GTU Department of Computer Engineering
CSE 411 – Spring 2023
Database Project

Hasan Mutlu
1801042673

Problem Definition

This project is about online shopping. There is a business which sells its own products on the internet. This project focuses on establishing a database for such a company. The database isn't built for a general shopping website like Hepsiburada, Trendyol etc. Instead, it can be used by individual stores' shopping sites.

User Requirements

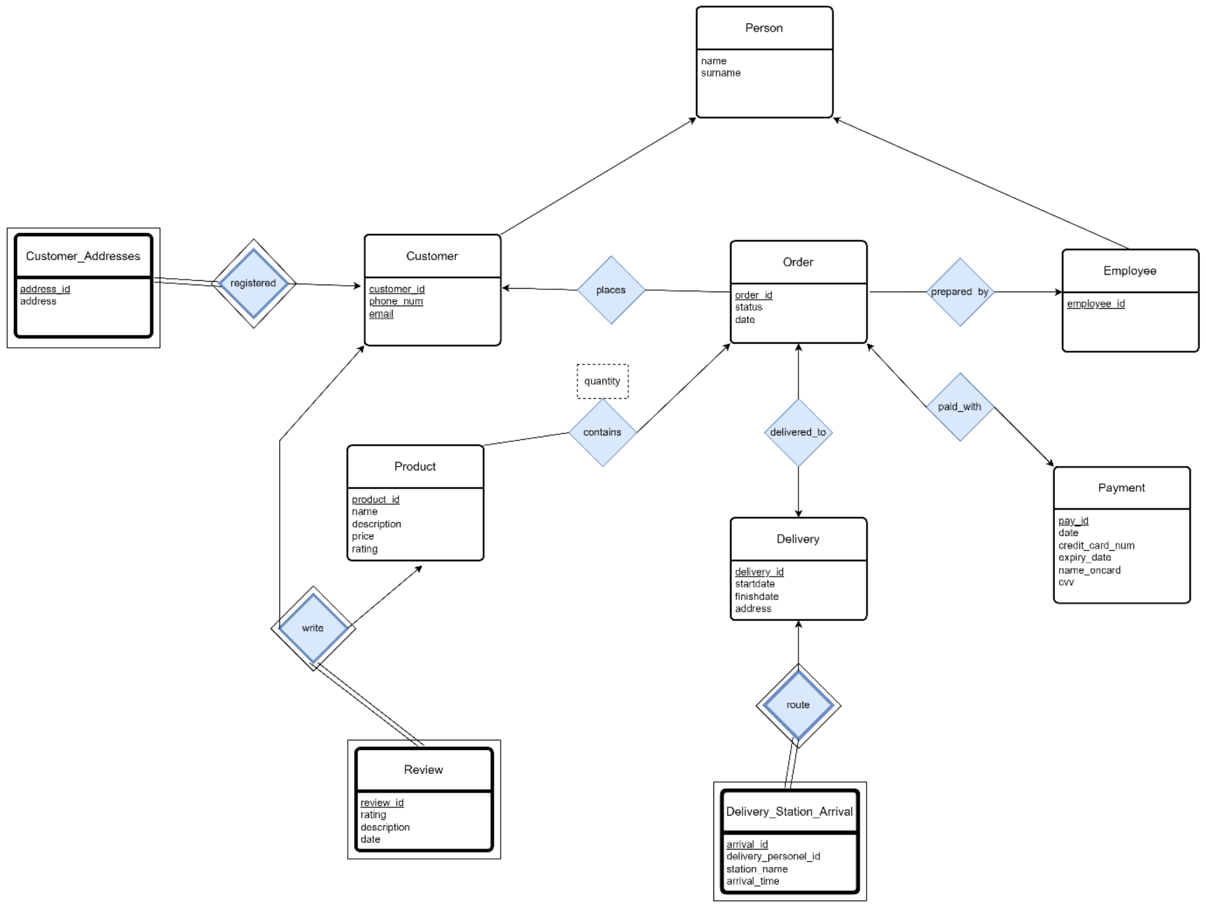
- **Placing Order:** Customers should be able to place order.
- **Saving Addresses:** Customers should be able to save multiple addresses of their own.
- **Writing Review:** Customers should be able to place reviews to the products they bought.
- **Order Ingredients:** Orders should allow to contain multiple products of different quantities.
- **Product Rating:** Products should have reviews. A rating should be calculated out of its reviews.
- **Delivery Recording:** Orders' deliveries should be recorded.
- **Delivery Status Recording:** Delivery status should be recorded in each step of delivery. The last place of the delivery, delivering personel's id from the external delivery company should be recorded.

- ➔ **Payment Method:** Orders should be paid with a credit card. Credit card information of each customers' musn't be recorded.
- ➔ **Responsible Employee:** The employee who has prepared the order should be recorded. In case of complaint, the employee can be found. The employee data isn't supposed to be kept fully, it comes from an external employee database.

System Requirements

- ➔ **SQL:** Used to implement the database.
- ➔ **MySQL:** Used to create database server.
- ➔ **Python:** Used to implement user program.
- ➔ **Pymysql Package:** Used to connect to SQL server from user program.

ER Diagram



Weak Entities:

- ➔ Customer_Address
- ➔ Review
- ➔ Delivery_Station_Arrival

Roles: Roles are pretty straight-forward so aren't re-defined.

Ternary Relationship: "write" relationship between Customer, Product and Revies is ternary.

Inheritances: Customer and Employee inherit from Person Enitity.

Functional Dependencies:

Customer:
customer_id -> name, surname, phone_num, email

Address:
Address_id -> address

Order:
Order_id -> status, date

Employee:
Employee_id -> name, surname

Product:
Product_id -> name, description, price, rating

Review:
Review_id -> rating, description, date

Delivery:
Delivery_id -> startdate, finishdate, address

Delivery_station_Arrival:
Arrival_id -> delivery_personel_id, station_name, arrival_time

Payment:
Pay_id -> date, credit_card_number, expiry_date, name_oncard, cvv

Normalization

1NF:

All relations are in 1NF since all of them contain atomic values. There is no multi-valued attributes.

2NF:

All relations are in 2NF since all non-key attributes are fully functional dependent on the primary key.

3NF:

All relations are in 3NF because there is no partial transitive dependencies.

BCNF or 3NF?

All relations are also in BCNF because for all functional dependencies in F+, left hand side of the dependency (determinant) is superkey.

For example:

Delivery:

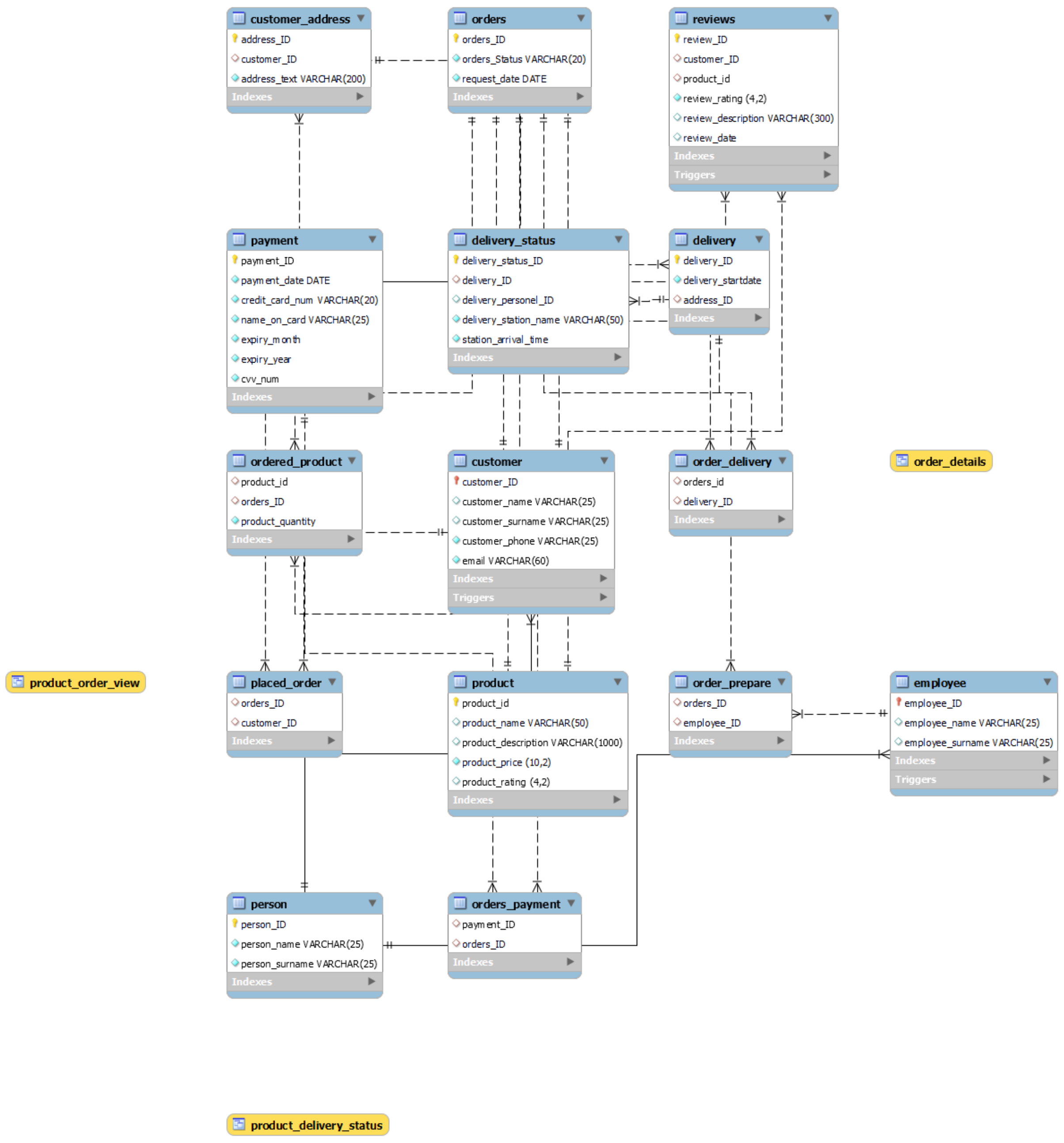
Delivery_id -> startdate, finishdate, address

Delivery_id is superkey.

This situation applies to all tables.

Tables in SQL

After implementing the ER diagram in SQL, the tables and the attributes in them are implemented as follows:



Views

Several views are implemented in the database.

Order Details View

```sql
CREATE VIEW order_details AS
SELECT po.orders_ID, po.customer_ID, op.product_id, op.product_quantity
FROM placed_order po
JOIN ordered_product op ON po.orders_ID = op.orders_ID;
```

It is used to get the customer and ingredients of an order, used as a helper in one of the functions.

Ordered Products View

```sql
CREATE VIEW product_order_view AS
SELECT op.orders_ID, p.product_name, op.product_quantity, p.product_price
FROM product p
JOIN ordered_product op ON p.product_id = op.product_id;
```

It is used to get the product ingredients of orders, used to show data to the user.

Product Delivery Status View

```sql
CREATE VIEW product_delivery_status AS
SELECT op.product_id, ds.delivery_status_ID, ds.delivery_station_name,
ds.station_arrival_time
FROM ordered_product op
JOIN orders o ON op.orders_ID = o.orders_ID
JOIN order_delivery od ON o.orders_ID = od.orders_ID
JOIN delivery d ON od.delivery_ID = d.delivery_ID
JOIN delivery_status ds ON d.delivery_ID = ds.delivery_ID;
```

It is used to see the status of the delivery, where it is last located.

Functions

Several functions are implemented for the

Total Price Function

```sql
CREATE FUNCTION calculate_total_price(order_id INT) RETURNS NUMERIC(10, 2)
DETERMINISTIC
BEGIN
    DECLARE total_price NUMERIC(10, 2);

    SELECT SUM(op.product_quantity * p.product_price)
    INTO total_price
    FROM ordered_product op
    INNER JOIN product p ON op.product_id = p.product_id
    WHERE op.orders_ID = order_id;

    RETURN total_price;
END
```

It is used to calculate the total price of an order.

Total Revenue Function

```sql
CREATE FUNCTION calculate_total_revenue() RETURNS NUMERIC(10, 2)
READS SQL DATA
DETERMINISTIC
BEGIN
    DECLARE total_revenue NUMERIC(10, 2);

    SELECT SUM(op.product_quantity * p.product_price)
    INTO total_revenue
    FROM ordered_product op
    INNER JOIN product p ON op.product_id = p.product_id;

    RETURN total_revenue;
END//
READS SQL DATA
DETERMINISTIC
```

Calculates the total revenue of the store.

Triggers

Several triggers are implemented for the database.

Create Person Trigger

```
CREATE TRIGGER employee_insert_trigger
BEFORE INSERT ON employee
FOR EACH ROW
BEGIN
    INSERT INTO person (person_ID, person_name, person_surname)
    VALUES (NEW.employee_ID, NEW.employee_name, NEW.employee_surname);
END//

CREATE TRIGGER customer_insert_trigger
BEFORE INSERT ON customer
FOR EACH ROW
BEGIN
    INSERT INTO person (person_ID, person_name, person_surname)
    VALUES (NEW.customer_ID, NEW.customer_name, NEW.customer_surname);
END//
```

While creating a new customer or employee, its inherited data fields are firstly created in the Person table.

Update Product Rating Triggers

```
CREATE TRIGGER update_product_rating
AFTER INSERT ON reviews
FOR EACH ROW
BEGIN
    DECLARE avg_rating NUMERIC(4,2);

    SELECT AVG(review_rating) INTO avg_rating
    FROM reviews
    WHERE product_id = NEW.product_id;

    UPDATE product
    SET product_rating = avg_rating
    WHERE product_id = NEW.product_id;
END //
```

```
CREATE TRIGGER delete_review_update_product_rating
AFTER DELETE ON reviews
FOR EACH ROW
BEGIN
    DECLARE avg_rating NUMERIC(4, 2);

    SELECT AVG(review_rating) INTO avg_rating
```

```sql
    FROM reviews
    WHERE product_id = OLD.product_id;

    IF avg_rating IS NULL THEN
        SET avg_rating = NULL;
    END IF;

    UPDATE product
    SET product_rating = avg_rating
    WHERE product_id = OLD.product_id;
END //
```

When a new review is placed or an exising review is deleted on a product, the product's rating is re-calculated and updated.

Restrict Review Placement

```sql
CREATE TRIGGER restrict_review_placement
BEFORE INSERT ON reviews
FOR EACH ROW
BEGIN
    DECLARE order_exists INT;

    SELECT COUNT(*) INTO order_exists
    FROM order_details
    WHERE customer_ID = NEW.customer_ID AND product_id = NEW.product_id;

    IF order_exists = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'You can only place a
review for products you have ordered.';
    END IF;
END//
```

Before a new review is inserted, check if the reviewing customer have purchased the reviewed product. If not, don't allow the review to be inserted.