**GTU Department of Computer Engineering**

**CSE414 Databases - Spring 2022**

**Project Report**

**Akif KARTAL**

**171044098**

**1) Problem Definition**

****

The problem is to make a database management systemforthe **yemeksepeti** platform.

**2) Solution**

The project was **finished as expected** in project assignment announcement.

**2.1) User Requirements**

In this project we 3 users. These are customer, restaurant owner and admin of system.

1. **Customer** shall be able to sign up.
2. **Customer** shall be able to add food to his/her basket.
3. **Customer** shall be able to order food.
4. **Customer** shall be able to see status of the order and old orders.
5. **Customer** shall be able to add new address or edit them.
6. **Customer** shall be able to see all foods and restaurants.
7. **Customer** shall be able to add restaurant in his/her favorites.
8. **Customer** shall be able to load money on the digital wallet.
9. **Customer** shall be able to make review to order.
10. **Customer** shall be able to see and use his/her discount coupons.
11. **Restaurant Owner** shall be able to add new food on his/her menu or edit them.
12. **Restaurant Owner** shall be able to add new payment type for his/her restaurant.
13. **Restaurant Owner** shall be able to change working hours of restaurant.
14. **Restaurant Owner** shall be able to add new campaign for foods.
15. **Admin** shall be able to add new restaurants or edit them.
16. **Admin** shall be able to add new discount coupon to the customers
17. **Admin** shall be able to see all information in the system.

**2.2) Database Tables and Relations**

In this project we have **25 table**. These tables are followings.

* Customer
* Address
* AddressType
* CustomerAdress
* TelephoneNumber
* Email
* Orders
* FavouriteRestaurant
* Basket
* BasketFood
* OrderFood
* DigitalWallet
* Restaurant
* PaymentType
* PaymentRestaurant
* WorkingHours
* Food
* Category
* Review
* Campaign
* Coupon
* CouponCustomer
* FoodLog
* WalletLog
* Offer

**Relations**

Between these tables we have following relations;

* **4** one to one relation
* **20** one to many relation
* **6** many to many relation

Total we have **30** relations. You can see in the E-R diagram.

**Creating Tables and Inserting Data**

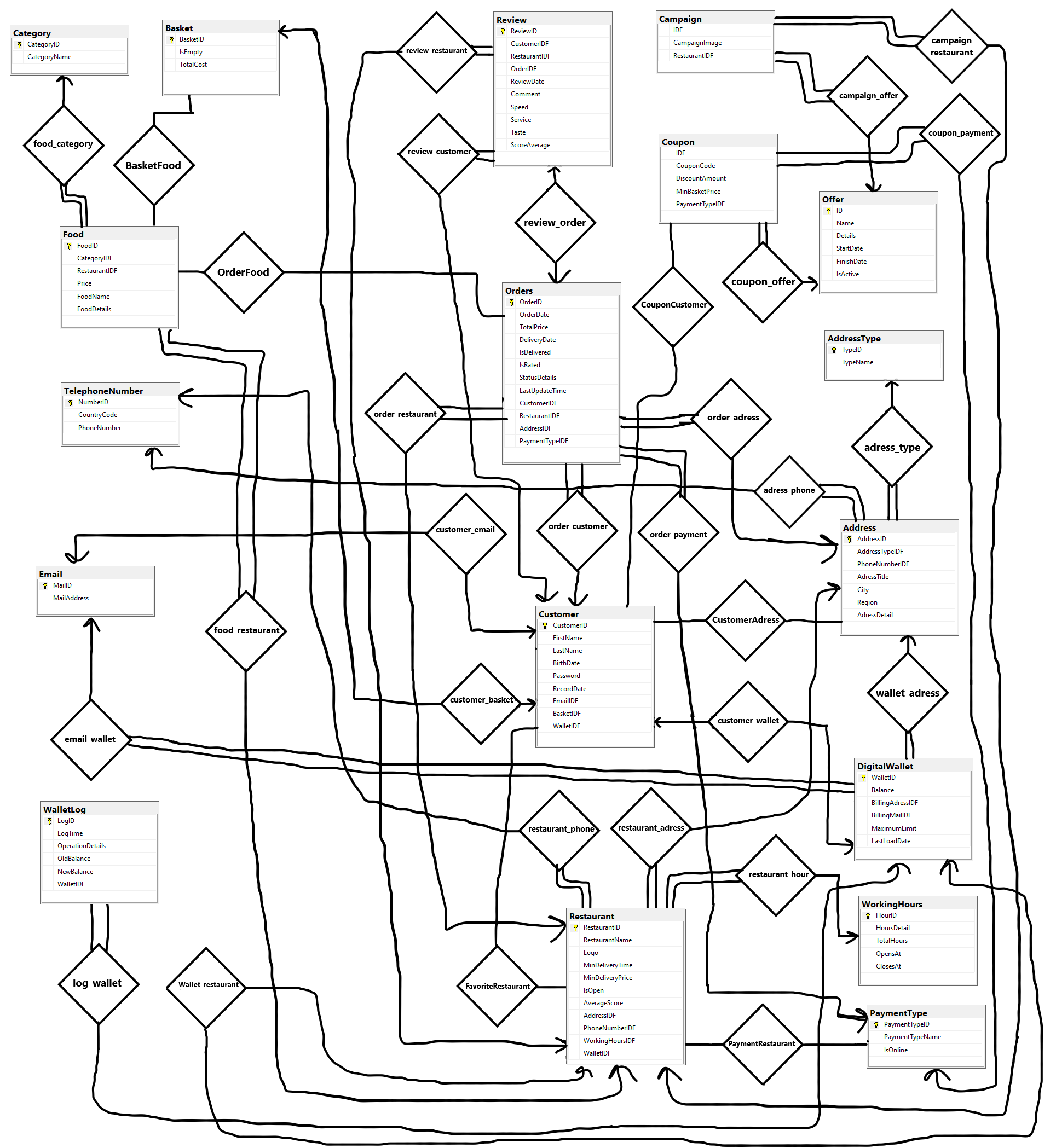
In this database everything made with SQL language. You can see in following images.

Graphical user interface, text, application

Description automatically generated Graphical user interface, text

Description automatically generated

**2.3) E-R Diagram**



**2.4) Normalization**

As you can see from the tables I have applied normalization such that **tables are well separated** and **no data is repeated**. Also, there are no partial and transitive dependency. Lastly all my primary keys are candidate key. Therefore, **in this project I have 3NF and BCNF normalization.** For example, we have **WorkingHours** table for restaurant. This is example of 3NF and BCNF normalization.

**2.5) Functional Dependencies**

**Customer**

CustomerID FirstName, LastName, Password, BirthDate, EmailIDF, BasketIDF, WalletIDF, RecordDate

**Address**

AddressID PhoneNumberIDF, AddressTypeIDF, AdressTitle, City, Region, AdressDetail

**AddressType**

TypeID TypeName

**TelephoneNumber**

NumberID CountryCode, PhoneNumber

**Email**

MailID MailAddress

**Orders**

OrderID CustomerIDF, RestaurantIDF, AddressIDF, PaymentTypeIDF, StatusDetail, LastUpdateTime, OrderDate, TotalPrice, IsDelivered, IsRated, DeliveryDate

**Basket**

BasketID IsEmpty, TotalCost

**DigitalWallet**

WalletID Balance, BillingAdressIDF, BillingMailIDF, MaximumLimit, LastLoadDate

**Restaurant**

RestaurantID RestaurantName, Logo, AverageScore, MinDeliveryTime, MinDeliveryPrice, IsOpen, WorkingHoursIDF, AddressIDF, PhoneNumberIDF

**PaymentType**

PaymentTypeID PaymentTypeName, IsOnline

**WorkingHours**

HourID HoursDetail, TotalHour, OpensAt, ClosesAt

**Food**

FoodID FoodName, FoodDetail, Price, CategoryIDF, RestaurantIDF

**Category**

CategoryID CategoryName

**Review**

ReviewID CustomerIDF, RestaurantIDF, Comment, ReviewDate, Speed, Service, Taste, ScoreAverage, OrderIDF

**Campaign**

IDF CampaignImage, RestaurantID

**Coupon**

IDF CouponCode, DiscountAmount, MinBasketPrice, PaymentTypeIDF

**FoodLog**

LogID DeletedTime, FoodName, FoodDetail, Price, RestaurantName

**WalletLog**

LogID LogTime, OperationDetails, OldBalance, NewBalance, WalletIDF

**Offer**

ID Name, Details, StartDate, FinishDate, IsActive

**2.6) Triggers**

1. **trg\_LogWallet :** By using this trigger we will **insert a new log** to WalletLog after updating DigitalWallet. In this way we can see the user operations in any problem. You can see the changes in WalletLog table in following image.

Graphical user interface, text, application

Description automatically generated

1. **trg\_ReviewAverage:** By using this trigger after a new review(speed,service,taste) made by the customer **we can update restaurant average score**. You can see the changes in Restaurant table in following image.

**Graphical user interface, text, application, email

Description automatically generated**

1. **trg\_BasketPriceUpdate:** By using this trigger **we will update basket total price** after a food added to the basket. You can see the changes in Basket table in following image.

Graphical user interface, text, application

Description automatically generated

1. **trg\_LogFood:** By using this trigger, **after deleting a food we can insert information** about that food in FoodLog table to check later. You can see the changes in FoodLog table in following image.

Graphical user interface, text, application, email

Description automatically generated

1. **trg\_OrderFood :** By using this trigger, when a **customer give the order(insert),** the foods in the **basket will be removed and they will added to order table**. You can see the changes in Basket table in following image.

Graphical user interface, text, application, email

Description automatically generated

**2.7) Views**

**1. vm\_Customer:** By using this view **we can see the customer information detailed way(joined).** Normally customer information separated into different tables. With this view we will join them. See the results in following image.

A screenshot of a computer

Description automatically generated with medium confidence

**2. vm\_Order :** By using this view we can see the order information detailed way. Normally order information separated into different tables. With this view we will join them and add additional information. See the results in following image.

A screenshot of a computer

Description automatically generated

**3. vm\_Max40Food:** By using this view we can see the foods whose price less than 40 TL. See the results in following image.

Graphical user interface, text

Description automatically generated with medium confidence

**4. vm\_Restaurant:** By using this view we can see the restaurant information detailed way. Normally restaurant information separated into different tables. With this view we will join them and add additional information. See the results in following image.

Application, table

Description automatically generated

**5. vm\_Comment:** By using this view we can see the review information detailed way. Normally review information separated into different tables. With this view we will join them and add additional information. See the results in following image.

Table

Description automatically generated

**2.8) Left, Right and Full Outer Join**

Since Customer and Email tables are relational tables, we can join them by using following queries.

* select \* from Customer c Left Outer Join Email e on c.EmailIDF = e.MailID
* select \* from Customer c Right Outer Join Email e on c.EmailIDF = e.MailID
* select \* from Customer c Full Outer Join Email e on c.EmailIDF = e.MailID

**2.9) Transactions**

**1. OrderPayment**

When a customer gives an order, if the customer's balance is enough to give the order we transfer the order price from the customer's digital wallet to the restaurant's digital wallet(account).

Graphical user interface, text, application, email

Description automatically generated

**2. insertCustomer**

While inserting a customer if the given email, wallet, or basket is already used we will not insert the customer, otherwise customer will be inserted.

**3. insertReview**

A new review cannot be inserted for an order that has been reviewed before. Otherwise, it will be added.

**Graphical user interface, text, application, email

Description automatically generated**

**2.10) Inheritance (Specialization)**

Between Campaign and Coupon tables there is an inheritance because both are related to discount, and they have common attributes. Therefore, they inherited from Offer table.

Graphical user interface, application, Teams

Description automatically generated

As you can see Name, Details, StartDate, FinishDate and IsActive attributes are **common** between Campaign and Coupon tables and IDF’s are foreign key. We will access the information by using this foreign key.

**2.11 Extra Details**

**1. Using Store Procedure**

In order to implement transaction and insertion operation efficient way**, I have used store procedures**. You can see implementation and usage in following images.

Graphical user interface, text, application

Description automatically generated



Table

Description automatically generated with low confidence

**2. Using Inner Joins**

In views, I have used inner joins you can see in following image.

Text

Description automatically generated

**3. Using Check Constraint in Create Table query**

In Create Table step**, I have used check constraint.** You can see in following image.

Graphical user interface, application

Description automatically generated

**4. Using Case-When-Then, Declare and Automatic Attribute**

In triggers and create table queries I have used SQL language case-when-then, declare and cast operation. You can see in following images.

**Text

Description automatically generated**

**Graphical user interface, text

Description automatically generated**

**2.12 User Interface (GUI)**

A picture containing graphical user interface

Description automatically generated