  
CS 353 Database Systems  
Project Proposal  
  
Project Name: Restaurant Express

**Online Restaurant Order System**  
Group 35

Semih Teker - 21300964

Halil İbrahim Çavdar - 21501613

Yılmaz Berkay Beken – 21400294

Bahadır Durmaz - 21502098

[**1 Revised E/R Model**](#_q6y23falblgh) **3**

[**2 Relation Schemas**](#_12f9l3ce93xb) **5**

[2.1 user](#_3znysh7) 6

[2.2 customer](#_2et92p0) 7

[2.3 restaurant](#_tyjcwt) 8

[2.4 customer\_address](#_3dy6vkm) 9

[2.5 district](#_1t3h5sf) 10

[2.6 order](#_4d34og8) 11

[2.7 delivery\_staff](#_2s8eyo1) 12

[2.8 product](#_98m5cuies52l) 13

[2.9 ingredient](#_17dp8vu) 14

[2.10 contains](#_7wnxl2ktxc46) 15

[2.11 consist\_of](#_3rdcrjn) 16

[2.12 staff\_serves](#_26in1rg) 17

[**3 Functional Dependencies and Normalization of Tables**](#_3j2qqm3) **17**

[**4 Functional Components**](#_rakor83tx1z2) **18**

[4.1 Use Cases / Scenarios](#_2q9b8frsrvly) 18

[4.2 Algorithms](#_ybn9fiwt0i3a) 22

[4.2.1 Restaurant-related Algorithms](#_450ud8wybc6i) 22

[4.2.3 Order and Delivery-Related Algorithms](#_68wpdko8bf9f) 22

[4.2.4 Customer-Related Algorithms](#_b0548o66pg4q) 22

[**5 User Interface Design and Corresponding SQL Statements**](#_k3d2xuauyb39) **24**

[5.1 Login](#_ywuoxdl66xi7) 24

[5.2 Customer Register](#_admck9mgxth6) 25

[5.3 Restaurant Login](#_p0ghaxs76oto) 26

[5.4 Home Page](#_p7cq5xe66ahz) 27

[5.5 Adjustment](#_f8hl86nitxvh) 29

[5.6 Add new Address](#_hmq51uxef5dw) 31

[5.7 Edit Address Info](#_g4zl1bcv4hhk) 33

[5.8 List Restaurants](#_izkgu39eeyqy) 35

[5.9 Search](#_2yzzzp8cd776) 35

[5.10 List Menus of the Restaurant](#_b7djohkrndoj) 38

[5.11 Cart](#_5l9uqe362smn) 40

[5.12 Restaurant Page](#_npjtx2zc4h6m) 41

[5.13 Add new Product / Edit Products](#_lpmevxuyc2kv) 43

[5.14 Restaurant Info and Rate](#_85fzwtlvkvsx) 45

[**6 Advanced Database Components**](#_nmf14n) **46**

[6.1 Views](#_ov30fxvc3wpj) 46

[6.1.1 Customer view of Order](#_2y0bbe8atnxx) 46

[6.1.2 Restaurant view of Order](#_atwy96h7f85m) 46

[6.2 Reports](#_42rwtt9yebcb) 47

[6.2.1 Number of orders of each food](#_lotqxyrj199w) 47

[6.2.2 Number of orders of each customer](#_5r65gw8uvoij) 47

[6.2.3 Amount of money from one customer](#_1ubzouklqq32) 48

[6.3 Triggers](#_8q1ho0ehjfmf) 48

[6.4 Constraints](#_sqyw64) 48

[6.5 Stored Procedures](#_m78plc5wdrm7) 49

[**7 Implementation Plan**](#_3cqmetx) **49**

[**8 Website**](#_1rvwp1q) **50**

# 1 Revised E/R Model

According to assistant's review, we revised our E/R model considering the feedback as follows:

* We added the IsA relation for “user”, we changed the name of “user” entity with “customer” and we added the “user” entity, now “customer” and “restaurant” entities inherits “user” entity.
* We changed the name of the “menu” entity with “product” to avoid the misunderstandings.
* We added “consist\_of” relation between the “order” and “product” entities.
* Instead of “notes” and “cancels” relations between “customer” and “order” entities, we combine these as attributes of the “makes” relation.
* As a special feature, to show the total calories of the menus, we add the “ingredient” entity.
* Some cardinality and total participation problems fixed.
* Some words are replacing more clear ones such as address\_type is changed with address\_tittle for “customer\_address” entity and area is changed with city for “district” entity.

We also made the following changes in our E/R Diagram:

* We removed the “pays” relation between the “customer” and “delivery\_staff”.
* We removed the “comments” relation between the “customer” and “restaurant”. Instead of it, we added comments as attributes of the “makes” relation.
* We remove the credit\_card\_information of the customer and in addition name attribute, we added surname attribute too.
* The source and destination addresses were kept in “order” entity instead of “delivery\_staff” entity.
* Redundant delivery\_time() function in the “delivery\_staff” entity was removed.
* phone\_number and address\_description were added into “customer\_address” entity.
* We added the calorie and portion attributes to “product” entity.
* We added isopen(), rest\_IBAN and most\_favorite\_foods() functions-attributes to “restaurant” entity.

# 2 Relation Schemas

* user (u\_id, e\_mail, password)
* customer(u\_id, e\_mail, password, name, surname, {phone\_number})
* restaurant(u\_id, email, password, name, address, {rest\_phone}, working\_hours, isOpen, rate, d\_id)
* customer\_address(u\_id, address\_title, name, surname, phone\_number, street\_name, street\_number, city, zipcode, adress\_desc, d\_id)
* district(d\_id, district\_name, area, city)
* order(ord\_id, date, order\_notes, source\_address, dest\_address, pay\_type, comment, rate, cust\_id, rest\_id, staff\_id)
* delivery\_staff(staff\_id, staff\_name)
* product(rest\_id, food\_name, food\_type, price, portion)
* ingredient(ingr\_name, ingr\_calorie, quantity)
* contains(rest\_id, food\_name, ingr\_name)
* consist\_of(ord\_id, rest\_id, food\_name)
* staff\_serves(d\_id, staff\_id)

## 

## 2.1 user

**Relational Model:**

user (u\_id, e\_mail, password)

**Functional Dependencies:**

u\_id -> e\_mail password

**Candidate Keys:**

{(u\_id, e\_mail)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE user(

u\_id int AUTO\_INCREMENT NOT NULL,

email varchar(40) NOT NULL,

password varchar(20) NOT NULL,

PRIMARY KEY (u\_id)

UNIQUE (email)

);

## 

## 2.2 customer

**Relational Model:**

customer(u\_id, e\_mail, password, name, surname, {phone\_number})

**Functional Dependencies:**

u\_id -> e\_mail password name surname {phone\_number}

**Candidate Keys:**

{(u\_id, e\_mail)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE customer(

u\_id int,

email varchar(40),

password varchar(20),

name varchar(20) NOT NULL,

surname varchar(20) NOT NULL,

FOREIGN KEY (u\_id, email, password) REFERENCES user(u\_id, email, password)

);

CREATE TABLE phone\_number(

u\_id int,

phone\_num varchar(20) NOT NULL,

FOREIGN KEY (u\_id) REFERENCES customer(u\_id),

UNIQUE (phone\_num)

);

## 

## 2.3 restaurant

**Relational Model:**

restaurant(u\_id, email, password, name, address, {rest\_phone}, working\_hours, isOpen, rate, d\_id)

**Functional Dependencies:**

u\_id -> email name address password {rest\_phone} working\_hours rating isOpen d\_id rate

**Candidate Keys:**

{(u\_id email name address)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE restaurant(

u\_id int,

email varchar(40),

password varchar(20),

name varchar(20) NOT NULL,

address varchar(60) NOT NULL,

working\_hours varchar(20) NOT NULL,

isOpen bit NOT NULL,

rate numeric(3,2),

d\_id int,

PRIMARY KEY (u\_id),

FOREIGN KEY (u\_id, email, password) REFERENCES user(u\_id, email, password)

FOREIGN KEY (d\_id) REFERENCES district(d\_id),

UNIQUE (address),

UNIQUE (name)

);

CREATE TABLE rest\_phone(

u\_id int,

phone\_num varchar(20) NOT NULL,

FOREIGN KEY (u\_id) REFERENCES restaurant(u\_id),

UNIQUE (phone\_num)

);

## 

## 2.4 customer\_address

**Relational Model:**

customer\_address(u\_id, address\_title, name, surname, phone\_number, street\_name, street\_number, city, zipcode, adress\_desc, d\_id)

**Functional Dependencies:**

u\_id address\_title -> name surname phone\_number street\_name street\_number city zipcode address\_description d\_id

**Candidate Keys:**

{(u\_id, address\_title)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE customer\_adress(

u\_id int,

adress\_type varchar(20) NOT NULL,

name varchar(20),

surname varchar(20),

phone\_number varchar(20) NOT NULL,

street\_name varchar(20) NOT NULL,

street\_number numeric(8,0) NOT NULL,

city varchar(20) NOT NULL,

zipcode numeric(5,0) NOT NULL,

adress\_desc varchar(80),

d\_id int,

PRIMARY KEY (u\_id, address\_title),

FOREIGN KEY (u\_id, name, surname) REFERENCES customer(u\_id, name, surname),

FOREIGN KEY (d\_id) REFERENCES district(d\_id),

UNIQUE (phone\_num)

);

## 2.5 district

**Relational Model:**

district(d\_id, district\_name, area, city)

**Functional Dependencies:**

d\_id -> district\_name area city

**Candidate Keys:**

{(d\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE district(

d\_id int,

district\_name varchar(20) NOT NULL,

area varchar(20) NOT NULL,

city varchar(20) NOT NULL,

PRIMARY KEY (d\_id)

);

## 

## 2.6 order

**Relational Model:**

order(ord\_id, date, order\_notes, source\_address, dest\_address, pay\_type, comment, rate, cust\_id, rest\_id, staff\_id)

**Functional Dependencies:**

ord\_id -> date order\_notes source\_address dest\_address pay\_type cust\_id rest\_id staff\_id comment rate

**Candidate Keys:**

{(ord\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE order(

ord\_id int AUTO\_INCREMENT NOT NULL,

date date NOT NULL,

order\_notes varchar(200),

source\_address varchar(60) NOT NULL,

dest\_address varchar(60) NOT NULL,

pay\_type varchar(40) NOT NULL,

comment varchar(200),

rate int,

cust\_id int,

rest\_id int,

staff\_id int,

PRIMARY KEY (ord\_id),

FOREIGN KEY (cust\_id) REFERENCES customer\_address(u\_id),

FOREIGN KEY (rest\_id) REFERENCES restaurant(u\_id),

FOREIGN KEY (staff\_id) REFERENCES delivery\_staff(staff\_id),

FOREIGN KEY (source\_address) REFERENCES restaurant(address),

FOREIGN KEY (dest\_address) REFERENCES customer\_address );

## 2.7 delivery\_staff

**Relational Model:**

delivery\_staff(staff\_id, staff\_name)

**Functional Dependencies:**

staff\_id -> staff\_name

**Candidate Keys:**

{(staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE delivery\_staff (

staff\_id int AUTO\_INCREMENT,

staff\_name varchar(20) NOT NULL,

PRIMARY KEY (staff\_id)

);

## 

## 2.8 product

**Relational Model:**

product(rest\_id, food\_name, food\_type, price, portion,)

**Functional Dependencies:**

rest\_id food\_name -> food\_type price portion

**Candidate Keys:**

{(rest\_id, food\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE product(

rest\_id int,

food\_name varchar(32) NOT NULL,

food\_type varchar(32) NOT NULL,

price numeric(4,2),

portion numeric(3,1),

PRIMARY KEY (rest\_id, food\_name),

FOREIGN KEY (rest\_id) REFERENCES restaurant(u\_id)

);

## 

## 2.9 ingredient

**Relational Model:**

ingredient(ingr\_name, ingr\_calorie, quantity)

**Functional Dependencies:**

ingr\_name -> ingr\_calorie quantity

**Candidate Keys:**

{(ingr\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Ingredient(

ingr\_name varchar(32) NOT NULL,

ingr\_calorie varchar(32) NOT NULL,

quantity numeric(3,1),

PRIMARY KEY (ingr\_name),

);

## 

## 2.10 contains

**Relational Model:**

contains(rest\_id, food\_name, ingr\_name)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{(rest\_id, food\_name, ingr\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE contains(

rest\_id int,

food\_name varchar(32) NOT NULL,

ingr\_name varchar(32) NOT NULL,

PRIMARY KEY (rest\_id, food\_name, ingr\_name),

FOREIGN KEY (rest\_id, food\_name) REFERENCES product(rest\_id, food\_name),

FOREIGN KEY (ingr\_name) REFERENCES ingredient(ingr\_name)

);

## 

## 2.11 consist\_of

**Relational Model:**

consist\_of(ord\_id, rest\_id, food\_name)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{( ord\_id, rest\_id, food\_name)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE Comment (

ord\_id int,

rest\_id int,

food\_name varchar(32) NOT NULL,

PRIMARY KEY (ord\_id, rest\_id, food\_name),

FOREIGN KEY (ord\_id) REFERENCES order(ord\_id),

FOREIGN KEY (rest\_id, food\_name) REFERENCES product(rest\_id, food\_name)

);

## 

## 2.12 staff\_serves

**Relational Model:**

staff\_serves(d\_id, staff\_id)

**Functional Dependencies:**

NONE

**Candidate Keys:**

{(d\_id, staff\_id)}

**Normal Form:**

BCNF

**Table Definition:**

CREATE TABLE staff\_serves(

d\_id int,

staff\_id int,

FOREIGN KEY (d\_id) REFERENCES district(d\_id),

FOREIGN KEY (staff\_id) REFERENCES delivery\_staff(staff\_id));

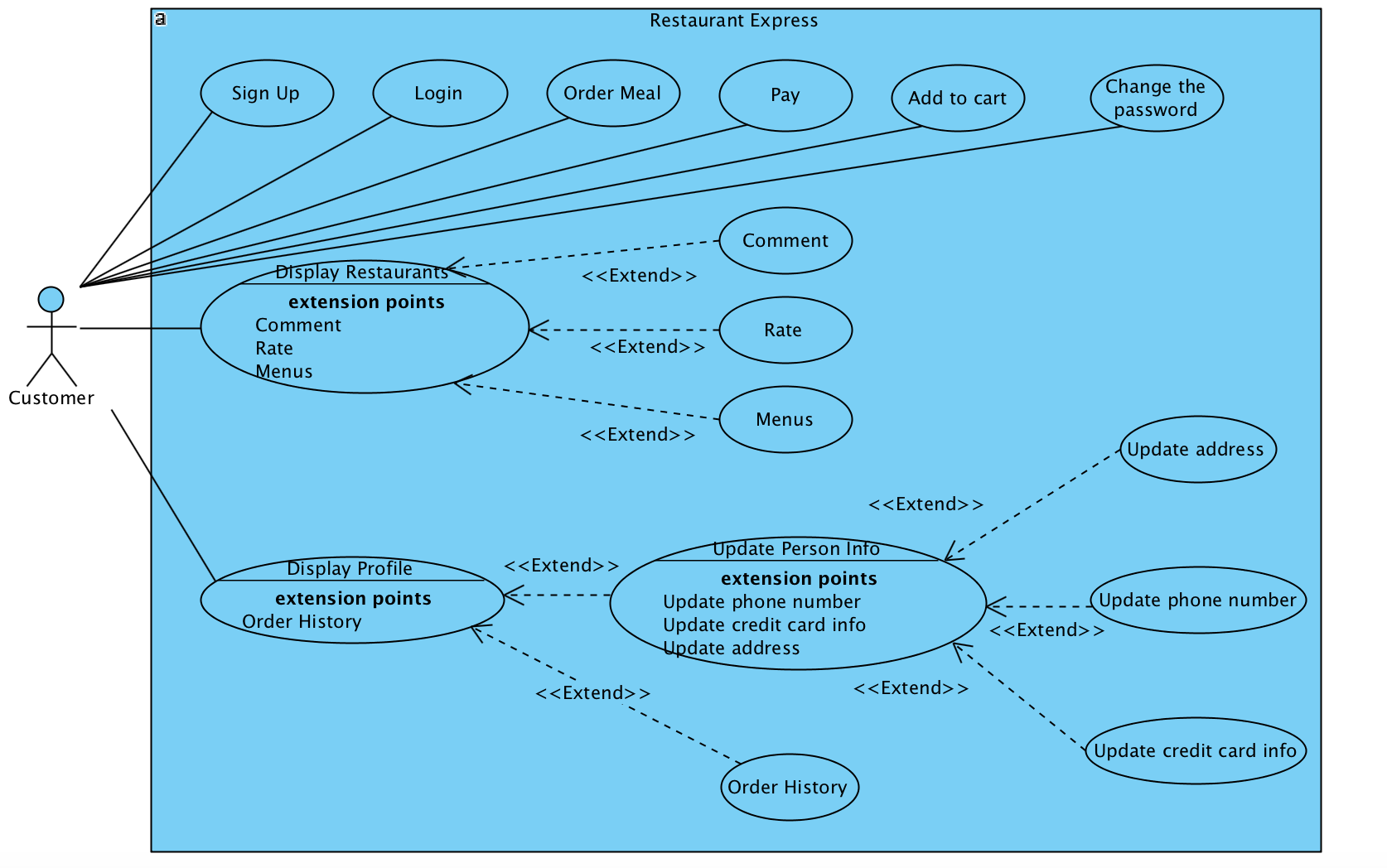
# 3 Functional Dependencies and Normalization of Tables

The Relation Schemas part of our design report contains all the functional dependencies and normal forms. Since the relations are all in Boyce-Codd Normal Form (BCNF), there is no need for any decomposition nor normalization.

# 4 Functional Components

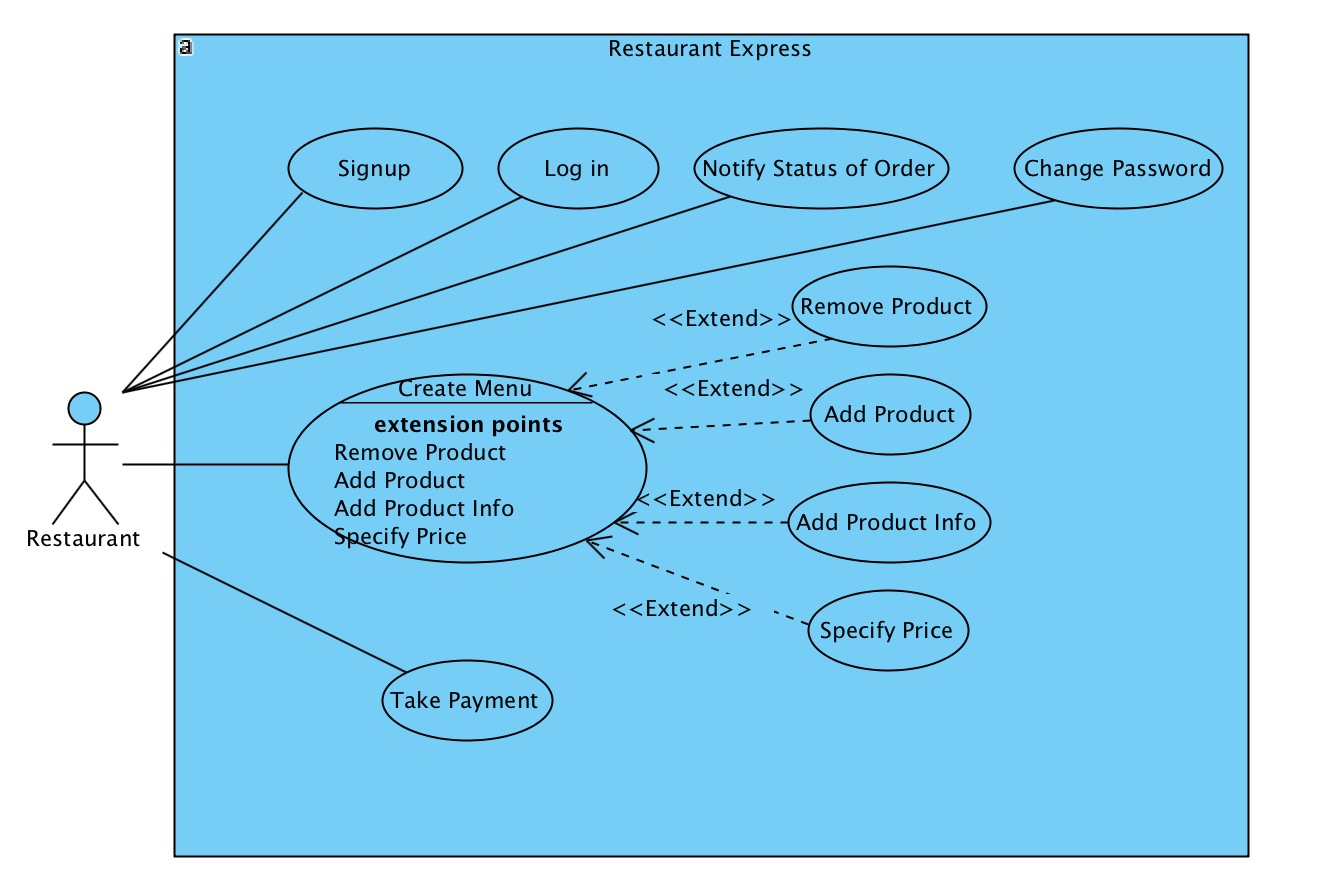
## 4.1 Use Cases / Scenarios

In Restaurant Express System, there are three types of user which are customers, restaurant and delivery staff. User types have some similar and different roles. Customers and Restaurants have to register and login to the Restaurant Express system.



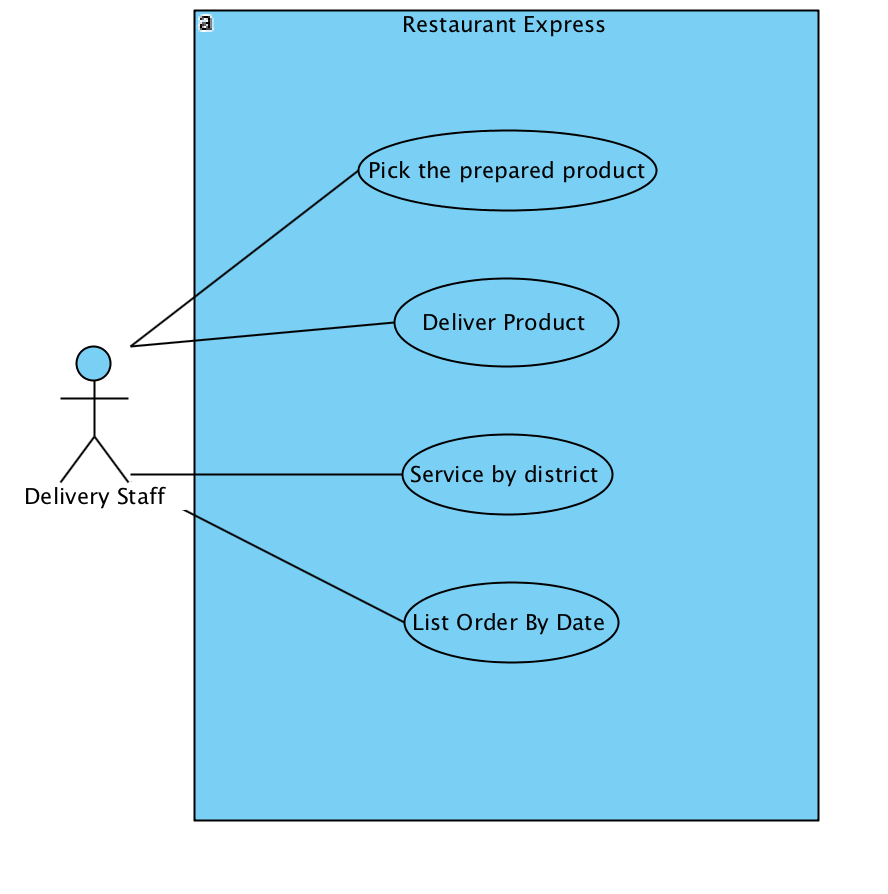
**Customer:**

* Customer can sign up to the system by his/her choosing username, entering email address, phone number and password.
* Customer can login his/her profile for ordering meal.
* Customer can change his/her password.
* Customer can order a meal from a restaurant which is chosen by the customer.
* Customer can pay his/her order’s price by his/her credit card information.
* Customer can pay his/her order’s price by cash to delivery staff.
* Customer can add his/her orders to cart for completing the ordering process.
* Customer can display the restaurant information which is consist of restaurant’s menu, comment and rate.
* Customer can display the restaurant’s top 5 products.
* Customer can make a comment to a restaurant after delivering his/her order from this restaurant.
* Customer can rate the restaurant after delivering his/her order from this restaurant.
* Customer can display the restaurant’s menu to get information about what menu is included, foods’ and drinks’ prices, ingredients and calories of foods and drinks.
* Customer can display the profile for changing personal information which includes address, phone number, credit card information and display the previous orders of his/her own.
* Customer can update his/her address ( add new address, edit address, delete address).
* Customer can update his/her phone number ( add new phone number, edit phone number, delete phone number if there are more than 1 telephone number).

**Restaurant:**

* Restaurant can sign up to the system by his/her writing their restaurant name, entering email address, entering location, phone number and password.
* Restaurant can login his/her profile for receiving the orders of the customers.
* Restaurant can change his/her password.
* Restaurant can send notifications status of the order to delivery staff.
* Restaurant can pay his/her order’s price by his/her credit card information.
* Restaurant can create a menu for the customers to give information of their own products to the customer which includes prices of the foods and drinks, ingredients and calories of foods and drinks.
* Restaurant can remove product(s) from their own menu.
* Restaurant can add product(s) from their own menu.
* Restaurant can add product information(s) from their own menu.
* Restaurant can specify the prices of the products from their own menu.
* Restaurant can take payment of the customers by credit card.

**Delivery Staff:**



* Delivery Staff can pick the prepared product of the restaurant which is ordered by the customer.
* Delivery Staff can deliver the product to the customer.
* Delivery Staff can arrange the services of the products by district.
* Delivery Staff can list products by date.

## 4.2 Algorithms

## 4.2.1 Restaurant-related Algorithms

Though restaurants are also essentially users of the system just like the customers, their usage of the system is fundamentally different than the customers in the way that restaurants are sellers.

Every restaurant has a dedicated page with product lists, address, working hours, is open or not, phone numbers and credit card info stored in our data structure. Since serving citywide wouldn’t be profitable for a restaurant (due to cost of delivery and time restraints) each restaurant also specifies which districts they will serve.

Restaurants need to provide all this info when signing up to the system, and can change it anytime. For example a restaurant can change their is open status to closed, after which our service will no longer take orders for that restaurant.

Each restaurant will also have a rating and have a list of most ordered foods. However, since these attributes change often due to new orders they will not be explicitly stored in our database but be generated from the history of past orders and ratings at intervals.

## 4.2.3 Order and Delivery-Related Algorithms

Orders are made by the customer through a restaurant page for food products. Each order must come specifically from one restaurant, if user wants products from multiple restaurants then new orders must be made for each. Every order is received is tagged with an ID number generated by the data structure. Each order also has a time of order, pay type (being cash, online payment or credit card), notes for the restaurant, address of the restaurant and the address of the destination.

Delivery staff acts upon the pay type attribute for interacting with the customer, for example bringing a POS device if customer requests credit card payment or not requesting any form of payment from the customer if pay type is online. Estimated delivery time is not a stored attribute in our database but rather a generated one utilizing Google Traffic API to guess the time using time, source and destination addresses as input.

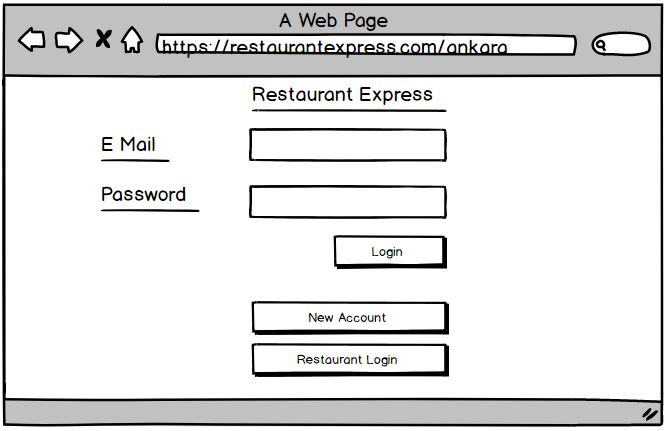
## 4.2.4 Customer-Related Algorithms

Each customer is a user of the system with name surname and phone numbers additional to the email and password as a user. Each customer has a unique phone number and is internally identified by the generated unique D number all of which is stored in our database. Users also have address infos including adress title, phone number,street name, street number, district, zip code and description. Users can define multiple address types all of which is stored in our data structure. which can be more than one. Once address is defined users can select one of their districts and see the restaurants that serve to their address and be able to order. To make it easier to see restaurants first time users will have their district detected by the ip address and thus be able to see the restaurants in their area right away and be able to add food to their carts but won’t be able to order unless they sign in or login. Users who are not active for a specific timeframe will be logged out and have to log back in next time they try to use the service.

Users will also be able to see their past orders and occasionally users will get joker deals based on their past orders and their tendencies.

# 5 User Interface Design and Corresponding SQL Statements

## 5.1 Login



**Inputs:** @email, @password.

**Process:** If customer has an account, he/she can enter the system by entering the email and password. Otherwise, he/she can go to register page by clicking the “Create Account” button. Restaurants can also go to login page by clicking the “Restaurant Login” link.

**SQL Statements:**

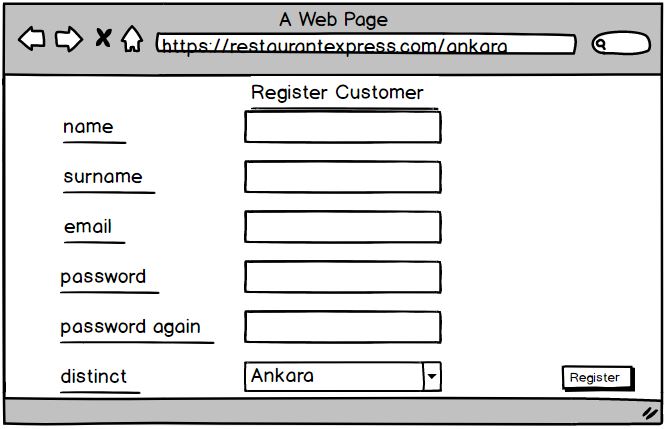
**Get User:**

SELECT \*

FROM customer

WHERE email = @email AND password = @password;

## 5.2 Customer Register



**Inputs:** @name, @surname, @email, @password1, @password2, @district

**Process:** Customer can register the system by entering the his/her name, surname, email, password and district.

@password1 is the password in the first field

@password2 is the password in the second field

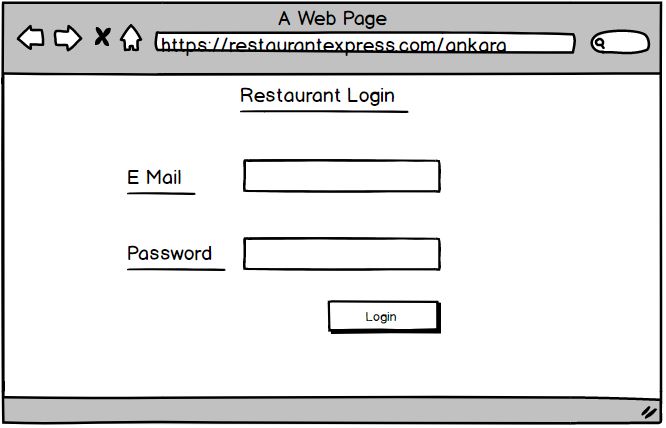
**SQL Statements:**

**Add User:**

INSERT INTO customer(email, password)

VALUES (@email, @password1);

## 5.3 Restaurant Login



**Inputs:** @email, @password

**Process:** Restaurant can enter the system by entering the its email, password.

**SQL Statements:**

**Get Registered Restaurant:**

SELECT \*

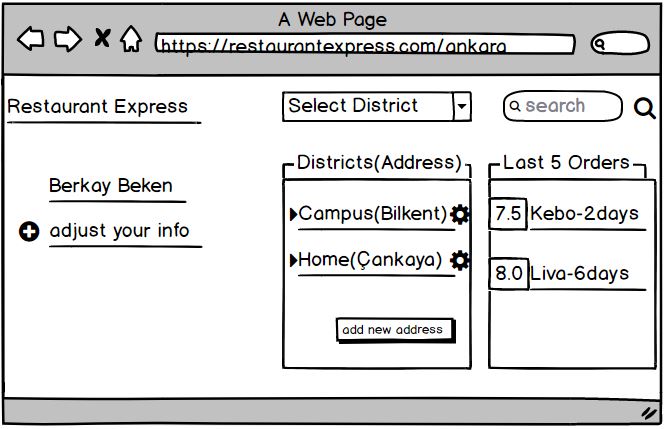
FROM restaurant

WHERE email = @email AND password = @password;

## 

## 

## 5.4 Home Page



**Inputs:** @currentUserID

**Process:** There will be one column to show the addresses of the customer and one other column to show the recent orders. Also, there will be two links. One of them will be to view and change the personal info and the other will be to add an address. Customer can adjust their address informations by clicking the setting icon nearby the address names. Finally, there will be a search box on top which will make a search from products of all restaurants in the selected district.

**SQL Statements:**

**Get Addresses:**

SELECT C.address\_title, D.district\_name

FROM customer\_address C JOIN district D ON C.d\_id= D.d\_id

WHERE C.u\_id=@currentUserID ;

**Get Recent Orders:**

SELECT R,rate, R.name

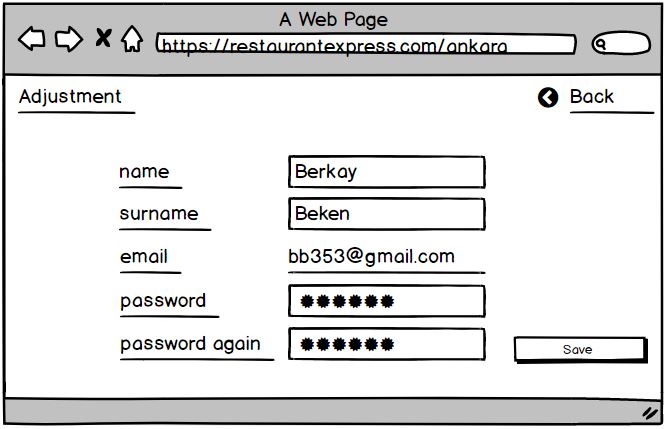
FROM order O JOIN restaurant R ON O.rest\_id= R.u\_id

WHERE O.cust\_id=@currentUserID

ORDER BY O.ord\_id DESC

LIMIT 5;

## 5.5 Adjustment



**Inputs:** @currentUserID, @name, @surname, @email, @password, @password2

**Process:** Customers can see their informations and they can adjust them. They can change their names, surnames and passwords by entering the new ones in the corresponding text fields. At the end, they can save their new informations by clicking “save” button or they can leave without any changes by clicking the “back” button.

@password is the password in the first password field

@password2 is the password in the second password field

**SQL Statements:**

**Get Current Values:**

SELECT name, surname, email, password

FROM customer

WHERE u\_id=@currentUserID ;

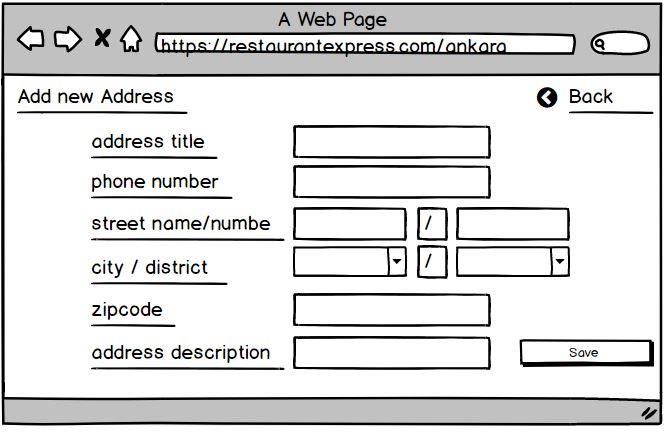
**Update the Values:**

UPDATE customer

SET name = @name, surname= @surname, email= @email, password= @password

WHERE u\_id=@currentUserID ;

## 5.6 Add new Address



**Inputs:** @currentUserID, @currentName, @currentSurname, @address\_title, @phone\_number, @street\_name, @street\_number, @city, @zipcode, @address\_desc, @district\_id

**Process:** Customers can add new address by entering the address title, phone number, street-name, street-number, city, district, zip-code and address description. They can save the new address by clicking the “done” button or they can leave without any changes by clicking the “back” button.

**SQL Statements:**

**Get districts of the city (Fill second field after the first one):**

SELECT district\_name

FROM district

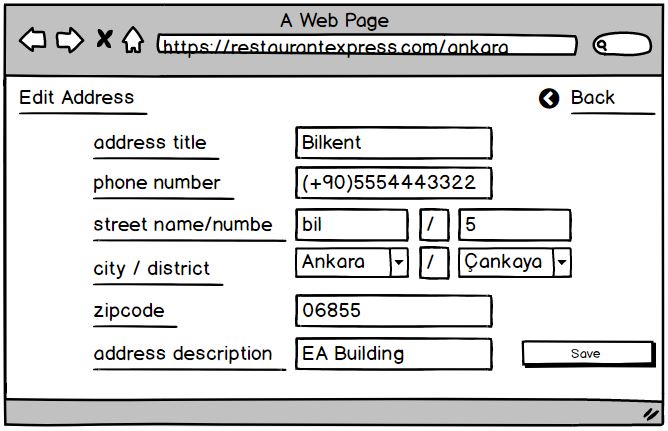
WHERE city = @city

**Add Address:**

INSERT INTO customer\_address(u\_id, address\_title, name, surname, phone\_number, street\_name, street\_number, city, zipcode, adress\_desc, d\_id)

VALUES (@currentUserID, @address\_title, @currentName, @currentSurname, @phone\_number, @street\_name, @street\_number, @city, @zipcode, @adress\_desc, @district\_id);

## 5.7 Edit Address Info



**Inputs:** @currentUserID, @address\_title, @phone\_number, @street\_name, @street\_number, @city, @zipcode, @address\_desc, @district\_id

**Process:** Customers can see their address informations and they can adjust them. They can change their address title, phone number, street-name, street-number, city, district, zip-code and address description by entering the new ones in the corresponding text fields. At the end, they can save their new informations by clicking “save” button or they can leave without any changes by clicking the “back” button.

**SQL Statements:**

**Get Current Values:**

SELECT address\_title, phone\_number, street\_name, street\_number, d\_id, city, zipcode, address\_desc

FROM customer\_address

WHERE u\_id=@currentUserID ;

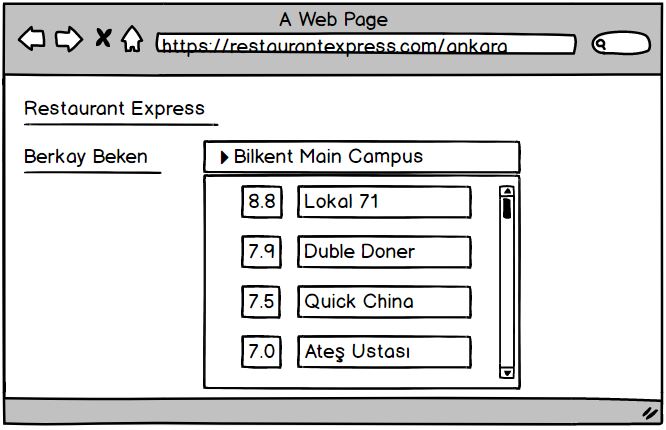
**Update the Values:**

UPDATE customer\_address

SET address\_title= @address\_title, phone\_number= @phone\_number, street\_name= @street\_name, street\_number= @street\_number, d\_id= @district\_id, city= @city, zipcode= @zipcode, address\_desc= @address\_desc

WHERE u\_id=@currentUserID ;

## 5.8 List Restaurants

****

**Inputs:** @currentDistrictID

**Process:** Customers can list the restaurant in the selected district. They can see the restaurant’s names and rates.

**SQL Statements:**

**Get All Restaurants:**

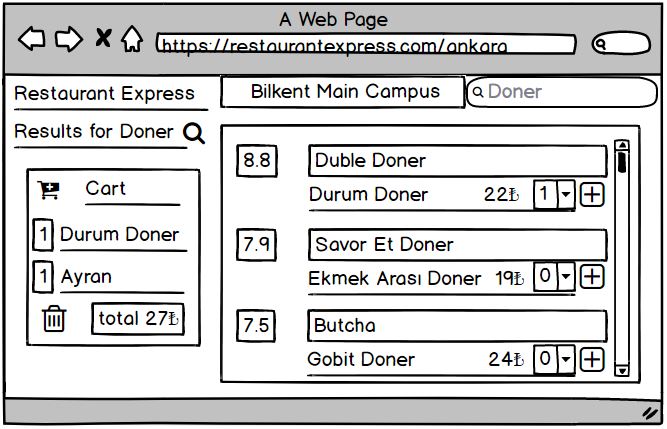
SELECT rate, name

FROM restaurant

WHERE d\_id=@currentDistrictID

ORDER BY rate DESC;

## 5.9 Search

****

**Inputs:** @currentUserID, @currentDistrictID, @searchedText

**Process:** Customers can search the meals with keywords and the district. The result of the search is that the list of the restaurants which have the desired meal on their menus. Under the each restaurant title, the related menus are listed. Customer can see the price of the menus near the menus’ title and can choose the portion number. Customer can add any different menus of only one restaurant to cart by clicking the plus icon and the total price appears accordingly.

**SQL Statements:**

**Get All Restaurants:**

SELECT R.rate, R.name, P.food\_name, P.price

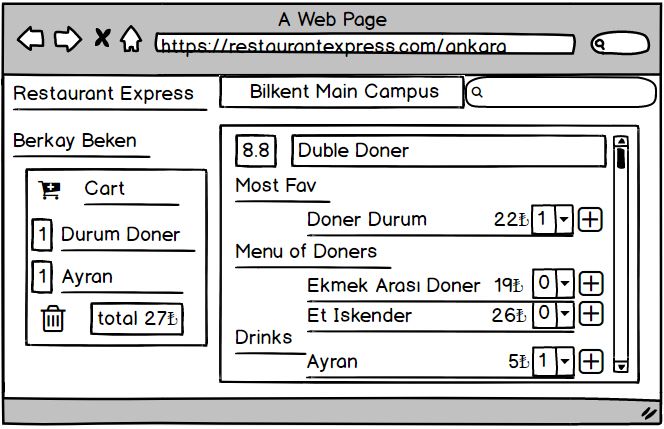
FROM restaurant R JOIN product P

ON R.u\_id=P.rest\_id

WHERE d\_id=@currentDistrictID AND (P.food\_name LIKE “%@searchedText%” OR R.name LIKE “%@searchedText%”)

ORDER BY rate DESC;

## 5.10 List Menus of the Restaurant

****

**Inputs:** @currentRestID,

**Process:** Customer can list the all menus of the selected restaurant. Customer can see the price of the menus near the menus’ title and can choose the portion number. The most favorite foods or best seller meals are appeared at the top of the menus (to represent it on the mock-up just one line is allocated, but the number of most favorite foods can be at most five in our project). Customer can add any different menus of the restaurant to cart by clicking the plus icon and the total price appears accordingly.

**SQL Statements:**

**Get Menus of a Restaurant:**

SELECT food\_name, price

FROM product

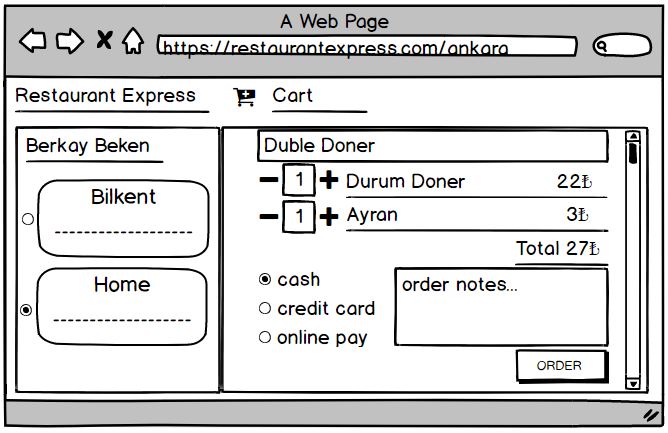
WHERE rest\_id=@currentRestID

GROUP BY food\_type ;

**Get Most Favorite Menus of a Restaurant:**

We will explain the SQL of the getting most favorite menus of the restaurants in section 6.2.1.

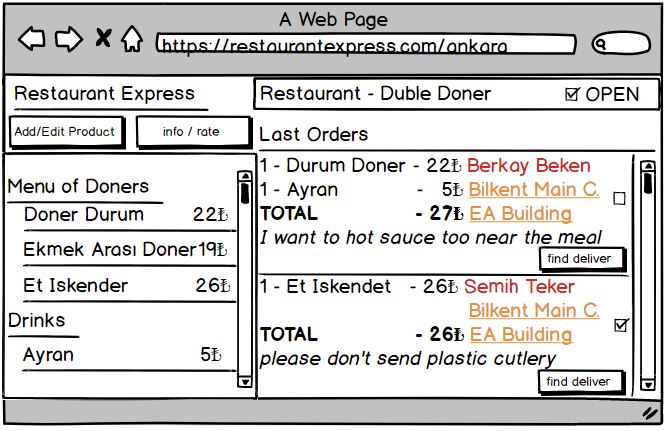
## 5.11 Cart



**Process:** Customer can list the all selected menus to buy. Customer can see the price of the menus near the menus’ title and can increase or decrease the portion number by clicking the plus and minus icon, the price is updating by accordingly. Customer can discard the menus’ item by decreasing the portion as zero. Customer can also see their list of address and they can choose one of these address if and only if address in the restaurant’s district. Customers can choose one of the payment type and if they want to add a note about order, they can type the note box. At the end, by clicking “order” button they confirm the meal ordering.

Note : We decided to make this part in the application without using a cart entity.

## 5.12 Restaurant Page



**Inputs:** @currentUserID

**Process:** Restaurant can display the page by pressing Add/Edit Product button. Thus, restaurant will have a chance to change the number of remaining products. Restaurant can press info/rate button for displaying the customers’ ratings and comments about their products. Thus, if their rate and comments are bad, they can better prepare their foods according to given rates and comments. If their rate and comments are good, more people will choose their restaurant. Thus, their profit will rise. Restaurant can edit their status by OPEN-CLOSE writing. If their status is open, customers can order but if it close any customer can not order from this restaurant. Restaurant can mark the box which takes part near the restaurant status, to show customers their order is preparing. Restaurant can find a delivery staff with pressing find deliver button after finishing the preparation of the order. Restaurant can display the notes of the customer about the orders. Restaurant can display the orders of the customers with their prices.

**SQL Statements:**

**Restaurant view of orders (Also the view in 6.1.2):**

CREATE VIEW rest\_order AS

SELECT O.date, O.order\_notes, O.dest\_address, O.staff\_id, O.pay\_type, C.food\_name, P.price, P.portion

FROM order O NATURAL JOIN consist\_of C NATURAL JOIN product P, restaurant R

WHERE R.u\_id=C.rest\_id AND R.name=@currentRestName

**Show Orders with respect to their dates:**

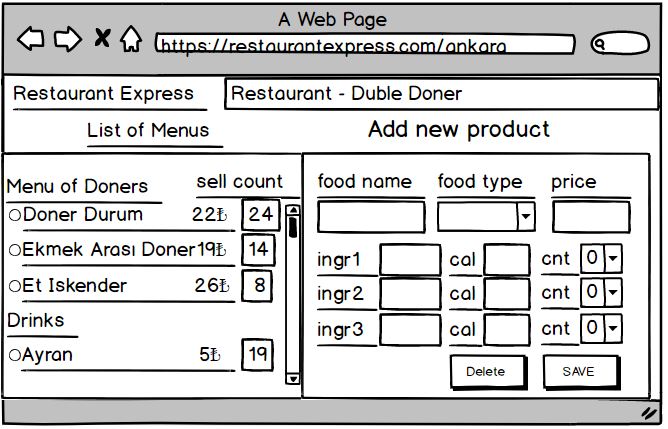
SELECT \*

FROM rest\_order

ORDER BY Order.ord\_id DESC;

## 

## 5.13 Add new Product / Edit Products

****

**Inputs: @**currentRestID, @food\_name, @food\_type, @price, @ingr1, @cal1, @cnt1, @ingr2, @cal2, @cnt2, @ingr3, @cal3, @cnt3

**Process:** Restaurant can display the remaining products at the menu with numbers. Restaurant can add the product number as they wish or can delete the existing menu by clicking delete button. Restaurant can edit the products’ prices as they wish. This change will not affect the users cart price if they already order their product before editing the price. Restaurant can add or remove something from their ingredients list of the food. Restaurant can add calorie information of all ingredients to give brief information to customer. Restaurant can enter the number of ingredients to a product with combo box. Restaurant can sort the products by type on the menu.

**SQL Statements:**

**Get Sell Count of Menus (Also in 6.2.1):**

SELECT food\_name, COUNT(\*)

FROM consists of

GROUP BY food\_name

**Get Menu of restaurant (categorized):**

SELECT food\_name, price

FROM product

WHERE rest\_id=@currentRestID

GROUP BY food\_type ;

**Add Product:**

INSERT INTO ingredient(ingr\_name, ingr\_calorie, quantity)

VALUES (@ingr1, @cal1, @cnt1);

VALUES (@ingr2, @cal2, @cnt2);

VALUES (@ingr3, @cal3, @cnt3);

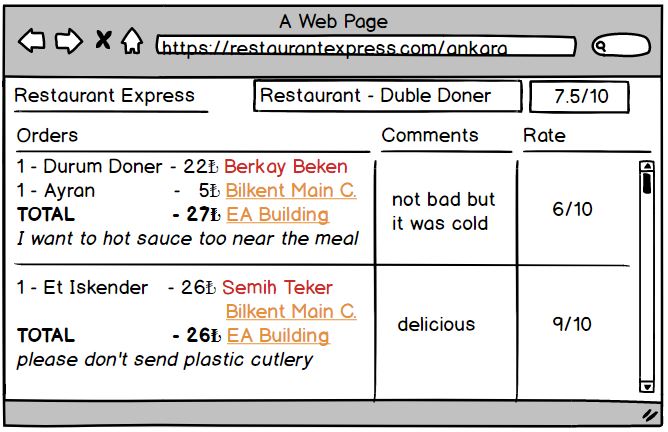
INSERT INTO product(rest\_id, food\_name, food\_type, price)

VALUES (@currentRestID, @food\_name, @food\_type,@price );

INSERT INTO contains(rest\_id, food\_name, ingr\_name)

VALUES (@currentRestID, @food\_name, @ingr\_name);

## 5.14 Restaurant Info and Rate

****

**Process:** Restaurant can display the previous orders from customer with see the portions of products, see the cost of the order, see the names of the customers, see the notes of the customers, see the locations of delivery, see the comments’ and rates’ of the customers after ordering food from their own restaurants. Restaurant also can see the overall rate of their own restaurant.

**SQL Statements:**

**Restaurant view of orders (Extended version of the view in 6.1.2):**

CREATE VIEW rest\_order AS

SELECT O.date, O.order\_notes, O.dest\_address, O.staff\_id, O.pay\_type, C.food\_name, P.price, P.portion, O.comment, O.rate

FROM order O NATURAL JOIN consist\_of C NATURAL JOIN product P, restaurant R

WHERE R.u\_id=C.rest\_id AND R.name=@currentRestName;

# 6 Advanced Database Components

## 6.1 Views

### 6.1.1 Customer view of Order

* After the order, a customer cares only about important things to reorder the same food and the date of that order. Therefore, we will only need these 5 things to show in the recent orders page.

1. The date of the order
2. The name of the food
3. The name of the restaurant
4. The portion of that food (If it was not enough he/she can increase the portion or if it was too much he/she can decrease the portion.)
5. The price

SQL query of the view is:

CREATE VIEW cust\_order AS

SELECT O.date, C.food\_name, R.name, P.price, P.portion

FROM order O NATURAL JOIN consist\_of C NATURAL JOIN product P, restaurant R

WHERE R.u\_id=C.rest\_id

### 6.1.2 Restaurant view of Order

* After the order, a restaurant cares about how good was the order and how satisfied the customer is. Therefore, unlike the customer, the boss of the restaurant will care about the staff member or the notes of the order. To name what we exactly need for this view:

1. The date of the order
2. The notes about the order
3. The destination address
4. The delivery staff
5. Payment type (If customer comes up with a problem about payment)
6. The name of the food
7. The portion of that food
8. The price

SQL query of the view is:

CREATE VIEW rest\_order AS

SELECT O.date, O.order\_notes, O.dest\_address, O.staff\_id, O.pay\_type, C.food\_name, P.price, P.portion

FROM order O NATURAL JOIN consist\_of C NATURAL JOIN product P, restaurant R

WHERE R.u\_id=C.rest\_id AND R.name=@currentRestName

## 6.2 Reports

### 6.2.1 Number of orders of each food

* Restaurants are interested in which food is the most preferred one. So, they should be able to see that information for every food. Besides, counting the numbers is also useful for viewing the most preferred ones to the customer.

SQL query of this report is:

SELECT food\_name, COUNT(\*)

FROM consists of

GROUP BY food\_name

### 6.2.2 Number of orders of each customer

* Restaurants are interested in which customer is most dedicated for that restaurant and they can try to surprise him/her with extra dessert etc. They will be able to see the number of orders of each customer.

SQL query of this report is:

CREATE VIEW order\_count\_of\_cust\_report AS

SELECT O.cust\_id, C.name, C.surname, COUNT(\*)

FROM order O JOIN customer C

ON O.cust\_id=C.u\_id

GROUP BY O.cust\_id

### 6.2.3 Amount of money from one customer

* It is written “Restaurants are interested in which customer is most dedicated for that restaurant and they can try to surprise him/her with extra dessert etc.” in the previous part. This part is about the total money coming from one customer. Restaurant can use this information for bigger deals like 20% off for one week as an award to that specific customer. This part is not of our interest, we will only provide the information.

SQL query of this report is:

SELECT O.cust\_id, C.name, C.surname,SUM(P.price) as amount

FROM order O NATURAL JOIN consist\_of C NATURAL JOIN product P, customer C

WHERE O.cust\_id=C.u\_id

GROUP BY O.cust\_id

## 6.3 Triggers

* When a comment on the order is made by a customer, the rating of the corresponding restaurant will be updated accordingly.
* When an order is made the corresponding reports will be updated accordingly (the number of times a specific food is ordered etc.)
* When an order is made the number of remaining product will be updated accordingly.
* When the restaurant status change to close, customer will not access the order page of closed restaurant.
* When the restaurant add a product to restaurant menu or remove a product from restaurant menu, menu of this restaurant will be updated immediately.

## 6.4 Constraints

* Only certain parts of the service can be used (such as seeing restaurants serving to the IP address location and add products to the cart but not be able to complete the order) if user is not logged in.
* Users will get logged out of the system due to inactivity.
* Users can not order unless they have a defined address and are logged in.
* Any online payment that isn’t completed (due to insufficient funds, internal problems…) will get the order terminated.
* Any order from multiple restaurants need to be separated into multiple orders as, same delivery agent can not deliver an order from 2 different restaurants at the same time.
* Orders cancelled by the restaurant or the system need to be notified to the user either by phone, or email. (There is needed to pass the minimum price)
* User dissatisfaction must be able to be addressed to the service by a communication apparatus.
* Customers and restaurants who do not follow the community guidelines must be able to be flagged and banned from the service if necessary.
* Joker sales are only valid between start and finish dates.

## 6.5 Stored Procedures

We plan to use stored procedures while ordering. Ordered products will form a tuple of products where the price and the quantity of the products will be calculated by summing all the products in the order to form a total price and quantity. We will use this total price also for the quota for the min price of a order. For example if the restaurant is in Kızılay and has to go to Gölbaşı for an order of tl total price than that wouldn’t be profitable.

Since we don’t store users’ past orders or restaurants favorite ordered foods explicitly as an attribute we don’t need a procedure to reverse those in case an order is cancelled. We generate the most ordered food and the order history attributes by using a function that goes one by one through the consecutive users’.

# 7 Implementation Plan

For our system functionalities and user interface in our hypertext dictionary system, we are planning to use HTML, CSS, JavaScript, Asp.net. In order to manage the flow of data in our project, we are planning to use MySQL Server.

# 8 Website

https://github.com/git-bahadir/Restaurant-Express