A

Project Report On The Electronic Store Management System for Midterm

By

Symbat Mekembayeva	**210103098**
Safira Abilkaiyr	**2101033230**
Nuray Aliyeva	**210103406**
Akezhan Sagidrakhmanov	**210103005**
Dulat Sultanov	**180103197**

Bachelor Degree Program
DEPARTMENT OF INFORMATION SYSTEMS
FACULTY OF Engineering
Course: Database Management System 2

Declaration

I hereby declare that this is our team’s own work. This project, neither in whole nor in part, has been previously submitted for anywhere.

Abstract:

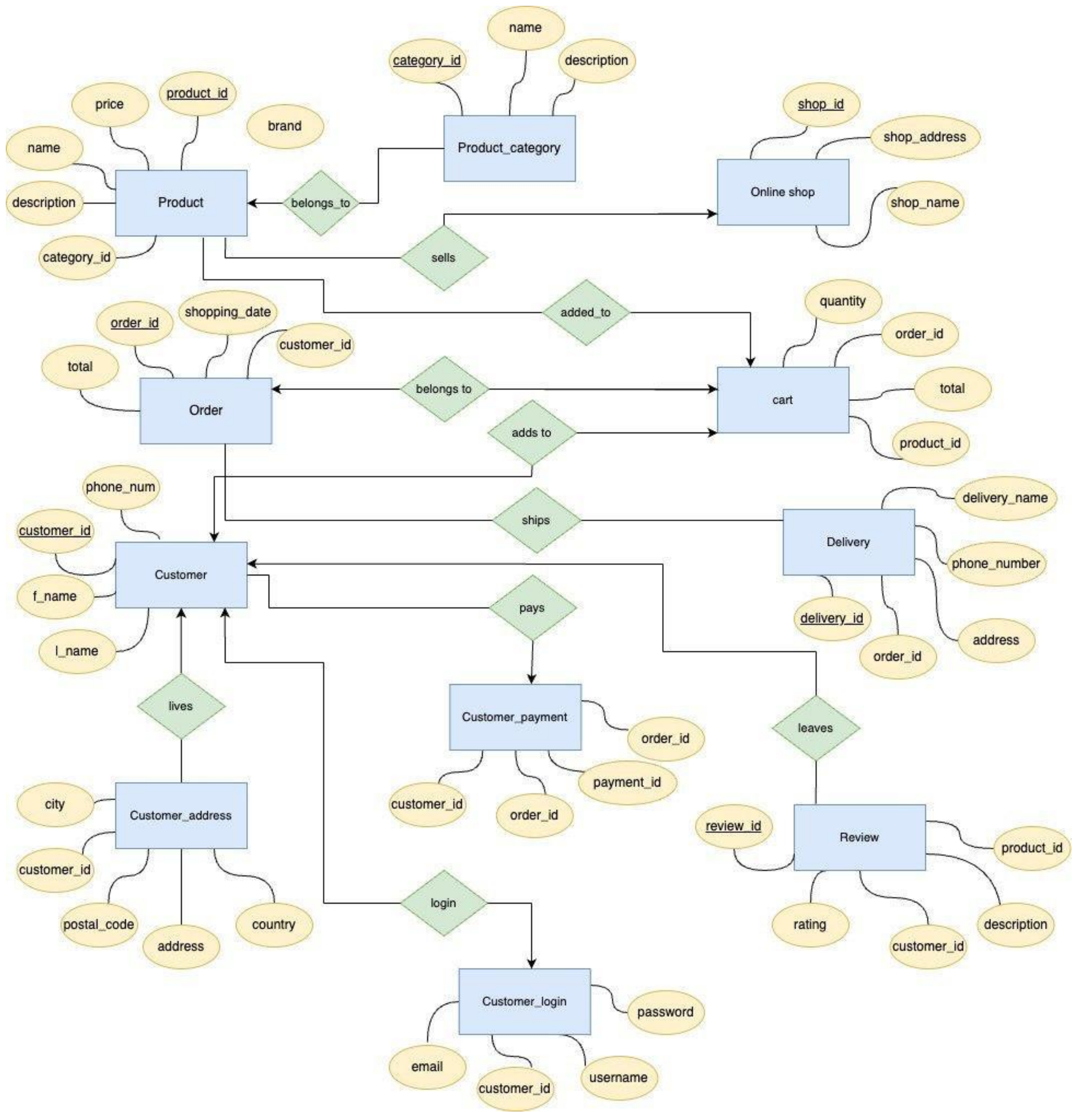
Our main focus is design a unique e-commerce Management system System for electronic store.The system is written in PL|SQL and SQL.

INTRODUCTION

Our electronic store is a e-commerce shop which is used for ordering items online by users. Our management system deals with various activities related to shopping online.

We have used x entities which are defined as tables, x procedures and functions, triggers and exceptions depending on requirements for realization of our project. Our project includes various tasks such as logging in to the site, adding goods to basket, paying for them and giving feedback about bought products.

ERD


name
category_id
description
price
product_id
brand
shop_id
name
Product_category
shop_address
description
Product
belongs_to
Online shop
shop_name
category_id
sells
quantity
shopping_date
added_to
order_id
order_id
customer_id
total
belongs to
total
cart
Order
adds to
product_id
phone_num
delivery_name
customer_id
ships
Delivery
Customer
phone_number
f_name
pays
delivery_id
address
l_name
order_id
Customer_payment
order_id
lives
leaves
payment_id
city
Customer_address
customer_id
order_id
review_id
Review
product_id
customer_id
postal_code
login
rating
description
country
address
customer_id
Customer_login
password
email
customer_id
username

ERD EXPLANATION

This is our ERD with 11 entities in total, and tables are implemented in the same way. Here is the explanation of ERD and table connections. Our operation begins with the fact that:

1. The Customer logs into an account in an online electronics store and enters his data: in our case, E-mail, Username and Password
2. The Customer's table contains information such as the Customer's name, phone number, and ID
3. This table is linked to a table with information about the Customer's place of residence: address, city, country and postal code
4. One of the main tables is a Product that has a Name, price, description, category, brand
5. The buyer can put the goods in the product basket, where you can find out the quantity , the total amount and the Order ID
6. In the Orders table, there are attributes such as the quantity of goods, the date of purchase, and the Order and Customer ID's
7. The order is paid using the Customer's Payment table with the attributes of ID of payment, Customer's ID and Order ID
8. The order has a delivery with the courier's name, phone number
9. At the end of the operation, the buyer can leave his feedback about a particular product with a Review table with a description, rating and product ID

NORMAL FORMS

Based on the provided entities and their attributes, here is how the structure can be organized into 1NF, 2NF, and 3NF::

1NF (First Normal Form):

Each attribute must have atomic values (i.e., no multi-valued attributes or repeating groups).
All the provided entities seem to have attributes with atomic values, so they already meet the requirements of 1NF.

2NF (Second Normal Form):

The table must meet the requirements of 1NF.
All the non-prime attributes (attributes that are not part of the primary key) must depend fully on the primary key.
In the "cart" table, the "total" attribute depends on the "quantity" and "order_id" attributes, but not fully on the "order_id" attribute. It seems like "total" should be moved to the "order" table, as it depends on both "quantity" and "price" attributes in the "product" table, which are related to the "order" entity.

3NF (Third Normal Form):

The table must meet the requirements of 2NF.
There should be no transitive dependencies, i.e., no non-prime attribute should depend on another non-prime attribute.
In the "delivery" table, the "phone_number" attribute seems to be a non-prime attribute that is dependent on the "delivery_id" attribute, which is the primary key. This could indicate a transitive dependency issue, as "phone_number" should be a direct attribute of the "customer" entity, rather than being dependent on the "delivery_id" attribute.

Exception

```
DECLARE
  item_name  product.name%type := :item_n;
  -- user defined exception
  invalid_name  EXCEPTION;
BEGIN
  IF length(item_name)<5 THEN
    RAISE invalid_name;
  END IF;
EXCEPTION
  WHEN invalid_name THEN
    dbms_output.put_line('Name of product should not be less than 5
characters');
END;
```

This exception is used when customers search for products. It is raised when the name of a product which is entered by the user contains less than 5 characters. And when exception is raised, it outputs a message which says that product name should contain at least 5 characters. Here item_n is inputted string which is assigned to item_name for searching.

Trigger

```
    1) Create or replace trigger num_of_rows
Before insert on cart
Declare
total_rows:=sql%rowcount;
begin
dbms_output.put_line(total_rows);
end
```

This trigger’s function is to output a number of rows in the cart table before every insertion to it. We are declared total_rows variable which is counted by sql%rowcount and then outputs it when there is an insertion.

```
    2)  CREATE OR REPLACE TRIGGER trg_customer_address
  BEFORE INSERT ON Customer_address
DECLARE
  num_rows NUMBER;
BEGIN
  SELECT COUNT(*) INTO num_rows FROM Customer_address;
  DBMS_OUTPUT.PUT_LINE('The current number of rows in the Customer_address
table is ' || num_rows);
END;
/
```

This trigger will fire before every insert into the Customer_address table and will display the current number of rows in the table using the DBMS_OUTPUT.PUT_LINE procedure. Please note that in order to view the output of this trigger, you need to enable SET SERVEROUTPUT ON in your SQL client.

PROCEDURES AND FUNCTIONS

1)
```
CREATE OR REPLACE PROCEDURE review_by_product (
    proc_description IN review.description%TYPE,
    proc_customer_id IN review.customer_id%TYPE
)
IS
BEGIN
    FOR rec IN (
        SELECT product_id, COUNT(*) AS num_reviews
        FROM review
        WHERE description = proc_description AND customer_id =
proc_customer_id
        GROUP BY product_id
    ) LOOP
        DBMS_OUTPUT.PUT_LINE('Product ID: '  rec.product_id  ', Number of
Reviews: ' || rec.num_reviews);
    END LOOP;
END;
```

This procedure takes two input parameters, proc_description and proc_customer_id, which are used to group reviews by description and customer ID. The procedure then uses a SELECT statement to group reviews by product ID and count the number of reviews for each product.
And to call this procedure we use

```
BEGIN
   review_by_product('example description', '1234567890');
END;
```

2)
```
CREATE OR REPLACE FUNCTION count_customer_ids
RETURN NUMBER
IS
  record_count NUMBER;
BEGIN
  SELECT COUNT(*) INTO record_count
  FROM Customer;

  RETURN record_count;
END;
```

This function selects the number of all rows in the "Customer" table using the COUNT(*) aggregate function and stores the result in the "record_count" variable. Finally, it returns the counter to the calling program or user.

3)
```
CREATE OR REPLACE PROCEDURE update_records
IS
BEGIN
  UPDATE Product
  SET price = price * 1.1
  WHERE Category_id = '220103550';

  DBMS_OUTPUT.PUT_LINE('Number of rows updated: ' || SQL%ROWCOUNT);
END;
```

A PL/SQL procedure that updates the price of products in the Product table for a specific category and outputs the number of rows updated. The SQL%ROWCOUNT function returns the number of rows affected by the most recent statement, which in this case is the UPDATE statement.