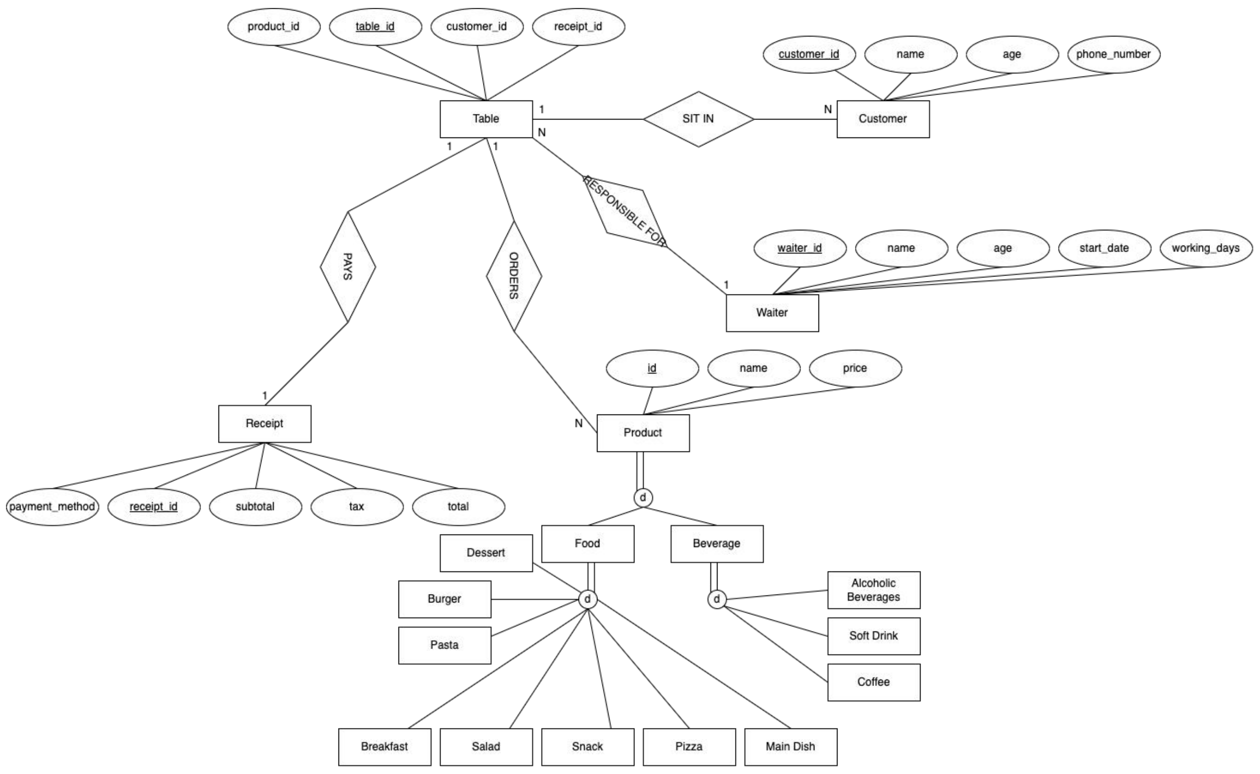
Restaurant Management System

Yusuf Erdemirler, Burak Yıldırım, Alkan Akısu, Mete Uz, Mert Köksal

Project Description:

We have developed an interface to keep the data ordered by the customers used in the restaurants, the waiters can save the orders for each table and create the total account fees by making selections from the data they access using this interface.

Entity-Relationship Diagram:



Relational Database Design:

```
CREATE TABLE CUSTOMER (customer_id VARCHAR(50) NOT NULL,
                       name VARCHAR(50),
                       age INTEGER,
                       phone_number VARCHAR(50),
                       PRIMARY KEY(customer_id))

CREATE TABLE RECEIPT (receipt_id VARCHAR(50) NOT NULL,
                              sub_total INTEGER,

                              tax INTEGER,

                              total INTEGER,

                              payment_method VARCHAR(50),

                              PRIMARY KEY(receipt_id))

CREATE TABLE WAITER (waiter_id VARCHAR(50) NOT NULL,

                              name VARCHAR(50),

                              age INTEGER,

                              start_date DATETIME,

                              working_days INTEGER,

                              PRIMARY KEY(waiter_id))

CREATE TABLE TABLES (table_id VARCHAR(50) NOT NULL,

                             customer_id VARCHAR(50) NOT NULL,

                             receipt_id VARCHAR(50) NOT NULL,

                             product_id VARCHAR(50) NOT NULL,

                             PRIMARY KEY(table_id),

                             FOREIGN KEY(customer_id) REFERENCES CUSTOMER(customer_id),

                             FOREIGN KEY(receipt_id) REFERENCES RECEIPT(receipt_id),

                             FOREIGN KEY(product_id) REFERENCES PRODUCT(id))

CREATE TABLE i (id VARCHAR(50) NOT NULL,

                             name VARCHAR(50),

                             price FLOAT NOT NULL,

                             PRIMARY KEY(id))
```

i = PRODUCT, FOOD, BEVERAGES, BREAKFAST, PIZZA, MAIN_DISH, SNACKS, PASTA, DESERT, SALADS, BURGER, ALCHOLIC_BEVERAGES, SOFT_DRINKS, COFFEE.

Data Sources:

We created our database by using the menus of American and Turkish restaurant chains, we used our own fictitious data for some entities such as Customer and Waiter entity.

Complex SQL Queries:

Query 1:

```
SELECT avg(total), customer.customer_id

From customer, product

Natural Join tables

Natural Join receipt

Where product_id = id AND customer.customer_id = tables.customer_id and id in (

        Select id

  From alcholic_beverages

  )

Group By customer.customer_id

Having count(*) > 0;
```

Returns : Average spent on alcohol per order by each customer.

Reasoning: It is useful for a restaurant to know who spends highly on alcohol since they are usually the top paying customers.

Query 2:

```
select price, name , Count(*) as timesOrdered

from tables, product

where id in (

        Select id

  from product
```

```
    where price >= (

                    Select max(price)

        From product))

            and product_id = id;
```

Returns : The most expensive item in the menu and the number of times it was ordered.

Reasoning: A restaurant might want to adjust the price of or simply get rid of a very expensive item by gauging its popularity.

Query 3:

```
Select sum(total) as totalPaid, sum((((tax / 100 ) + 1) * total) - total) as totalTaxPaid,
payment_method

From receipt

group by payment_method;
```

Returns : How much tax paid by each payment method and the original cost.

Reasoning: A restaurant might want to create incentive to pay with another method if it is taxed less.

Query 4:

```
Select name, sum(total) as totalPaid

From customer

Natural Join tables

Natural Join receipt

Where customer_id in (

            Select customer_id

    from tables

    Natural join receipt

    group by customer_id
```

    having sum(total) > 100)

group by customer_id

Returns : Names of customers who paid more than $100.

Reasoning: It is useful to know the top spenders and people who visit frequently.

Query 5:

Select sub_total , (sub_total * (1 +(count(*)/ 20))) as projectedTotal, product.name, price, price * (1 + (count(*)/ 20))as projectedPrice

From tables, product, receipt

where id = product_id and tables.receipt_id = receipt.receipt_id

group by id

having count(*) > 0;

Returns : Total paid per item and its menu price. Projected prices and totals if the restaurant wanted to raise prices by %5 per item ordered.

Reasoning: The restaurant can calculate potential profit if they wanted raise prices according to the item's popularity.

Screenshots:

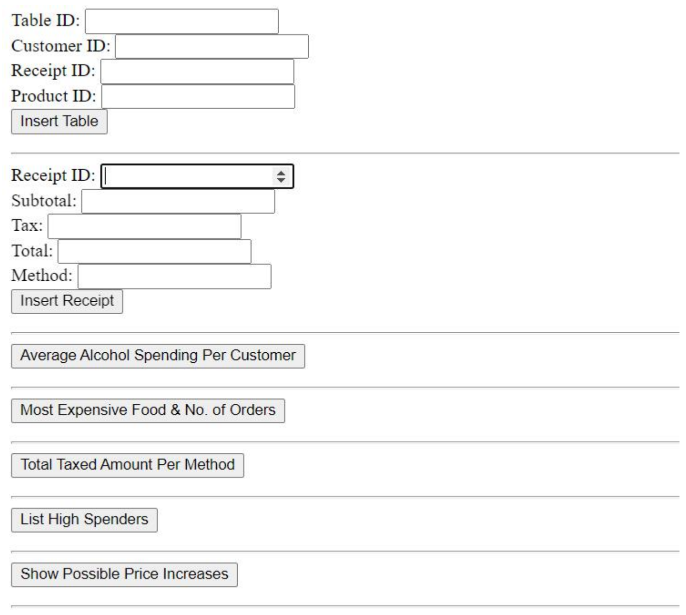


Table ID:
Customer ID:
Receipt ID:
Product ID:
Insert Table
Receipt ID:
Subtotal:
Tax:
Total:
Method:
Insert Receipt
Average Alcohol Spending Per Customer
Most Expensive Food & No. of Orders
Total Taxed Amount Per Method
List High Spenders
Show Possible Price Increases

Total Paid	Total Tax	Method
33	3.3000	card
2	0.0200	e
225	28.2500	cash
340	51.0000	sodexo
15	7.5000	coupon

Total Paid (Before Tax)	Total Paid After Increase	Product Name	Menu Price	Menu Price After Increase
10	10.5000	"Margarita"	11.5	12.075000000000001
1	1.0500	"Porn Star Martini"	10.1	10.605000400543213
10	10.5000	"Hand-Tossed Medium"	11.99	12.589499759674073
10	11.5000	"24oz Porterhouse"	34.95	40.19250087738037
150	172.5000	"Cosmopolitan"	12.45	14.317499780654906

Name	Total Paid
"Burak Yildirim"	121
"İsmail Hakkı Yesil"	128
"Mete Uz"	351