

Endterm Project

Online Shop

Yana Yerzhanova – 190103369

Askarov Ayan – 210103070

Aibek Otepbergen – 210103302

Baktiyar Rassil – 210103297

Mukhamedzhan Saifudin - 210103011

Database Management 2

Introduction

The database project for an online store is an important component of e–commerce, which provides storage of information about products, customers, orders, payment and delivery. This allows you to manage and control all aspects of online sales, from tracking goods in stock to processing and delivery of orders.

An ER diagram is a graphical representation of entities and the relationships between them. In the project, the ER diagram allowed us to understand what data needs to be stored and how they are related to each other. This made it possible to determine the correct database structure and correctly distribute information across tables.

Table normalization is a database optimization process that avoids redundant and inconsistent information. In the project, we used three forms of normalization, which allowed us to reduce the size of the database and improve its performance.

Triggers are software modules that are automatically triggered by certain user actions. In the project, we used triggers to automatically update information in the database with certain actions of the client or administrator, which increased the usability of the system.

As a result of our work, we have created a database that allows you to efficiently store, process and manage online store data. This allows you to reduce the time and cost of processing orders, as well as improve the quality of customer service.

Overview

Introduction

Entity Relationship Design:

1. List of tables
2. ER diagram & Relationship

Normalization:

1. Explanation
2. Tables in BCNF forms

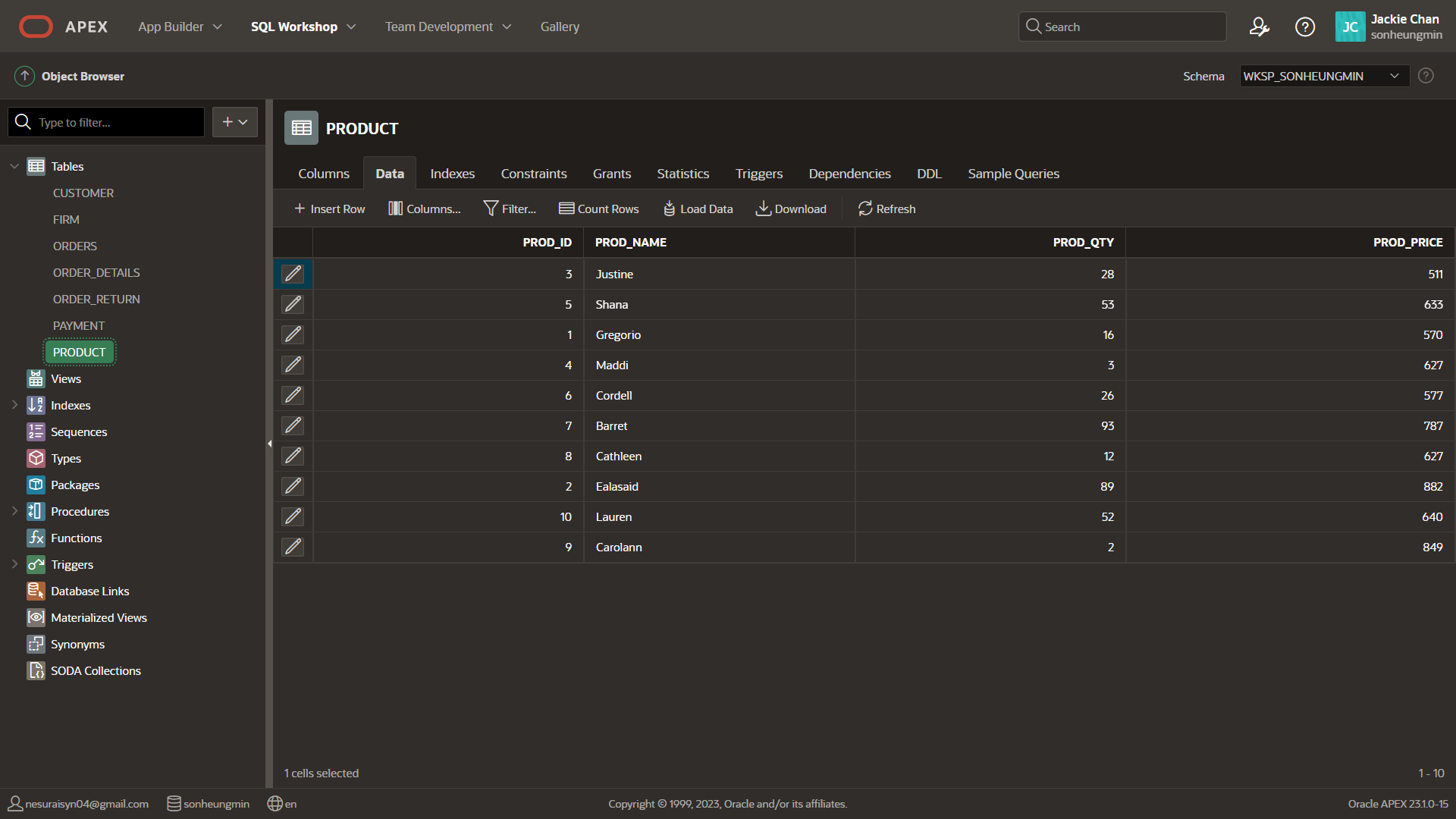
Functions & Triggers:

1. Functions
2. Procedures
3. Triggers

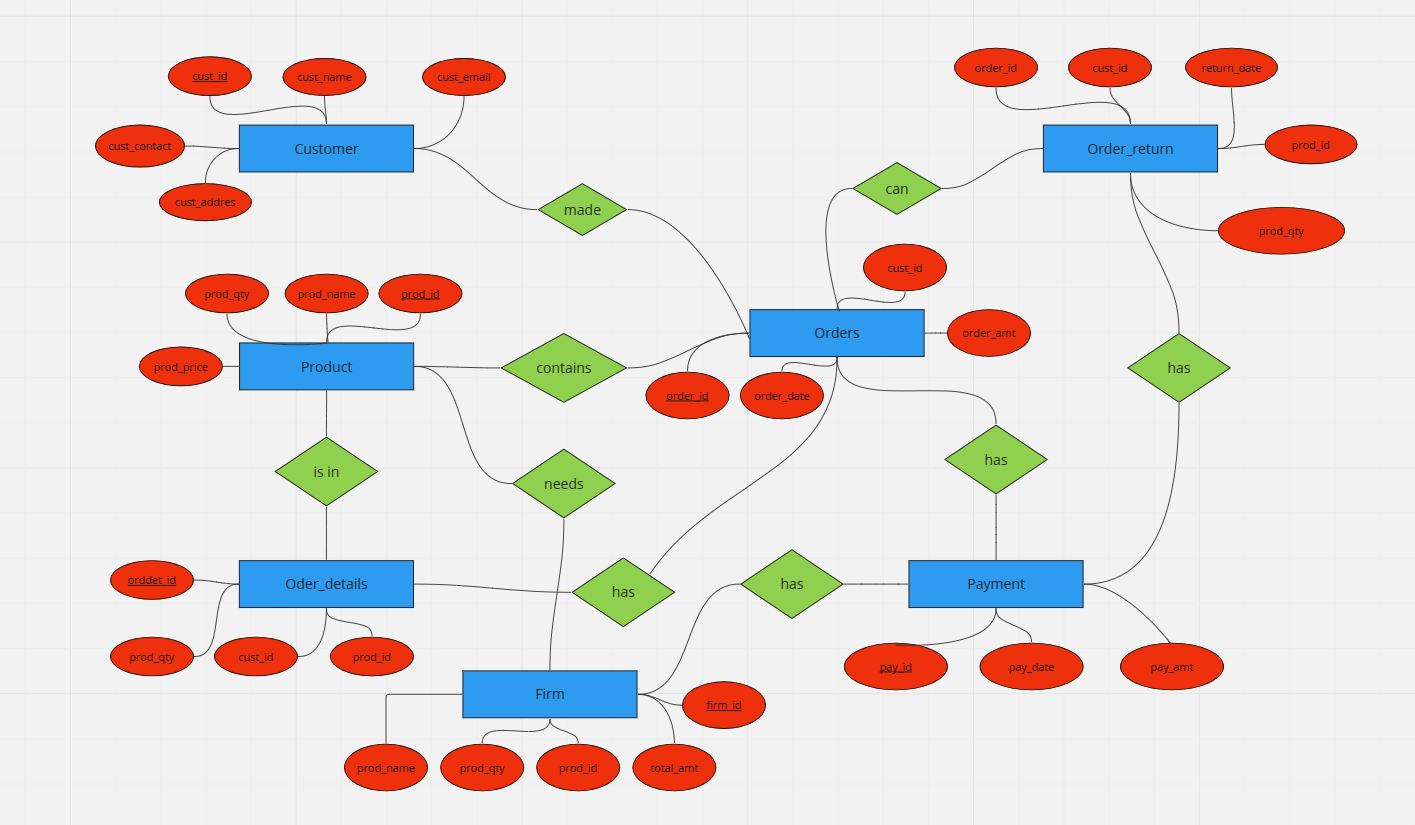
Conclusion

Entity Relationship Design

Tables were created in APEX Oracle



ER diagram



Relationships:

1) Customer places Orders  
One to many

2) Orders contains Product

Many to many

3) Product is in Order\_details  
One to many

4) Orders can Order\_return  
One to one

5) Product needs Firm  
Many to one

6) Orders has Payment  
One to one

7) Order\_return has Payment  
One to one

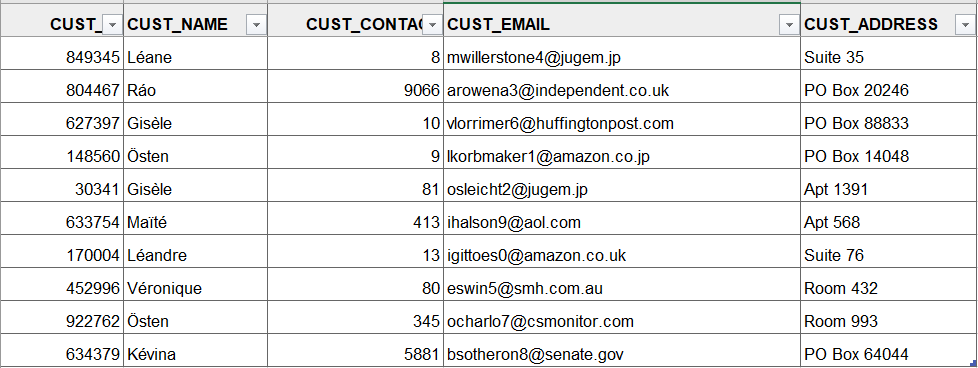
8) Firm has Payment  
One to one

9) Order\_details has Orders  
Many to one

Normalization

So, at the moment we have 7 tables. And my main task is to normalize these tables. That is, to transform them into the form of BCNF. In order to do this, we first need to transform all our tables into 1NF.This means that a table attribute **cannot contain multiple values**. It should only **have an attribute with one value.**

For example, we have a **Customer** table and has the following attributes:



As you can see, the meaning of our attributes is unique. That is, the **cust\_id** attribute has only the ID of the buyer and each of the buyers has only one ID. It's the same with the rest of the attributes.

The same scheme is in the other 6 tables.

Table **Firm:**

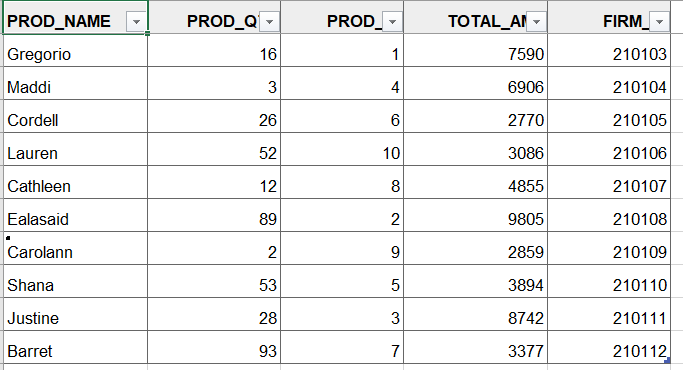


Table **Orders**:

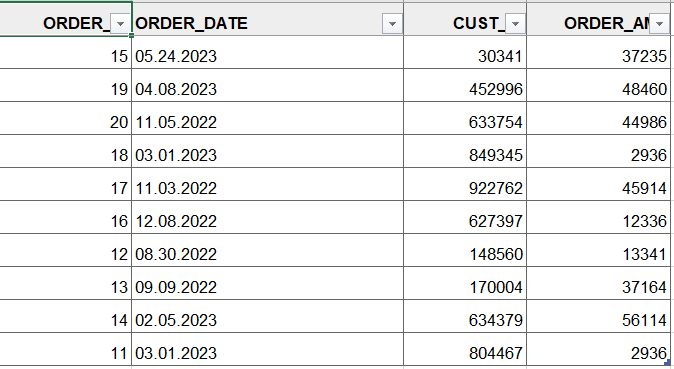


Table **Product**:

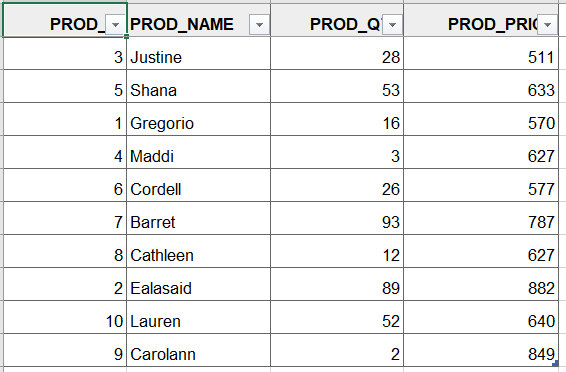
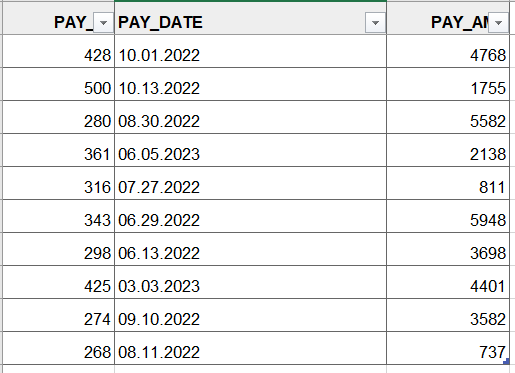
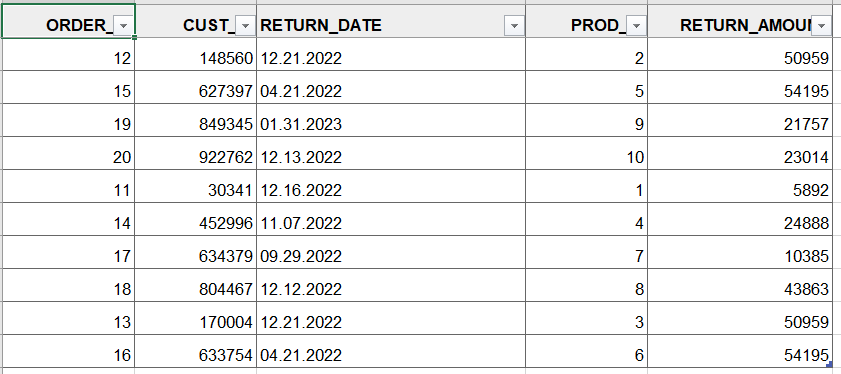


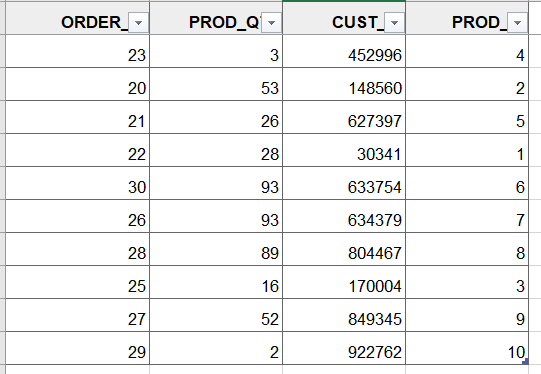
Table **Payment**:



And 2 last tables **Order\_return**:



**Order\_details**:



Now, the main task is to make 1NF -> 2NF. To do this, our table MUST be in the form of 1NF and all non-key attributes are fully functional and depend on the primary key. Simply put, we need to identify the **primary key**. The Customer table has the **cust\_id** attribute, which in turn is unique for each value. That is, we can say that the other attributes like **cust\_name**, **cust\_email**, **cust\_address**, **cust\_contact** are non-key attributes and are completely dependent on **cust\_id** (primary key). **The above table is already in 2NF form.** But, it is possible to divide one large table into 2 small tables.

Scheme 2 table: **cust\_id** -> {**cust\_name**, **cust\_email**}. This is the **first** table.

**cust\_id** -> {**cust\_contact**, **cust\_address**}. And this is the **second** table.

Next table **Orders**.

**order\_id** (primary key) -> {**order\_date**, **cust\_id**, **order\_amt**}

Table **Product**.

**prod\_id** (primary key) -> {**prod\_name**, **prod\_qty**, **prod\_price**}

Table **Payment.**

**pay\_id** (primary key) -> {**pay\_date**, **pay\_amt**}

Table **Order\_details**.

**order\_id** (primary key) -> {**prod\_qty**, **prod\_id**}

Table **Order\_return**.

**order\_id** (primary key) -> { **cust\_id**, **return\_date**, **prod\_id**, **return\_amount**}

Table **Firm**.

**firm\_id** (primary\_key) -> {**prod\_name**, **prod\_qty**, **prod\_id**, **total\_amt**}

Finally we did it! Now, moving on. Next, we have to transform all these 2NF tables into transform into 3NF! How original everything is…

To do this, our tables 1) must be in the form 2NF, 2) Does not contain any transitional partial dependency. A relation is in the third normal form if it fulfills at least one of the following conditions for each nontrivial functional dependence X → Y.

X is a superkey. Y is the main attribute, i.e. each element of Y is part of some candidate key.

A great example would be **table Firm.**

It has the attribute – **total\_amt**, which in turn depends on the primary key – **firm\_id**. But there are also other super keys, for example **prod\_id**, which is also the primary key in **table product**. **prod\_qty**, **prod\_name** are dependent on **prod\_id**. So, it turns out like this: **firm\_id** -> {**total\_amp**, **prod\_id**}/ **prod\_id** -> { **prod\_qty**, **prod\_name**}. **Our table is in the form of 3NF!**

**Table Order\_return**

**order\_id** -> {**return\_date**, **return\_amount, prod\_id**}

**prod\_id** -> {**cust\_id**}

**Table Order\_details**

**order\_id ->** {**prod\_id**}

**prod\_id** -> {**prod\_qty**, **cust\_id**}

**Table Orders**

**order\_id -> {order\_date, order\_amt, cust\_id}**

**cust\_id -> {order\_id}**

**Table Payment**

**pay\_id** -> {**pay\_date**, **pay\_amt**}

**Table Product**

**prod\_id** -> **{prod\_name**, **prod\_qty**, **prod\_price**}

**Table Customer**

**cust\_id** -> {**cust\_name**, **cust\_contact**, **cust\_email**, **cust\_address**}

BCNF is an improved version of 3NF. This is stricter than 3NF.

The table is in BCNF if each functional dependency X → Y, X is a superkey of the table.

Simply put, let's take an example from 3NF. We have converted **table Customer** to 3NF form. Now, we need to improve it to IST. We do this…Just take both keys and add them to a separate table. That's what we have.

**firm\_id -> {total\_amt} 1st table**

**prod\_id -> { prod\_qty**, **prod\_name} 2nd table**

**{ firm\_id, prod\_id} 3rd table**

**Now it's vISTA, because the left side of both functional dependencies is the key.**

**Table Order\_return**

**order\_id** -> {**return\_date**, **return\_amount, prod\_id**} **1st table**

**prod\_id** -> {**cust\_id**} **2nd table**

{ **order\_id**, **prod\_id**} **3rd table**

**Table Order\_details**

**order\_id ->** {**prod\_id**} **1st table**

**prod\_id** -> {**prod\_qty**, **cust\_id**} **2nd table**

{ **order\_id**, **prod\_id**} **3rd table**

**Table Orders**

**order\_id -> {order\_date, order\_amt, cust\_id} 1st table**

**cust\_id -> {order\_id} 2nd table**

{ **order\_id, cust\_id**}

**Table Payment**

**pay\_id** -> {**pay\_date**, **pay\_amt**} **1st table**

**Table Product**

**prod\_id** -> **{prod\_name**, **prod\_qty**, **prod\_price**} **1st table**

**Table Customer**

**cust\_id** -> {**cust\_name**, **cust\_contact**, **cust\_email**, **cust\_address**} **1st table**

Functions & Triggers

Procedures:

**AddCustomer -** is where we store human information in customer:

create or replace PROCEDURE addCustomer(c\_id customer.cust\_id%type,  
c\_name customer.cust\_name%type,  
c\_cont customer.cust\_contact%type,  
c\_ema customer.cust\_email%type,  
c\_add customer.cust\_address%type)  
IS  
BEGIN  
INSERT INTO customer (cust\_id, cust\_name, cust\_contact, cust\_email, cust\_address)  
VALUES(c\_id, c\_name, c\_cont, c\_ema, c\_add);  
END;  
/

**DelCustomer** - this is what we have, destroying a person from a customer:

create or replace PROCEDURE delCustomer(c\_id customer.cust\_id%type) IS  
BEGIN  
DELETE FROM customer  
WHERE cust\_id = c\_id;  
END;  
/

**Get Total**-it calculates the income received to us.

create or replace PROCEDURE gettotal(total\_revenue OUT NUMBER)  
AS  
BEGIN  
  SELECT SUM(pay\_amt) INTO total\_revenue  
  FROM payment;  
END;  
/

Functions:

**prod\_catalog** - this means that we know the price of the product by selecting the customer id.

create or replace FUNCTION prod\_catalog(p\_id product.prod\_id%type)  
RETURN product.prod\_price%type  
IS  
  p\_price product.prod\_name%type;  
BEGIN  
    SELECT prod\_price into p\_price  
    FROM product  
    where prod\_id = p\_id;  
  RETURN p\_price;  
END;  
/

**totalCustomer** - with this we will know how many times the customer has ordered.

create or replace FUNCTION totalCustomer(c\_id order\_details.cust\_id%type)   
RETURN number IS   
   total number;   
BEGIN   
   SELECT count(\*) into total   
   FROM order\_details  
   where cust\_id = c\_id;   
      
   RETURN total;   
END;  
/

Triggers:

**ODT** - after insert order\_details calculate the sum of the received order, gives a certificate of when it was made and sends an order to the company to replenish the product if the product is less than 10.

create or replace trigger "FTT"  
after  
insert on "FIRM"  
for each row  
begin  
    update product set prod\_qty = 50 where product.prod\_id = :new.prod\_id;  
    insert into payment values (:new.total\_amt \* 9, sysdate, 0 - (:new.total\_amt \* 0.9));  
end;  
/

**FTT**-after insert on firm calculates the sum of the order received by the firm and sends it to payment and fills in the product.

create or replace trigger odt  
after insert on ORDER\_DETAILS  
for each row  
DECLARE  
    n number(10);  
    price number(10);  
    name varchar(100);  
begin  
    select prod\_qty into n from product where product.prod\_id = :new.prod\_id; -- prod\_qty ---- dy n-ge tenestiremiz  
    select prod\_price into price from product where product.prod\_id = :new.prod\_id;  
    select prod\_name into name from product where product.prod\_id = :new.prod\_id;  
    update product set prod\_qty = n - :new.prod\_qty where product.prod\_id = :new.prod\_id;  
    insert into payment values (:new.orddet\_id \* 10, sysdate, :new.prod\_qty \* price);  
    if n < 10 then -- 10nan az bolsa firmaga zapros tastaidy  
    insert into firm values (name, 50 - n, :new.prod\_id,((50 - n)\*(0.9 \* price)), 123\*:new.prod\_id); --bizde stabilno 50 zat bolu kerek, sol uwin 50den azaitamyz  
    end if;  
    insert into orders values (:new.orddet\_id, sysdate, :new.cust\_id, price);  
end;  
/

**ORT** - after insert table puts the returned goods in their place, along with the money.

after  
insert or update or delete on "ORDER\_RETURN"  
for each row  
declare  
    price number(10);  
begin  
    select prod\_price into price from product where product.prod\_id = :new.prod\_id; -- prod\_qty dy n-ge tenestiremiz  
    insert into payment values ((price \* :new.prod\_qty) / 10, sysdate, 0 - (price \* :new.prod\_qty));  
    update product set product.prod\_qty = product.prod\_qty + :new.prod\_qty where product.prod\_id = :new.prod\_id;  
end;  
/

Conclusion

As a result of the implementation of the database project for the online store, our team has developed a reliable and efficient data management system that allows you to store, process and manage information about products, customers and orders.

The ER diagram allowed us to understand what data needs to be stored and how they are related to each other. This helped us to determine the correct database structure and correctly distribute information across tables.

Normalization of tables helped to optimize the database and improve its performance. Triggers provide automatic updating of information in the database at certain actions of the client or administrator, which increases the usability of the system.

The database developed by our team allows you to effectively manage online store data, reducing the time and cost of processing orders and improving the quality of customer service. We hope that our work will be useful and successful for the development of our client's business.

Necessary links:

Our slides: (<https://www.canva.com/design/DAFhCDofzwQ/RApcZevSpzFUvS9YGV3vkg/edit>)

Our GitHub link: (<https://github.com/yonusik/Project_DB2.git>)