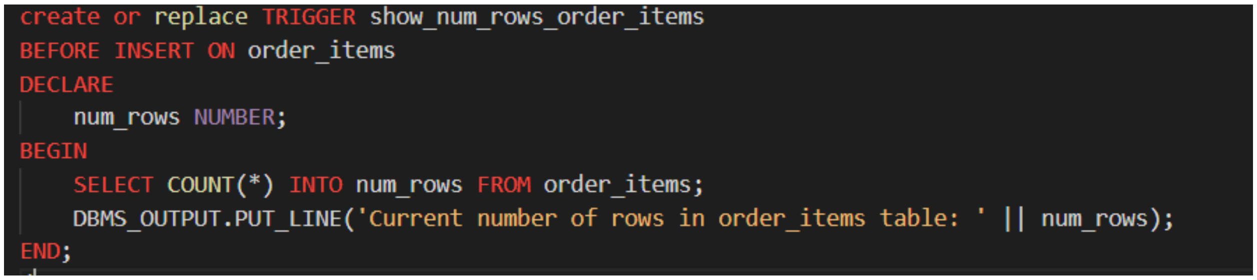
```
create or replace TRIGGER show_num_rows_customers
BEFORE INSERT ON customers
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM customers;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in customers table: ' || num_rows);
END;
```

5)



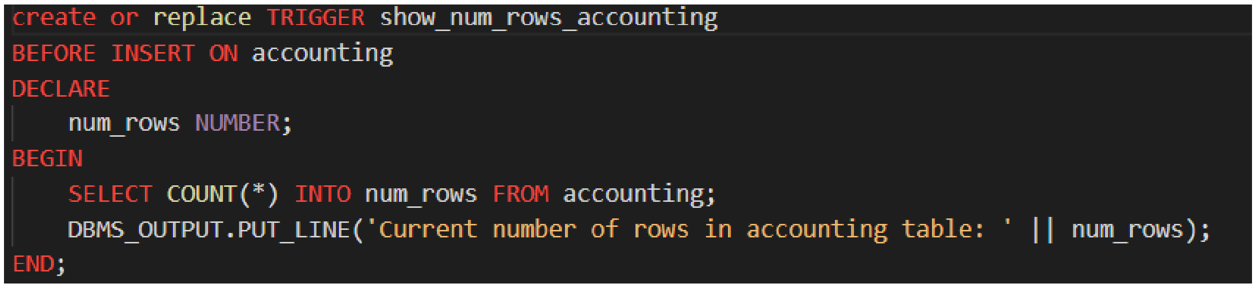
```
create or replace TRIGGER show_num_rows_orders
BEFORE INSERT ON orders
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM orders;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in orders table: ' || num_rows);
END;
```

6)



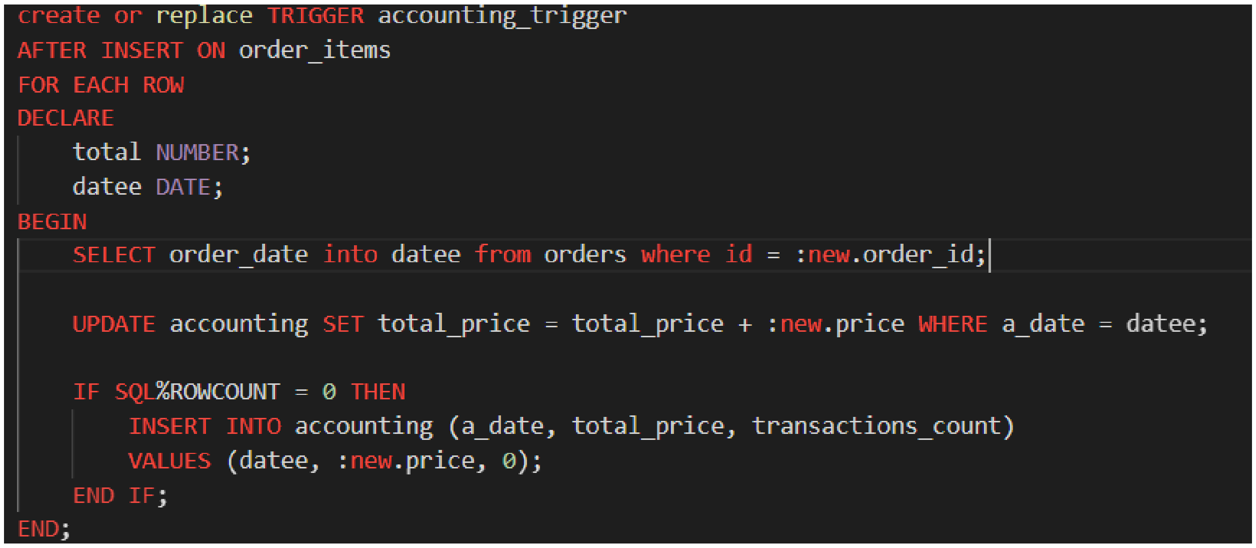
```
create or replace TRIGGER show_num_rows_order_items
BEFORE INSERT ON order_items
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM order_items;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in order_items table: ' || num_rows);
END;
```

7)



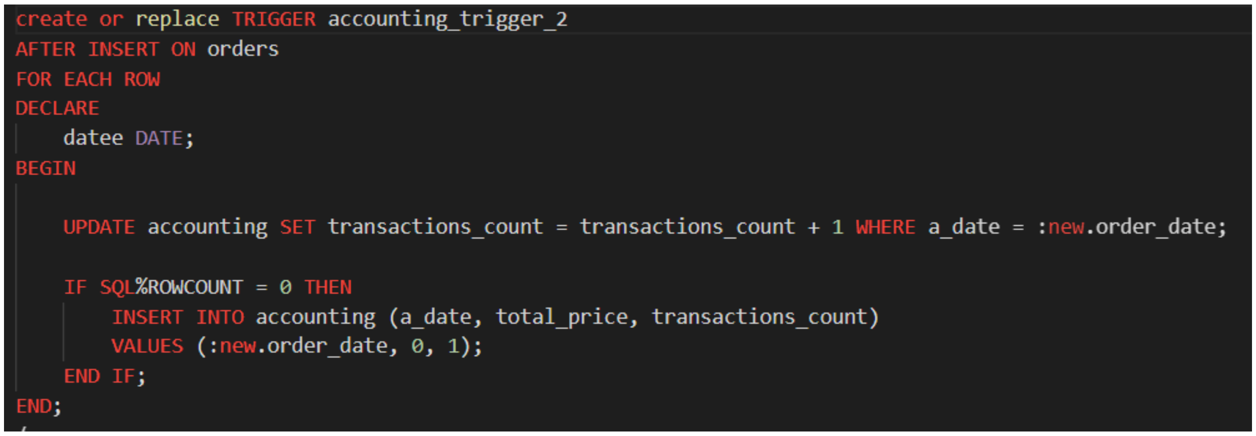
```
create or replace TRIGGER show_num_rows_accounting
BEFORE INSERT ON accounting
DECLARE
    num_rows NUMBER;
BEGIN
    SELECT COUNT(*) INTO num_rows FROM accounting;
    DBMS_OUTPUT.PUT_LINE('Current number of rows in accounting table: ' || num_rows);
END;
```

8 - 9 triggers for relationship accounting with orders and orders_item

8) 

```
create or replace TRIGGER accounting_trigger
AFTER INSERT ON order_items
FOR EACH ROW
DECLARE
    total NUMBER;
    datee DATE;
BEGIN
    SELECT order_date into datee from orders where id = :new.order_id;

    UPDATE accounting SET total_price = total_price + :new.price WHERE a_date = datee;

    IF SQL%ROWCOUNT = 0 THEN
        INSERT INTO accounting (a_date, total_price, transactions_count)
        VALUES (datee, :new.price, 0);
    END IF;
END;
```

this trigger updates the "accounting" table with the total price and transaction count for each date when an order item is inserted into the "order_items" table. If no corresponding row exists in the "accounting" table, it inserts a new row with the transaction information.

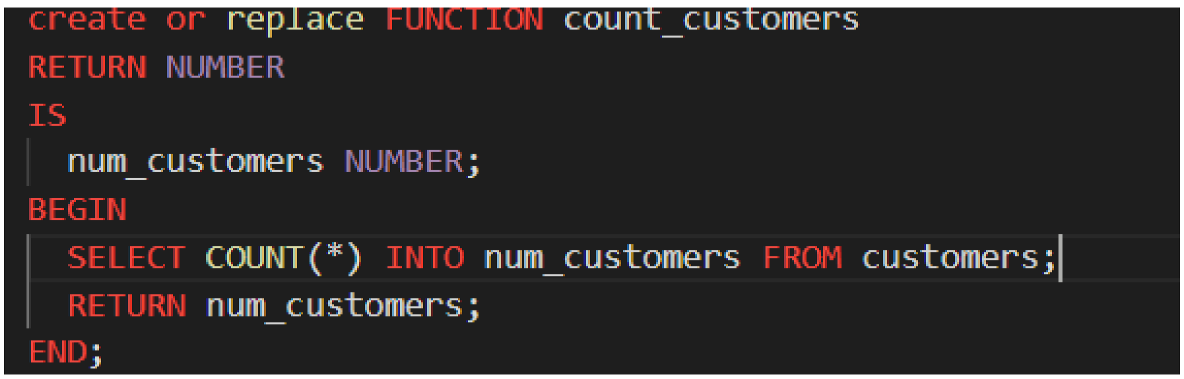
9) 

```
create or replace TRIGGER accounting_trigger_2
AFTER INSERT ON orders
FOR EACH ROW
DECLARE
    datee DATE;
BEGIN

    UPDATE accounting SET transactions_count = transactions_count + 1 WHERE a_date = :new.order_date;

    IF SQL%ROWCOUNT = 0 THEN
        INSERT INTO accounting (a_date, total_price, transactions_count)
        VALUES (:new.order_date, 0, 1);
    END IF;
END;
```

this trigger updates the "accounting" table with the transaction count for each date when an order is inserted into the "orders" table. If no corresponding row exists in the "accounting" table, it inserts a new row with the transaction information, including the date and a transaction count of 1. Note that this trigger does not update the total price for the corresponding date, unlike the previous trigger "accounting_trigger".

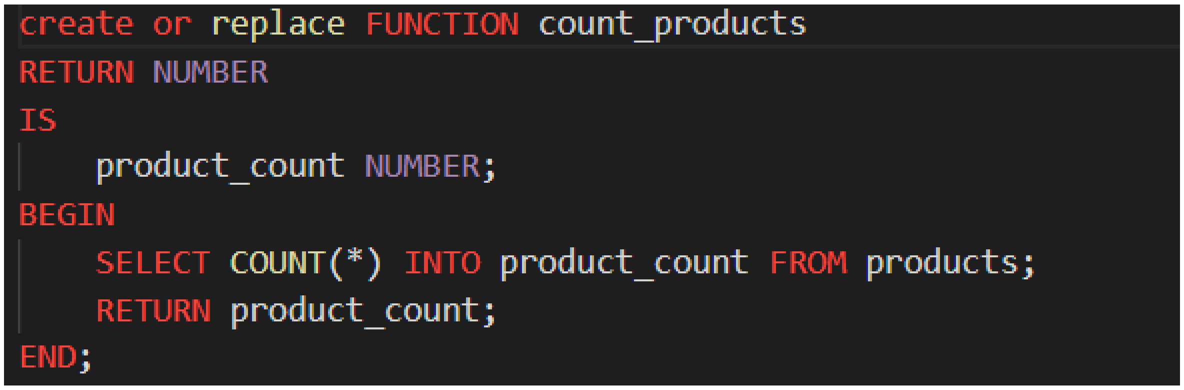
Functions

1)



```
create or replace FUNCTION count_customers
RETURN NUMBER
IS
  num_customers NUMBER;
BEGIN
  SELECT COUNT(*) INTO num_customers FROM customers;
  RETURN num_customers;
END;
```

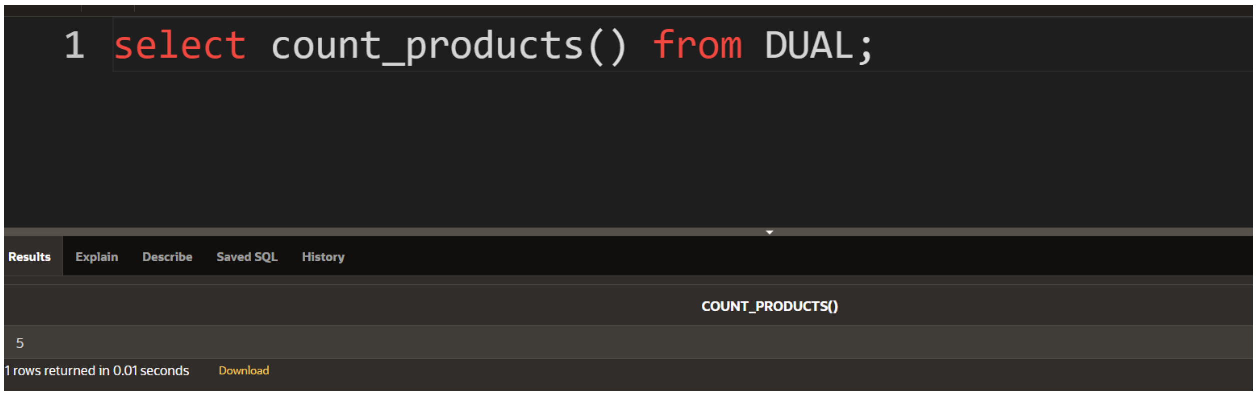
2)



```
create or replace FUNCTION count_products
RETURN NUMBER
IS
    product_count NUMBER;
BEGIN
    SELECT COUNT(*) INTO product_count FROM products;
    RETURN product_count;
END;
```

This is a PL/SQL function called "count_products" that returns the total number of rows in the "products" table of a database.

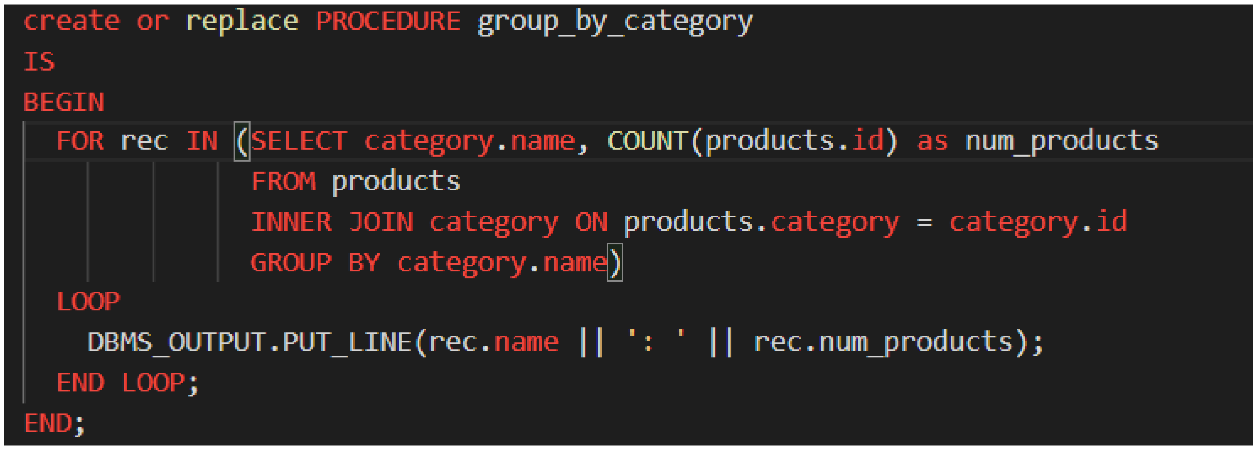
Ex:



```
select count_products() from DUAL;
```

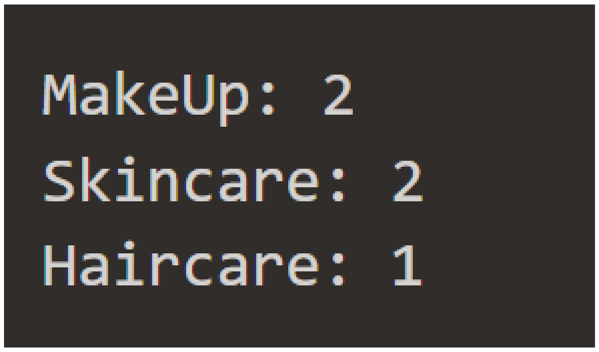
```
COUNT_PRODUCTS()
5
1 rows returned in 0.01 seconds
```

Procedure



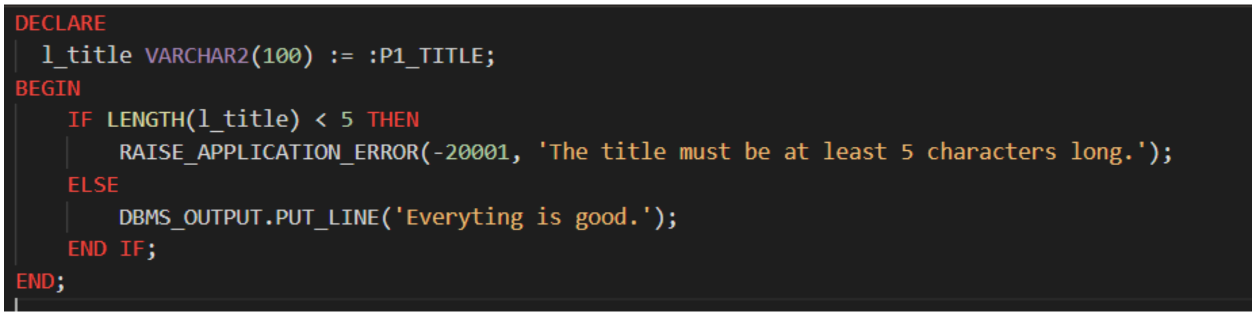
```
create or replace PROCEDURE group_by_category
IS
BEGIN
  FOR rec IN (SELECT category.name, COUNT(products.id) as num_products
              FROM products
              INNER JOIN category ON products.category = category.id
              GROUP BY category.name)
  LOOP
    DBMS_OUTPUT.PUT_LINE(rec.name || ': ' || rec.num_products);
  END LOOP;
END;
```

I created a procedure for grouping by category the products in my online store. Format of printed data would be by “category name: count of product with such category” template

Ex: 

```
MakeUp: 2
Skincare: 2
Haircare: 1
```

Exception



```
DECLARE
  l_title VARCHAR2(100) := :P1_TITLE;
BEGIN
    IF LENGTH(l_title) < 5 THEN
        RAISE_APPLICATION_ERROR(-20001, 'The title must be at least 5 characters long.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Everyting is good.');
    END IF;
END;
```

There is code for validating title or just checking the length of title need to be at least 5

Conclusion

The implementation of the project makes it possible to simplify the work with data, increase the efficiency and accuracy of information processing, as well as increase customer satisfaction and enhance their shopping experience in the store.

It is important to remember that the database is dynamic and requires constant updating and maintenance. Therefore, it is necessary to regularly analyze and update the database so that it meets all the requirements and needs of the store.