**DBMS Project**

**Course Name:** Database Management Systems

**Project Title:** Online Shopping Market

**Group Members:** Berdikozha Bayan (210103228)

Akyl Tomiris (210103212)

1. **Description.**
2. Introduction and Database Description.

This is a project for Online Shopping System. The basic idea is that the customers can buy products using online. This project is an attempt to provide the advantages of online shopping to customers of a real shop. It helps buying the products in the shop anywhere through internet by using a device. Additionally, the customer can get the service of online shopping and home delivery from his favorite shop.

An Online Marketplace is an e-commerce site that connects sellers with buyers. It’s often known as an electronic marketplace and all actions are managed by the website owner or admin. Examples of online marketplaces include Amazon, eBay and WildBerries.

Let’s talk about benefits of Online Marketplaces:

- It is a great way for customers to compare prices.

- It allows to your company to function 24/7.

- It is an additional source of revenue.

- Customers are more likely to purchase from an online marketplace with a wide range of options.

- It generates trust between your brand and customers.

1. Our database system have three main functions:
2. Matching Companies, Providers, Sellers and Customers;
3. Allow customers to easily find and sellers to easily promote products;
4. Service customers with actions such as online payment, online ordering and delivery.
5. End Users:

* Site visitors;
* Buyers and shoppers;
* Businessman’s and enterpreneurs;
* Analytics service;
* Deliveryman and drivers;
* Business analytics;
* Advertisers;
* Marketing specialists;
* Investors;
* Large companies;
* SEO content writer;
* Designer;
* Graphic designer;
* IT Technician;
* Digital Operations Manager;
* Director of ecommerce;

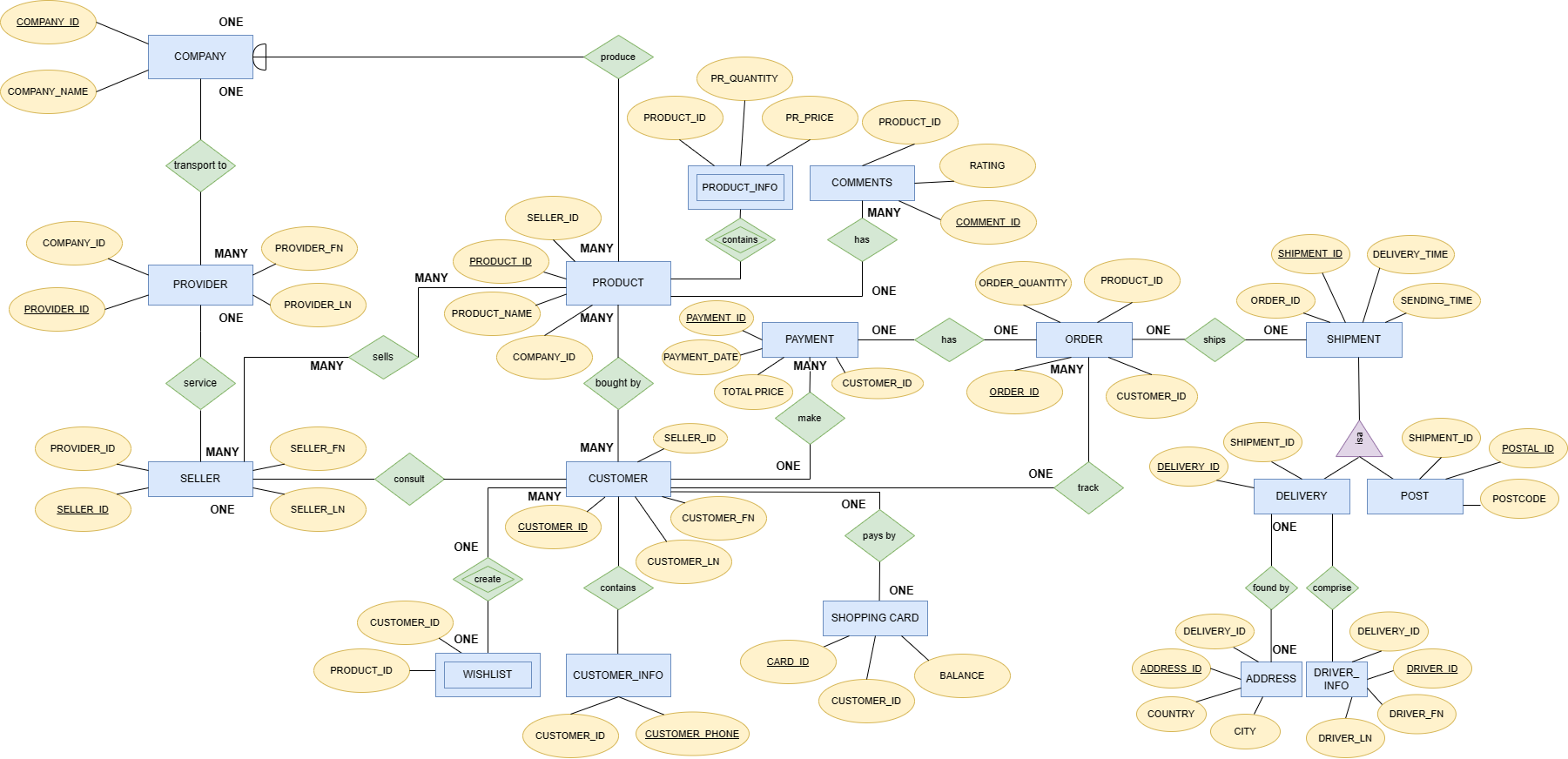
1. How will data obsolescence be handled?

* If the old data is not used in most queries, the best solution would be to separate the tables by a key that distinguishes the old data from the current data (for example, date, currency\_id, etc.). You can then put the stale data into separate tables and databases. It seems to me that partitioning mainly improves performance and simplifies maintenance.
* Separating the operating system and the reporting system. Have one database for online operational system and another for reporting. Then periodically move data from the operating system to the reporting system.
* Also update the old data. But it is indeed a case-by-case decision. You should know your system and the requirements best.

1. We searched for various resources from the Internet by searching on Google, then we formulated and write all the ideas into one thing (project). Also, we did a project with the help of “ask questions”. Our sentences begin with W-words – who, what, where, when, why and also.

**b) Entity Relationship Design:**

i. Entity Relationship Diagram:



ii. The online shopping system is client server-based web application.

Each company should produce at least 1 product. Each provider has one company identified by company\_id. One company may make multiple products. Each Company has Company\_name.

Companies transport products to Providers. Providers has own unique provider\_id, provider\_lastname and provider\_firstname. Then Providers service Sellers so that Sellers can sell products to customers.

Sellers consult Customers on product selection. There are many customers who list their products on their own wishlist.

The shopping site has huge number of products. Each product identified by their product\_ID and has product\_name. Products have comments which consist product\_id and rating.

There are many customers who can create wishlist and list their products with products\_price. After the customer makes his choice, he makes the payment online. Every customer pay with the help of own shopping\_card. After payment, order will be accepted.

Customer can track their order by a unique order\_ID provided by the shopping site. In the order section, each customer can see what products and in what quantity he ordered them.

Each order ships by Shipment. There are 2 categories of Shipment: Post and Delivery. If the Customer chooses Delivery, each order has their own delivery\_id and also information about address and driver (courier). If the Customer chooses Post, each post order identified by postal\_id and with the help of postal code find an address of customer.

iii. Relationships are shown in the er giagram.

**c) Normalization:**

i.

1. Company:

Company\_ID -> Company\_Name

Superkeys: [Company\_ID ], [Company\_ID, Company\_Name]

Key: Company\_ID

2. Provider:

Provider\_ID -> [Company\_ID, Provider\_FN, Provider\_LN]

Superkeys: [Provider\_ID], [Provider\_ID, Company\_ID], [Provider\_ID, Provider\_FN], [Provider\_ID, Provider\_LN], [Provider\_ID, Company\_ID, Provider\_FN], [Provider\_ID, Company\_ID, Provider\_LN], [Provider\_ID, Provider\_FN, Provider\_LN], [Company\_ID, Provider\_FN, Provider\_LN], [Provider\_ID, Provider\_FN, Provider\_LN, Company\_ID].

Key: Provider\_ID

1. Seller:

Seller\_ID -> [Seller\_FN, Seller\_LN, Provider\_ID]

Superkeys: [Seller\_ID], [Seller\_ID, Seller\_FN, [Seller\_ID, Seller\_LN], [Seller\_ID, Provider\_ID], [Seller\_ID, Seller\_FN, Seller\_LN], [Seller\_ID, Seller\_FN, Provider\_ID], [Seller\_FN, Seller\_LN, Provider\_ID], [Seller\_ID, Seller\_LN, Provider\_ID], [Seller\_ID, Seller\_FN, Seller\_LN, Provider\_ID]

Key: Seller\_ID

1. Customer:

Сustomer\_ID -> [Customer\_FN, Customer\_LN, Seller\_ID]

Superkeys: [Customer\_ID], [Customer\_ID, Customer\_FN], [Customer\_ID, Customer\_LN], [Customer\_ID, Customer\_FN, Customer\_LN], [Customer\_ID, Customer\_FN, Seller\_ID], [Customer\_ID, Customer\_LN, Seller\_ID], [Customer\_ID, Customer\_FN, Customer\_LN, Seller\_ID]

Key: Customer\_ID

1. Customer\_Info

Customer\_Phone -> Customer\_ID

Superkeys: [Customer\_Phone], [Customer\_ID, Cust\_phone]

Key: [Customer\_Phone]

1. Wishlist:

Customer\_ID -> [Product\_ID]

Superkeys: [Customer\_ID], [Customer\_ID, Product\_ID]

K[ey: Customer\_ID

1. Product:

Product\_ID -> [Product\_Name, Company\_ID, Seller\_ID]

Superkeys: [Product\_ID], [Product\_ID, Product\_Name], [Product\_ID, Company\_ID], [Product\_ID, Seller\_ID], [Product\_Name, Company\_ID], [Product\_ID, Product\_Name, Company\_ID], [Product\_ID, Product\_Name, Seller\_ID], [Product\_Name, Company\_ID, Seller\_ID], [Product\_ID, Company\_ID, Seller\_ID], [Product\_ID, Product\_name, Company\_ID, Seller\_ID]

Key: Product\_ID

1. Product\_Info:

Product\_ID -> [Pr\_Price, Pr\_Quantity]

Superkeys: [Product\_ID], [Product\_ID, Pr\_Price], [Product\_ID, Pr\_Quantity], [Product\_ID, Pr\_Price, Pr\_Quantity]

Key: Product\_ID

1. Comments:

Comment\_ID -> Product\_ID, Rating

Superkeys: [Comment\_ID], [Comment\_ID, Product\_ID], [Comment\_ID, Rating], [Comment\_ID, Product\_ID, Rating]

Key: Comment\_ID

1. Payment:

Payment\_ID -> [Payment\_Date, Total\_Price, Customer\_ID]

Superkeys: [Payment\_ID], [Payment\_ID, Payment\_Date], [Payment\_ID, Total\_Price], [Payment\_ID, Customer\_ID, Payment\_Date], [Payment\_ID, Customer\_ID, Total\_Price], [Payment\_ID, Payment\_date, Total\_Price], [Payment\_ID, Customer\_ID], [Payment\_ID, Payment\_Date, Total\_Price, Customer\_ID]

Key: Payment\_ID

1. Order:

Order\_ID -> [Product\_ID, Order\_Quantity, Customer\_ID]

Superkeys: [Order\_ID], [Order\_ID, Customer\_ID], [Order\_ID, Product\_ID], [Order\_ID, Order\_Quantity], [Order\_ID, Customer\_ID, Product\_ID], [Order\_ID, Customer\_ID, Order\_Quantity], [Order\_ID, Product\_ID, Order\_Quantity], [Order\_ID, Product\_ID, Order\_Quantity, Customer\_ID]

Key: Order\_ID

1. Shopping Card:

Card\_ID -> [Customer\_ID, Balance]

Superkeys: [Card\_ID], [Card\_ID, Customer\_ID], [Card\_ID, Balance], [Customer\_ID, Balance], [Card\_ID, Customer\_ID, Balance]

Key: Card\_ID

1. Shipment:

Shipment\_ID -> [Order\_ID, Delivery\_time, Sending\_time]

Superkeys: [Shipment\_ID], [Shipment\_ID, Order\_ID], [Shipment\_ID, Delivery\_time], [Shipment\_ID, Sending\_time], [Order\_ID, Delivery\_time], [Order\_ID, Sending\_time], [Shipment\_ID, Order\_ID, Delivery\_time], [Shipment\_ID, Order\_ID, Sending\_time], [Shipment\_ID, Delivery\_time, Sending\_Time], [Order\_ID, Delivery\_time, Sending\_time], [Shipment\_ID, Order\_ID, Delivery\_time, Sending\_time]

Key: Shipment\_ID

1. Delivery:

Delivery\_ID -> Shipment\_ID

Superkeys: [Delivery\_ID], [Delivery\_ID, Shipment\_ID

Key: Delivery\_ID

1. Address:

Address\_ID - > [Delivery\_ID, Country, City]

Superkeys: [Address\_ID], [Address\_ID, Delivery\_ID], [Address\_ID, Country], [Address\_ID, City], [Delivery\_ID, Country], [Delivery\_ID, City], [Address\_ID, Delivery\_ID, Country], [Address\_ID, Country, City] [Address\_ID, Delivery\_ID, City], [Delivery\_ID, Country, City], [Address\_ID, Delivery\_ID, Country, City]

Key: Address\_ID

1. Driver\_Info

Driver\_ID -> [Driver\_FN, Driver\_LN, Delivery\_ID]

Superkeys: [Driver\_ID], [Driver\_ID, Driver\_FN], [Driver\_ID, Driver\_LN], [Driver\_ID, Delivery\_ID], [Delivery\_ID, Driver\_FN], [Delivery\_ID, Driver\_LN], [Driver\_ID, Driver\_FN, Delivery\_ID], [Driver\_ID, Driver\_LN, Delivery\_ID], [Delivery\_ID, Driver\_FN, Driver\_LN], [Driver\_ID, Driver\_FN, Driver\_LN], [Driver\_ID, Driver\_FN, Driver\_LN, Delivery\_ID]

Key: Driver\_ID

1. Post:

Postal\_ID -> Shipment\_ID, Postcode

Superkeys: [Postal\_ID], [Postal\_ID, Shipment\_ID], [Postal\_ID, Postcode],

[Shipment\_ID, Postcode], [Postal\_ID, Shipment\_ID, Postcode]

Key: Postal\_ID

**d. Physical Design.**

All tables: [Tables - Google Drive](https://drive.google.com/drive/folders/1al-1UNVQMbI8I4fqGtDdqXTpYVJannRh)

<https://drive.google.com/drive/folders/1al-1UNVQMbI8I4fqGtDdqXTpYVJannRh?usp=sharing>

**e. Query Part.**

1. Find the sum of total price of all orders of who ordered more than one time and their total price (each order of customer) is greater than 15000.

Select Customer\_ID, Sum(Total\_price) as Total  
From Payment   
Where Total\_Price > 15000   
Group by Customer\_ID  
Having Count(Customer\_ID) > 1

πcustomer\_id, SUM (total\_price) → total  
 σCOUNT (customer\_id) > 1  
  γcustomer\_id, COUNT (customer\_id), SUM (total\_price)  
   σtotal\_price > 15000payment

1. Find all orders issued by the customer with ID ‘305’ with first name is Sidonia and last name is Manneville.

SELECT \*  
FROM orders  
WHERE Customer\_id IN  
    (SELECT Customer\_id   
     FROM Customer   
     WHERE Customer\_fn='Sidonia' and Customer\_ln = 'Manneville');

1. Find the average rating for each product in descending order (shown in the comments table).

Select Product\_ID, AVG(Rating) AS AVGrating  
from Comments  
Group by Product\_ID  
Order by AVG(Rating) DESC

τAVG (rating) ↓  
 πproduct\_id, AVG (rating) → avgrating  
  γproduct\_id, AVG (rating)comments

1. From the Comment table, create a view for those products rated over 5.

Create view Comments\_View AS

Select Comment\_ID, Product\_ID, Rating

From Comment

Where Rating > 5

Select \* from Comment

Change the rating of the product Redmi 10 to 9 in this view table.

Update Comments\_View

Set rating = 9

Where Product\_ID = ‘411’

1. Delete products from the view table Comments\_view with ID 400.

Delete From Comments\_View

Where Product\_ID = 400

1. From the table Comments and view table Comments\_view find union of product and rating where in the table Comments Product\_ID start with number 41 and rating is greater than 6.

SELECT Product\_ID, Rating  
FROM Comments  
WHERE Product\_ID like '41%' and Rating > 6  
UNION   
SELECT Product\_ID, Rating  
FROM Comments\_View

πproduct\_id, rating  
 σproduct\_id LIKE "41%" AND rating > 6comments ∪  
  πproduct\_id, ratingcomments\_view

1. Write a SQL statement to join the tables company, provider and seller so that the same column of each table appears once and only the relational rows are returned.

Select \*

From Seller

Natural Join Company

Natural Join Provider;

seller ⋈ company ⋈ provider

1. Find those shopping cards where the balance exists between 350.000 and 400.000. Return Customers First Name and Last name from Customers.

Select a.Customer\_FN, a.Customer\_LN,

b.Customer\_ID, b.Balance, b.Card\_ID

FROM Customer a, Shopping Card b

WHERE a.Customer\_ID = b.Customer\_ID

AND b.Balance Between 350000 AND 400000

πa . customer\_fn, a . customer\_ln, b . customer\_id, b . balance, b . card\_id  
 σa . customer\_id = b . customer\_id AND (350000 <= b . balance AND b . balance <= 400000)  
  (ρacustomer ×  
   ρb  
    ρcardshopping)

1. From provider and seller tables display the provider first name, provider last name. The results should be sorted by ascending provider\_id.

SELECT a.Provider\_FN, a.Provider\_LN,

b.Seller\_FN, b.Seller\_LN FROM Provider a LEFT JOIN Seller b

ON a.Provider\_ID = b.Provider\_id order by a.Provider\_ID;

1. Find those customers with return minimum quantity of order from table Order.

Select Customer\_ID, Min(Order\_Quantity) as minOrderQuantity

From Orders

Group by Customer\_ID

Order by Min(Order\_Quantity)

τMIN (order\_quantity)  
 πcustomer\_id, MIN (order\_quantity) → minorderquantity  
  γcustomer\_id, MIN (order\_quantity)orders

1. Find the Card Numbers that have minimum and maximum balance on the card.

Select Min(Balance) as minBalance, Max(Balance) as maxBalance

From Shopping\_Card

πMIN (balance) → minbalance, MAX (balance) → maxbalance  
 γMIN (balance), MAX (balance)shopping\_card

1. Add rating 3 to the product with ID 401.

Insert into Comments (Comment\_ID, Product\_ID, Rating)

Values (1761, 401, 3);

1. Change the name of column Postal\_ID to PostalID in table Post.

Alter Table Post

Rename Column Postal\_ID TO Post\_ID

1. Change the name of table Customer\_INFO to Customer\_Details.

ALTER TABLE Customer\_INFO

RENAME TO Customer\_Details;

1. Create a NOT NULL constraint on the Balance column.

ALTER TABLE Shopping\_Card  
ALTER COLUMN ‘Balance’ ‘Balance’ int NOT NULL

1. Use the EXCEPT statement which will return all the records from the Product\_INFO table where the quantity is less than or equal to 100:

SELECT Product\_ID, Pr\_Quantity, Pr\_Price FROM Product\_INFO

Except

SELECT Product\_ID, Pr\_Quantity, Pr\_Price FROM Product\_INFO

WHERE Pr\_Quantity > 100

πproduct\_id, pr\_quantity, pr\_priceproduct\_info ∖  
 πproduct\_id, pr\_quantity, pr\_price  
  σpr\_quantity > 100product\_info

1. Create a view table from the Product\_Info table where Product Price start with number 18.

Create view Pr\_Info\_View AS

Select Product\_ID, Pr\_Price

From Product\_INFO

Where Pr\_Price like ‘18%’

Intersect Prices from two table Product\_Info and Pr\_Info\_View.

Select Pr\_Price

From Product\_Info

Intersect

Select Pr\_Price

From Pr\_Info\_View

Order by Pr\_Price DESC

πpr\_priceproduct\_info ∩  
 τpr\_price ↓  
  πpr\_pricepr\_info\_view

1. Create a trigger when Comment\_ID will be 1701 and Product\_ID will be 405, then rating with Comment\_ID 1702 will be less by 1 and Product\_ID will be 406. (INSERT)

create or replace trigger "COMMENTS\_T1"

AFTER

insert on "COMMENTS"

for each row

declare pragma autonomous\_transaction;

begin

if :new.COMMENT\_ID = 1701 then

insert into COMMENTS(COMMENT\_ID, PRODUCT\_ID, RATING) values (1702, :new.PRODUCT\_ID + 1, :new.Rating - 1);

commit;

END if;

end;

1. Create a trigger that will change the price of the product with ID 405 by the same amount as product with ID 400. (UPDATE)

create or replace trigger "PRODUCT\_INFO\_T1"  
AFTER  
update on "PRODUCT\_INFO"  
for each row  
declare   
pragma autonomous\_transaction;  
begin  
    if(:NEW.PRODUCT\_ID = 400 ) then  
        UPDATE PRODUCT\_INFO SET PR\_PRICE = PR\_PRICE -(:OLD.PR\_PRICE - :NEW.PR\_PRICE) WHERE PRODUCT\_ID = 405;  
        commit;  
    end if;  
end;

1. If the seller with ID 201 refuses to sell a product with the name Samsung j5, then the seller with ID 202 will sell this product.

create or replace trigger "PRODUCT\_T1"  
AFTER  
delete on "PRODUCT"  
for each row  
declare   
pragma autonomous\_transaction;  
begin  
    if(:old.PRODUCT\_NAME = 'Samsung j5' ) then  
        Insert into PRODUCT(PRODUCT\_ID, PRODUCT\_NAME, COMPANY\_ID, SELLER\_ID) values (:old.PRODUCT\_ID + 1, :old.PRODUCT\_NAME, :old.COMPANY\_ID, :old.SELLER\_ID + 1);  
        commit;  
    end if;  
end;

**Transaction and index:**

* Create a transaction that insert datas in the table Product, then in the table Product\_Info.

START TRANSACTION;  
SELECT   
    @Product\_ID:=MAX(Product\_ID) + 1  
FROM product;  
INSERT INTO product (Product\_ID, Product\_name, Company\_ID, Seller\_ID)  
VALUES (@Product\_ID, 'Iphone7', 2 , 205);  
  
INSERT INTO product\_info (Product\_ID, Pr\_quantity, Pr\_price)  
VALUES (@Product\_ID, 200, 120000);  
COMMIT;

* Create index Pr\_Index:

create Unique index Pr\_index on Product(Product\_ID);