DBMS**.**2

**END TERM PROJECT**

**PROJECT NAME:** PET PRODUCTS

**PRESENTED TO:** AZAMAT SEREK

**PRESENTED BY:**  BIRZHAN BAIMURZA 210107007

BEKBOLAT BALGYNBEK 210103205

NURGABYL ZHANERKE 210107019

**PROJECT DESCRIPTION:**

* The goal of this project is to develop a database management system (DBMS) that can help in storing and manipulating data in the environment of pet products. The goal is to provide online sales, provide customers with prompt and accurate product information, effectively manage consumer and transaction data, and track inventory, sales transactions, and financial transfers.
* By implementing the proposed system, the pet products will be able to effectively store and manage data, including stock levels, pricing information, and availability. Furthermore, the system will be able to incorporate customer information, such as personal details, purchase history, and credit card information, to facilitate smooth sales transactions. The system will also collect comprehensive data on sales, such as revenue, cost of goods sold, and profits, and generate reports that offer valuable insights into the pet product's performance.
* The system will also have the capabilities to manage online transactions, track the delivery and keep records of the status of payment and delivery. These features ensure that customers promptly receive updated information about their orders and that pet products keeps accurate records of all transactions.
* The intended DBMS is custom-made to cater to the requirements of small and medium-sized businesses, offering them a complete solution for data storage and manipulation, transaction processing, and financial management. Implementing this system would enable pet products to simplify their operations, minimize errors, and improve the customer experience.

**PROJECT STRUCTURE:**

This database is designed for an online pet store, and it consists of 12 tables that are interrelated through different types of relationships. Here is an overview of the relationships between the tables:

* The Customer table stores information about each customer who makes purchases on the online store. The Customer\_ID column is the primary key of this table.
* The User Account table stores information about customer account details, such as username, email, and password. It is related to the Customer table through the Customer\_ID foreign key.
* The Addresses table stores information about physical addresses that can be associated with one or more users. The Address\_ID column is the primary key of this table.
* The User Addresses table represents a many-to-many relationship between users and addresses. It is related to both the Customer and Addresses tables through foreign keys that together form the composite primary key.
* The Payment Type table stores information about different types of payment methods available to customers. The PaymentType\_ID column is the primary key of this table.
* The Payment table stores information about payments made by customers for their orders. It is related to the Customer and Payment Type tables through foreign keys and has a primary key of Payment\_ID.
* The Category table stores information about the different categories of products sold on the online store. It has a primary key of Category\_ID and is related to itself through the parent Category\_ID foreign key.
* The Supplier table stores information about the suppliers of products sold on the online store. It is related to the Addresses table through the AddressID foreign key and has a primary key of Supplier\_ID.
* The Product table stores information about the products sold on the online store. It is related to the Category and Supplier tables through foreign keys and has a primary key of ProductID.
* The Order table stores information about the orders placed by customers. It is related to the Customer and Payment tables through foreign keys and has a primary key of OrderID.
* The Order Item table stores information about the individual products that are part of each order. It is related to the Order and Product tables through foreign keys and has a primary key of OrderItemID.
* The Shipment table stores information about the shipments of products to customers. It is related to the Order and Addresses tables through foreign keys and has a primary key of ShipmentID.

**Normalization:**

The database is designed to satisfy the normal forms of 1NF, 2NF, and 3NF. Each table has a primary key, and each column in the table has a single atomic value, which satisfies the requirements of 1NF. Each non-key column in the table is fully functionally dependent on the primary key, which satisfies the requirements of 2NF. And finally, there is no transitive dependency between non-key columns in the table, which satisfies the requirements of 3NF. The normalization of the database ensures data integrity and efficient data management.

**TABLES:** USER\_ACCOUNT, USER\_ADDRESS, ADDRESS, CUSTOMER, PRODUCT, CATEGORY, ORDERING, ORDER\_ITEM, SHIPMENT, SUPPLIER, PAYMENT, PAYMENT\_TYPE

**USER\_ACCOUNT:**

CREATE TABLE IF NOT EXISTS user\_account (

username VARCHAR(255) PRIMARY KEY,

customer\_id INTEGER NOT NULL,

email VARCHAR(255) NOT NULL,

password VARCHAR (32) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES Customer (id)

);

**USER\_ADDRESS:**

CREATE TABLE IF NOT EXISTS user\_addresses (

customer\_id INTEGER NOT NULL,

addrs\_id INTEGER NOT NULL,

PRIMARY KEY (customer\_id, addrs\_id),

FOREIGN KEY (customer\_id) REFERENCES Customer (id),

FOREIGN KEY (addrs\_id) REFERENCES Address (id)

);

**ADDRESS:**

CREATE TABLE IF NOT EXISTS Address (

id INTEGER PRIMARY KEY,

unit\_num INTEGER NOT NULL,

street VARCHAR(255) NOT NULL,

city VARCHAR(255) NOT NULL,

region VARCHAR(255) NOT NULL,

postal\_code VARCHAR(255) NOT NULL

);

**CUSTOMER:**

CREATE TABLE IF NOT EXISTS Customer (

id INTEGER PRIMARY KEY,

Fname VARCHAR(255) NOT NULL,

Lname VARCHAR(255) NOT NULL,

tel\_num VARCHAR(20) NOT NULL

);

**PRODUCT:**

CREATE TABLE IF NOT EXISTS Product (

id INTEGER PRIMARY KEY,

category\_id INTEGER NOT NULL,

supplier\_id INTEGER NOT NULL,

name VARCHAR(255) NOT NULL,

description TEXT NOT NULL,

price\_per\_one INTEGER NOT NULL,

FOREIGN KEY (category\_id) REFERENCES Category (id),

FOREIGN KEY (supplier\_id) REFERENCES Supplier (id)

);

**CATEGORY:**

CREATE TABLE IF NOT EXISTS Category (

id INTEGER PRIMARY KEY,

subcategory\_id INTEGER,

name VARCHAR(100) NOT NULL,

FOREIGN KEY (subcategory\_id) REFERENCES Category (id)

);

**ORDERING:**

CREATE TABLE IF NOT EXISTS Ordering (

id INTEGER PRIMARY KEY,

customer\_id INTEGER NOT NULL,

payment\_id INTEGER NOT NULL,

order\_date TIMESTAMP NOT NULL,

total\_price INTEGER NOT NULL,

status VARCHAR(50) NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES Customer (id),

FOREIGN KEY (payment\_id) REFERENCES Payment (id)

);

**ORDER\_ITEM:**

CREATE TABLE IF NOT EXISTS order\_item (

id INTEGER PRIMARY KEY,

order\_id INTEGER NOT NULL,

product\_id INTEGER NOT NULL,

quantity INTEGER NOT NULL,

price INTEGER NOT NULL,

FOREIGN KEY (order\_id) REFERENCES Ordering (id),

FOREIGN KEY (product\_id) REFERENCES Product (id)

);

**SHIPMENT:**

CREATE TABLE IF NOT EXISTS Shipment (

id INTEGER PRIMARY KEY,

addrs\_id INTEGER NOT NULL,

order\_id INTEGER NOT NULL,

shipment\_date TIMESTAMP NOT NULL,

tracking\_num INTEGER NOT NULL,

FOREIGN KEY (addrs\_id) REFERENCES Address (id),

FOREIGN KEY (order\_id) REFERENCES Ordering (id)

);

**SUPPLIER:**

CREATE TABLE IF NOT EXISTS Supplier (

id INTEGER PRIMARY KEY,

addrs\_id INTEGER NOT NULL,

name VARCHAR(255) NOT NULL,

email VARCHAR(255) NOT NULL,

FOREIGN KEY (addrs\_id) REFERENCES Address (id)

);

**PAYMENT:**

CREATE TABLE IF NOT EXISTS Payment (

id INTEGER PRIMARY KEY,

customer\_id INTEGER NOT NULL,

type\_id INTEGER NOT NULL,

provider VARCHAR(50) NOT NULL,

pay\_num INTEGER NOT NULL,

date DATE NOT NULL,

FOREIGN KEY (customer\_id) REFERENCES Customer (id),

FOREIGN KEY (type\_id) REFERENCES payment\_type (id)

);

**PAYMENT\_TYPE:**

CREATE TABLE IF NOT EXISTS payment\_type (

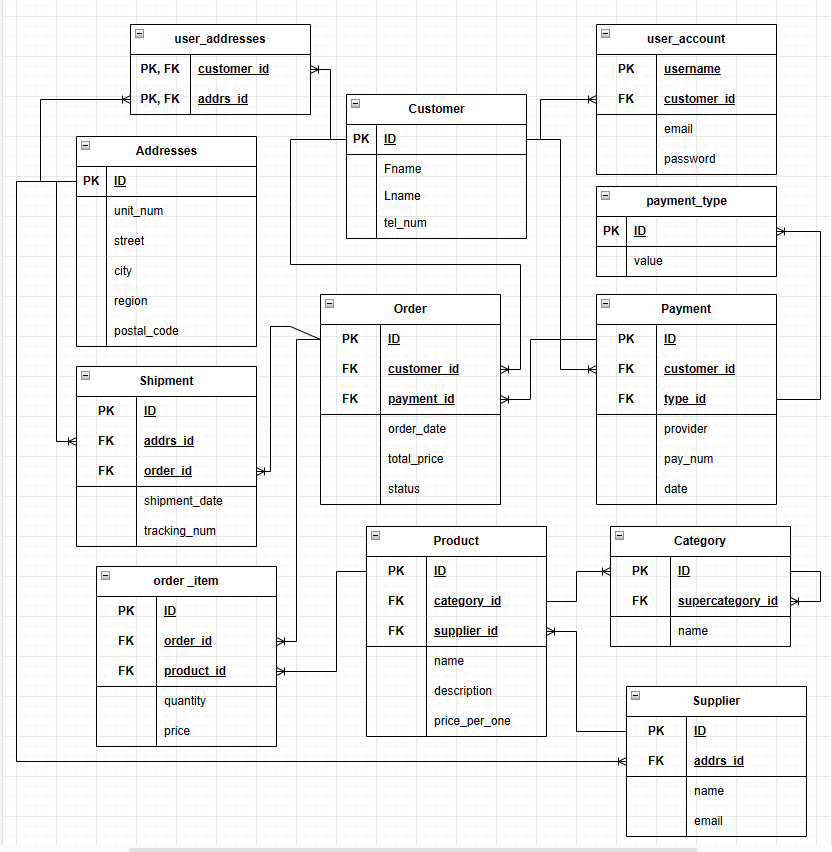
id INTEGER PRIMARY KEY,

value VARCHAR(255) NOT NULL

);

**ENTITY RELATIONSHIP DIAGRAM (ERD):**

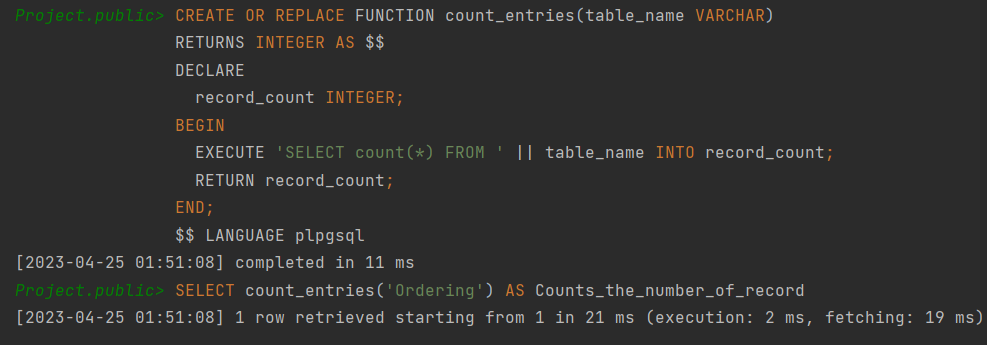
[**HERE IS THE LINK TO (ERD)**](https://drive.google.com/file/d/1kET_-pymHW7weAntJcwY8LO4ysSBkRJK/view)



**PROCEDURES:**

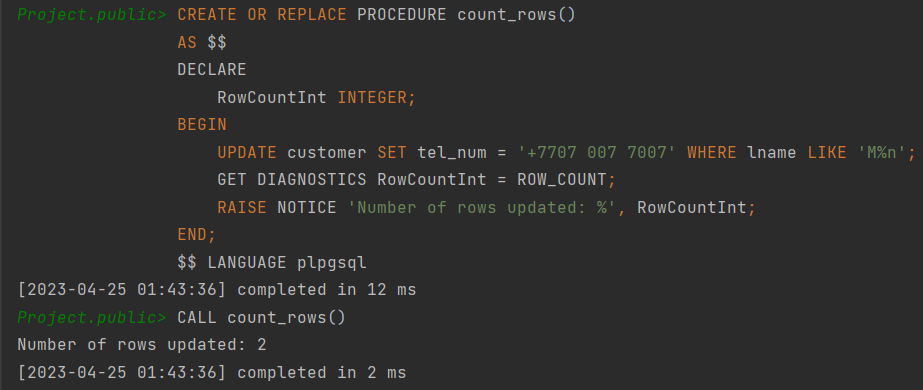
1. **PROCEDURE**

CREATE OR REPLACE PROCEDURE *get\_category\_sales*()  
AS $$  
DECLARE  
 category\_name varchar(255);  
 total\_sales int;  
 total\_revenue float;  
BEGIN  
 FOR category\_name, total\_sales, total\_revenue IN  
 SELECT c.name, *SUM*(oi.quantity) as total\_sales, *SUM*(oi.quantity \* p.price\_per\_one) as total\_revenue  
 FROM category c  
 JOIN product p ON c.id = p.category\_id  
 JOIN order\_item oi ON p.id = oi.product\_id  
 GROUP BY c.id  
 LOOP  
 RAISE NOTICE 'Category Name: %, Total Sales: %, Total Revenue: %', category\_name, total\_sales, total\_revenue;  
 END LOOP;  
END;  
$$ LANGUAGE plpgsql;  
CALL *get\_category\_sales*();



2) **PROCEDURE**

CREATE OR REPLACE PROCEDURE *count\_rows*()  
AS $$  
DECLARE  
 RowCountInt INTEGER;  
BEGIN  
 UPDATE customer SET tel\_num = '+7707 007 7007' WHERE lname LIKE 'M%n';  
 GET DIAGNOSTICS RowCountInt = ROW\_COUNT;  
 RAISE NOTICE 'Number of rows updated: %', RowCountInt;  
END;  
$$ LANGUAGE plpgsql;  
CALL *count\_rows*();



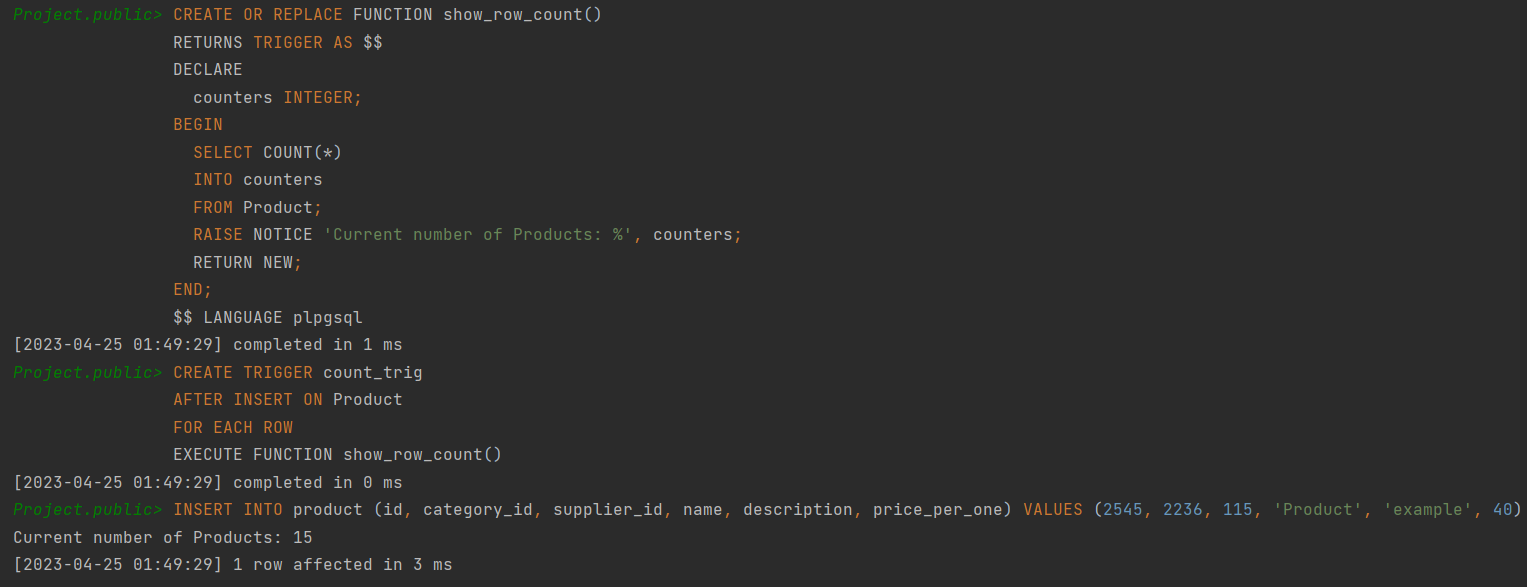
This code creates a procedure called "count\_rows". When executed, it will update the phone number of all customers with a last name that starts with "M" and ends with "n" to "+7707 007 7007".

The procedure uses the UPDATE statement with the WHERE clause that filters out the customers with last names that match the specified pattern. It then uses the GET DIAGNOSTICS statement to retrieve the number of rows affected by the UPDATE statement. It finally uses the RAISE NOTICE statement to print out the number of rows that were updated.

To execute the procedure, it can be called using the "CALL" statement followed by the name of the procedure ("count\_rows()").

**TRIGGER:**

CREATE OR REPLACE FUNCTION *show\_row\_count*()  
RETURNS TRIGGER AS $$  
DECLARE  
 counters INTEGER;  
BEGIN  
 SELECT *COUNT*(\*)  
 INTO counters  
 FROM Product;  
 RAISE NOTICE 'Current number of Products: %', counters;  
 RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER count\_trig  
AFTER INSERT ON Product  
FOR EACH ROW  
EXECUTE FUNCTION *show\_row\_count*();  
INSERT INTO product (id, category\_id, supplier\_id, name, description, price\_per\_one) VALUES (2545, 2236, 115, 'Product', 'example', 40);

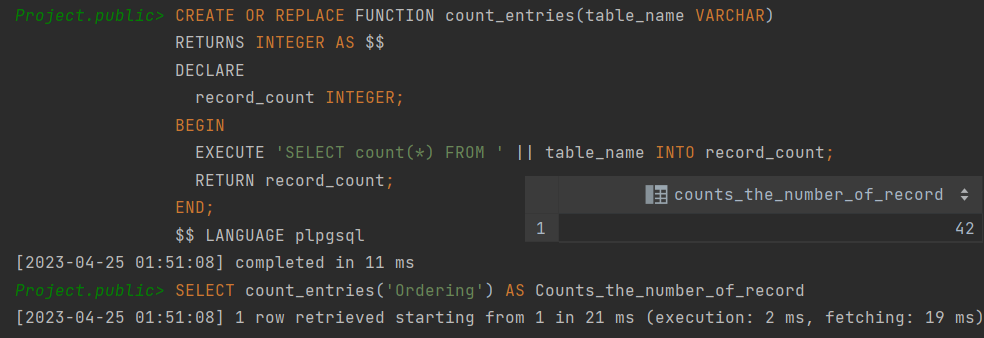


This is a SQL code that creates a function and a trigger for a database table called "Product".

The trigger is activated every time a new row is inserted into the "Product" table. It executes the "show\_row\_count()" function, which counts the number of rows in the "Product" table and stores it in the "counters" variable. Then, it prints out the current number of rows as a notice message.

**FUNCTION:**

CREATE OR REPLACE FUNCTION *count\_entries*(table\_name VARCHAR)  
RETURNS INTEGER AS $$  
DECLARE  
 record\_count INTEGER;  
BEGIN  
 EXECUTE 'SELECT count(\*) FROM ' || table\_name INTO record\_count;  
 RETURN record\_count;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT *count\_entries*('Ordering') AS Counts\_the\_number\_of\_record;

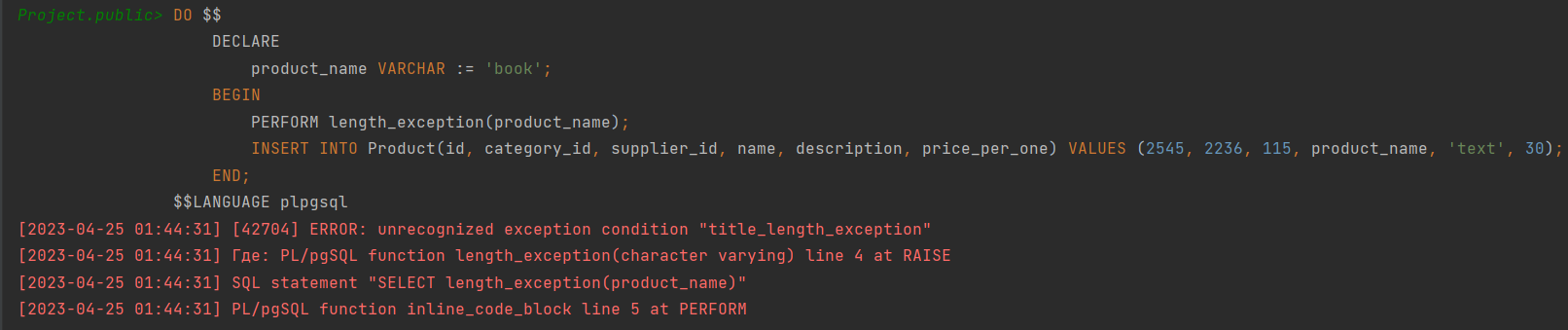


This code defines a function called "count\_entries" that takes a table name as input and returns the number of records in that table. The function uses dynamic SQL to execute a SELECT statement with the table name as a parameter. The record count is then stored in the "record\_count" variable. Finally, the function returns the value of "record\_count".

To use the function, the code calls it with the table name 'Ordering' as the input parameter and then calls the result "Counts\_the\_number\_of\_record". This will return a count of the number of records in the 'Ordering' table.

**EXCEPTION:**

CREATE OR REPLACE FUNCTION *length\_exception*(title VARCHAR)  
RETURNS VOID AS $$  
BEGIN  
 IF *LENGTH*(title) < 5 THEN  
 RAISE EXCEPTION 'Title must be at least 5 characters long.'  
 USING errcode = 'title\_length\_exception';  
 END IF;  
END;  
$$ LANGUAGE plpgsql;  
  
DO $$  
 DECLARE  
 product\_name VARCHAR := 'book';  
 BEGIN  
 PERFORM *length\_exception*(product\_name);  
 INSERT INTO Product(id, category\_id, supplier\_id, name, description, price\_per\_one) VALUES (2545, 2236, 115, product\_name, 'text', 30);  
 END;  
$$LANGUAGE plpgsql;



This code creates a function called "length\_exception" that takes a "title" argument and checks if the length of the title is less than 5 characters. If the length is less than 5, it raises an exception with a custom error message and error code.

The code then executes a "DO" block that declares a variable called "product\_name" with a value of 'book'. It then performs the "length\_exception" function on "product\_name". If the length of "product\_name" is less than 5 characters, the function raises an exception.