Department of Computer Engineering

Bilkent University

CS353 Database Systems

Spring 2018

Design Report

Tour Reservation Management System

Group 1

Ozan Kerem Devamlı 21301886

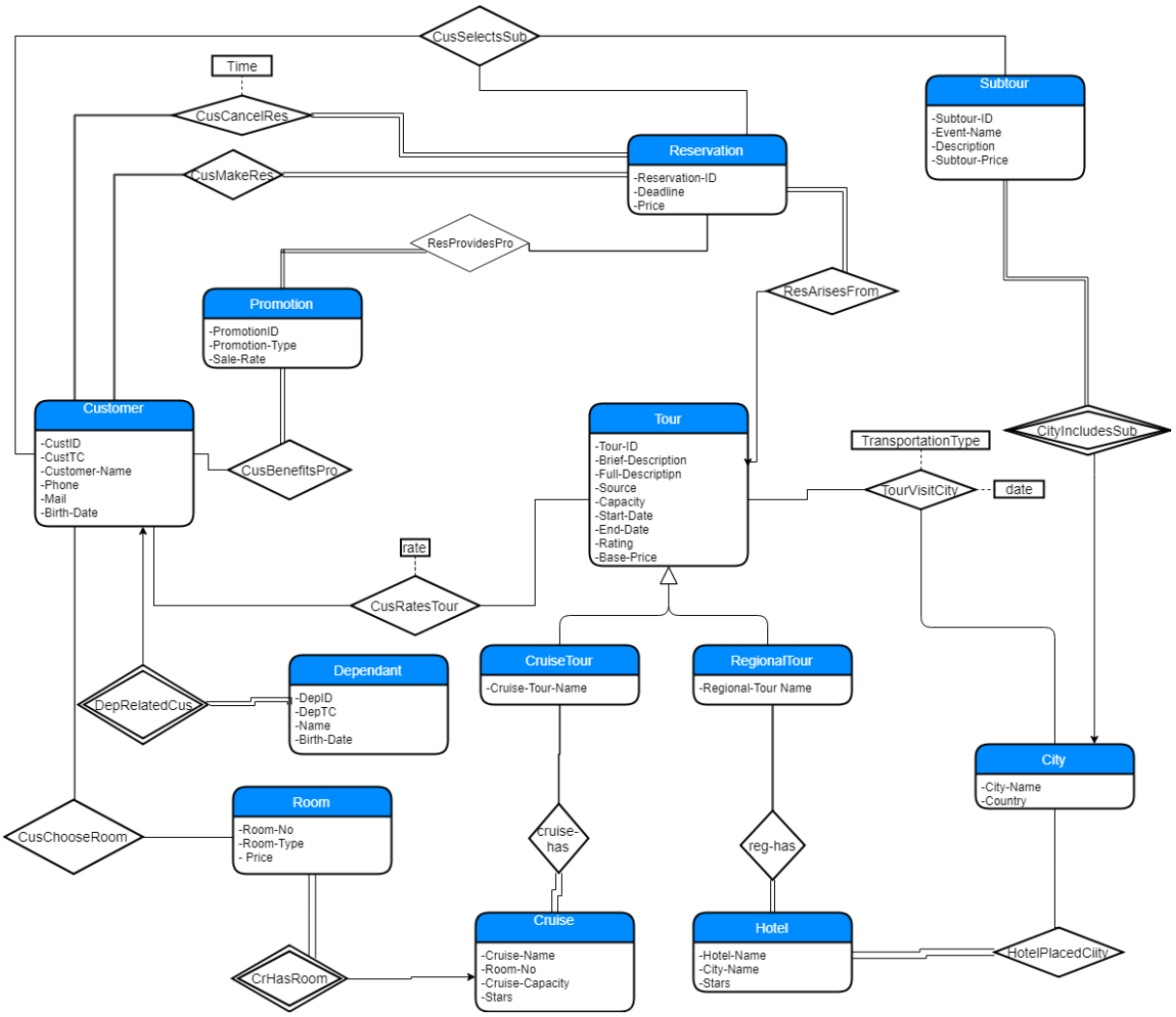
Alper Kılıçaslan 21502103

Abdullah Seçkin Özdil 21201929

Cansu Yıldırım 21502891

Table of Contents

1. Revised E/R Model



2. Relational Schema

2.1 Customer

Relational Model:

Customer(<u>custID</u>, custTC, customer-name, phone, mail, birth-date)

Functional Dependencies:

custID -> custTC, customer-name, phone, mail, birth-date

Candidate Key: {(custID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Customer(
        custID              int not null auto-increment
        custTC                      int not null
        customer-name       varchar(50) not null,
        phone               long not null unique,
        mail                varchar(50) not null unique,
        birth-date          date,
        PRIMARY KEY(custID)
)
```

2.2 Reservation

Relational Model:

Reservation (<u>reservationID</u>, deadline, price, tourID)

Functional Dependencies:

reservationID -> deadline, price, tourID

Candidate Key: {(reservationID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Reservation(
        reservationID       int not null,
        price               numeric(8,2),
        deadline            date,
        tourID              int not null
        PRIMARY KEY(reservationID)
        FOREIGN KEY tourID REFERENCES Tour(tourID)
)
```

2.3 Tour

Relational Model:

Tour(<u>tourID</u>, brief-description, full-description, source, capacity, startDate, endDate, rating, base-price)

Functional Dependencies:

tourID -> source, brief-description, full-description, capacity, startDate, endDate, rating, base-price

Candidate Key: {(tourID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Tour(
        tourID                  int not null auto-increment,
        brief-description       varchar(300) not null,
        full-description        varchar(1200),
        source                          varchar(20) not null,
        capacity                long not null,
        startDate               date,
        endDate                 date,
        rating                  numeric(11,9),
        base-price              numeric(8,2),
        PRIMARY KEY(tourID)
)
```

2.4 Regional Tour

Relational Model:

RegionalTour(<u>tourID</u>, regional-tour-name)

Functional Dependencies:

tourID -> regional-tour-name

Candidate Key: {(tourID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE RegionalTour(
    tourID                  int not null,
    regional-tour-name      varchar(50) not null,
    PRIMARY KEY(tourID),
    FOREIGN KEY tourID REFERENCES Tour(tourID)
)
```

2.5 Hotel

Relational Model:

Hotel(<u>hotelName</u>, cityName, stars)

Functional Dependencies:

hotelName -> cityName, stars

Candidate Key: {(hotelName)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Hotel(
    hotelName       varchar(20) not null,
    cityName        varchar(20) not null,
    stars           int(1) not null,
    PRIMARY KEY(hotelName)
)
```

2.6 Reg-Has

Relational Model:

Reg-Has(<u>tourID, hotelName</u>)

Functional Dependencies:

No functional Dependencies

Candidate Key: {(tourID, hotelName)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Reg-Has(
        hotelName           varchar(20) not null,
        tourID              int not null,
        FOREIGN KEY tourID REFERENCES RegionalTour(tourID),
        FOREIGN KEY hotelName REFERENCES Hotel(hotelName)
)
```

2.7 Cruise Tour

Relational Model:

CruiseTour(<u>tourID</u>, cruise-tour-name)

Functional Dependencies:

tourID -> cruise-tour-name

Candidate Key: {(tourID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CruiseTour(
    tourID                  int not null,
    regional-tour-name      varchar(50) not null,
    PRIMARY KEY(tourID),
    FOREIGN KEY tourID REFERENCES Tour(tourID)
)
```

2.8 Cruise

Relational Model:

Cruise (<u>cruise-name</u>, room-no, cruise-capacity, stars)

Functional Dependencies:

cruise-name -> room-no, cruise-capacity, stars

Candidate Key: {(cruise-name)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Cruise(
    cruise-name         varchar(50) not null,
    room-no             int not null,
    cruise-capacity     int not null,
    stars               int(1) not null,
    PRIMARY KEY(cruise-name)
)
```

2.9 City

Relational Model:

City(cityname, country)

Functional Dependencies:

cityname -> country

Candidate Key: {(cityName)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE City(
        cityName            varchar(20) not null,
        countryName         varchar(20) not null,
        PRIMARY KEY(cityName)
)
```

2.10 SubTour

Relational Model:

Subtour(subtourID, cityName, event-name, description, subtourPrice)

Functional Dependencies:

subtourID,cityName -> event-name, description, subtourPrice

Candidate Key: {(subtourID, cityName)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Subtour(
        event-name          varchar(20) not null,
        description         varchar(300) not null,
        subtourID           int not null,
        cityName            varchar(20) not null,
        subtourPrice        numeric(5,2) not null,
        PRIMARY KEY(subtourID, cityName),
        FOREIGN KEY cityName REFERENCES City(cityName)
)
```

2.11 Dependant

Relational Model:

Dependant(<u>custID, deptID</u>, deptTC, name, birthDate)

Functional Dependencies:

custID, deptID -> name, birthDate, deptTC

Candidate Key: {(deptID,custID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Dependant(
        deptID              int not null auto-increment,
        deptTC                      int not null
        name                varchar(40) not null,
        birthDate           date not null,
        custID              int not null,
        PRIMARY KEY(deptID, custID),
```

```
        FOREIGN KEY custID REFERENCES Customer(custID)
)
```

2.12 Room

Relational Model:

Room(<u>room-no, cruise-name,</u> room-type, price)

Functional Dependencies:

cruise-name, room-no -> room-type, price

Candidate Key: {(cruise-name, room-no)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Room(
        room-no             int(4) not null,
        cruise-name         varchar(20) not null,
        room-type           int(1) not null,
        price               int(8,2) not null,
        PRIMARY KEY(room-no,cruise-name)
)
```

2.13 Promotion

Relational Model:

Promotion(<u>promotionID</u>, promotion-type, sale-rate)

Functional Dependencies:

promotionID -> promotion-type, sale-rate

Candidate Key: {(promotionID)}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE Promotion(
        promotionID                     int not null,
        promotion-type                  int not null,
        sale-rate                       float not null,
        PRIMARY KEY(promotionID)
)
```

2.14 CusMakeRes

Relational Model:

CusMakeRes(<u>custID, reservationID</u>)

Functional Dependencies:

None

Candidate Key: {custID, reservationID}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CusMakeRes(
        custID              int not null,
        reservationID       int not null,
        FOREIGN KEY custID REFERENCES Customer(custID),
```

FOREIGN KEY reservationID **REFERENCES** Reservation(reservationID)

)

2.15 CusCancelRes

Relational Model:

CusCancelRes(<u>custID, reservationID,</u> time)

Functional Dependencies:

None

Candidate Key: {custID, reservationID}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CusCancelRes(
    custID          int not null,
    reservationID   int not null,
    time            date,
    FOREIGN KEY custID REFERENCES Customer(custID),
    FOREIGN KEY reservationID REFERENCES Reservation(reservationID)
)
```

2.16 CusSelectsSub

Relational Model:

CusSelectsSub(<u>custID, reservationID, subtourID,cityName</u>)

Functional Dependencies:

None

Candidate Key: {custID, reservationID, subtourID}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CusSelectsSub(
        custID              int not null,
        reservationID       int not null,
        subtourID           int not null,
        FOREIGN KEY custID REFERENCES Customer(custID),
        FOREIGN KEY reservationID REFERENCES
Reservation(reservationID),
        FOREIGN KEY subtourID REFERENCES Subtour(subtourID)
)
```

2.17 CusChooseRoom

Relational Model:

CusrChooseRoom(<u>custID,room-no,cruise-name</u>)

Functional Dependencies:

None

Candidate Key: {custID,room-no,cruise-name}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CusChooseRoom(
        custID              int not null,
        room-no             int not null,
        cruise-name           varchar(20) not null,
        FOREIGN KEY custID REFERENCES Customer(custID),
        FOREIGN KEY room-no,cruise-name REFERENCES
        Room(room-no,cruise-name)
)
```

2.18 CusRatesTour

Relational Model:

CusRatesTour(<u>custID,tourID,</u> rate)

Functional Dependencies:

None

Candidate Key: {custID,tourID}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CusRatesTour(
        custID              int not null,
        tourID              int not null,
        rate                int not null,
        FOREIGN KEY custID REFERENCES Customer(custID),
        FOREIGN KEY tourID REFERENCES  Tour(tourID)
)
```

2.19 CusBenefitsPro

Relational Model:

CusBenefitsPro(<u>custID,promotionID</u>)

Functional Dependencies:

None

Candidate Key: {custID,<u>promotionID</u>}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE CusBenefitsPro(
    custID          int not null,
    promotionID     int not null,
    FOREIGN KEY custID REFERENCES Customer(custID),
    FOREIGN KEY promotionID REFERENCES  Promotion(promotionID)
    )
```

2.20 ResProvidesPro

Relational Model:

ResProvidesPro(<u>promotionID,reservationID</u>)

Functional Dependencies:

None

Candidate Key: {promotionID,reservationID}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE ResProvidesPro(
        reservationID        int not null,
        promotionID          int not null,
        FOREIGN KEY reservationID REFERENCES
Reservation(reservationID),
        FOREIGN KEY promotionID REFERENCES Promotion(promotionID)
        )
```

2.21 TourVisitCity

Relational Model:

TourVisitsCity(tourID, cityName, transportationtype, date)

Functional Dependencies:

None

Candidate Key: {tourID,cityName}

Normal Form: Boyce-Codd Normal Form

Table Definition:

```
CREATE TABLE TourVisitsCity(
        tourID               int not null,
        cityName                     varchar(20) not null,
        transportationtype   varchar(20) not null,
        date                 date,
```

FOREIGN KEY cityName **REFERENCES** City(cityName),

FOREIGN KEY tourID **REFERENCES** Tour(tourID)

)

2.22 HotelPlacedCity

Relational Model:

HotelPlacedCity(cityName, hotel-name)

Functional Dependencies:

None

Candidate Key: {cityName, hotel-name}

Normal Form: Boyce-Codd Normal Form

Table Definition:

CREATE TABLE HotelPlacedCity(

hotel-name **varchar(20) not null,**

cityName **varchar(20) not null,**

FOREIGN KEY cityName **REFERENCES** City(cityName),

FOREIGN KEY hotel-name **REFERENCES** Hotel(hotel-name)

)

3. Functional Components

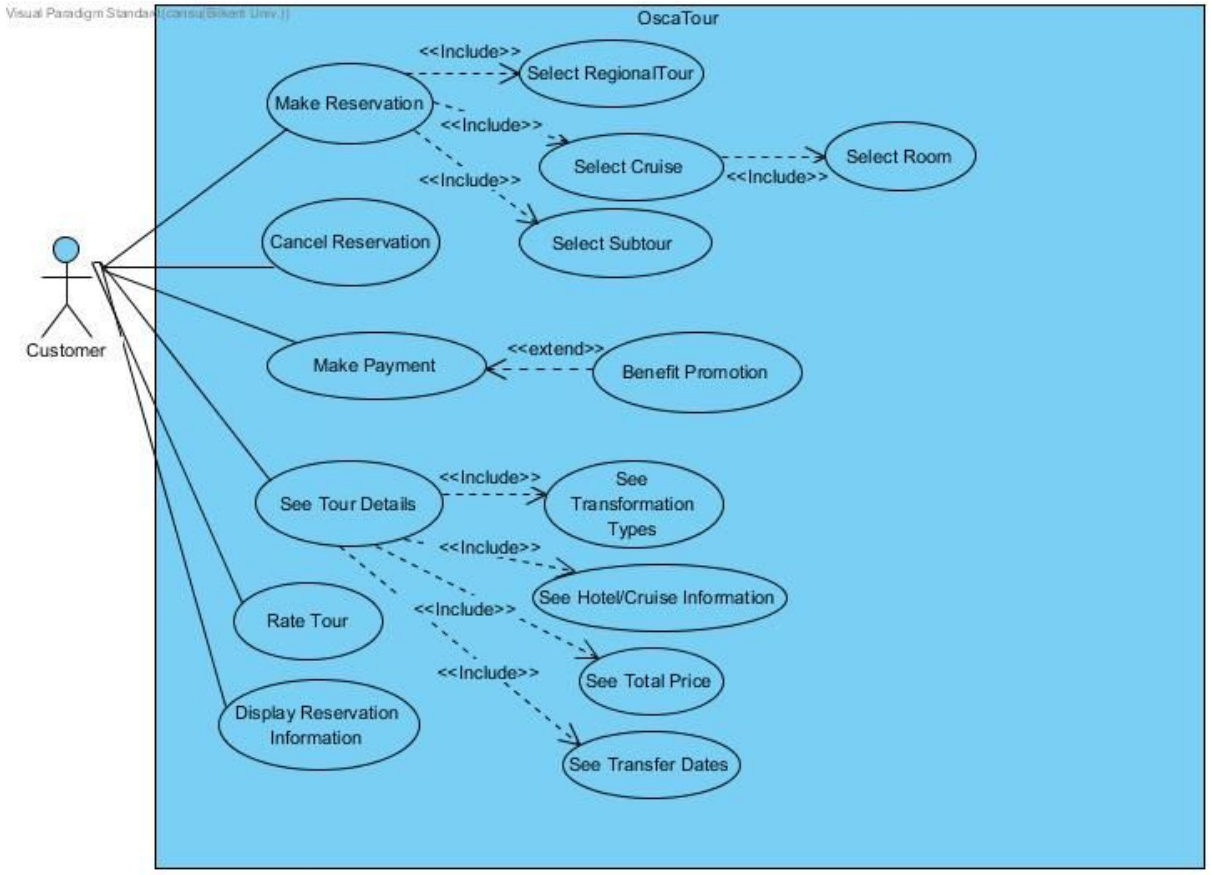
3.1 Use Cases

OscaTour has two different types of users which are Customer and Admin. Use Cases are below.

3.1.1 Customer

All Customers should be able

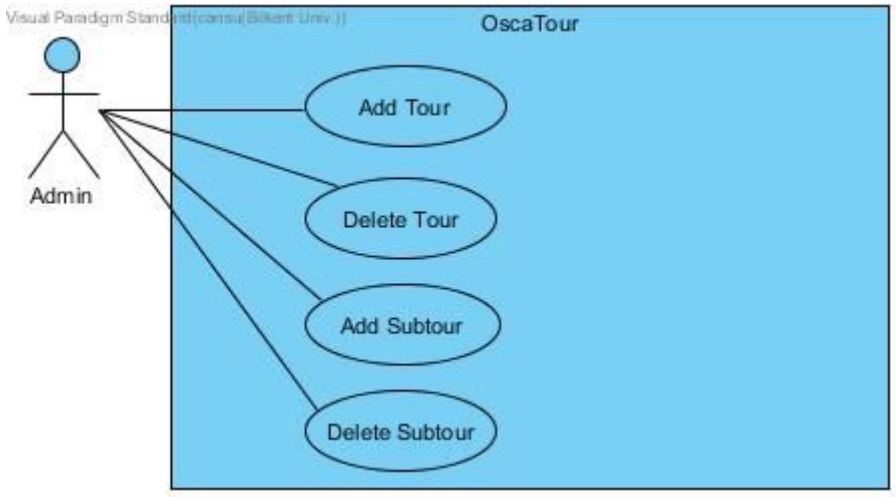
- to make reservations for more than one person
- to cancel their reservations due to a specified day
- to benefit from promotions for their children according to their ages
- to select the tour type which are hotel and cruise
- to select the room types for cruise tours
- to see all the dates for a tour
- to see the hotel/cruise information in detail (number of stars, capacity of the cruise ship etc.)
- to see all the transportation types (plane, bus, train etc.)
- to select the subtours to attend
- to see the total price that will be paid
- to make payment through the system
- to display their reservation information in detail
- to rate the tour that he/she attended



3.1.2 Admin

Admin should be able

- to add/delete tours to the system
- to add/delete subtours to the system



3.2 Algorithms

We will try to use different algorithms to keep track of information and process management for both customer and admin.

3.2.1 Customer Related Algorithm

When customers enter the system, in the home page, they are expected to select a tour type which are hotel and cruise. In cruise tours, customers stay the night in the ship. They can select their room types in the system as well because there are many rooms with different locations and sizes. In hotel tours, customers are not able to select their room types in this platform.

Then the available tours will be listed. Customers can see the details of each tour. Tours will be arranged at least for 2 nights accommodation. In the details of tours, start and end day of the trip, cities that will be visited, transportation types between the cities, detailed hotel information if it is a hotel tour or cruise information otherwise, the number of days of accomodation for each city, the price and city-specific subtours can be seen. Customer must select which subtour(s) s/he will attend on the system before the payment. After all the selections for a tour, payment must be done.

Customers can rate tours in our system. This process can be done during tour time which is between start and end date of tour.

Customer can display his/her reservation with this reservation id number in detail. Also, customers are able to cancel their reservations until a specified day by the company.

3.2.2 Admin Related Algorithm

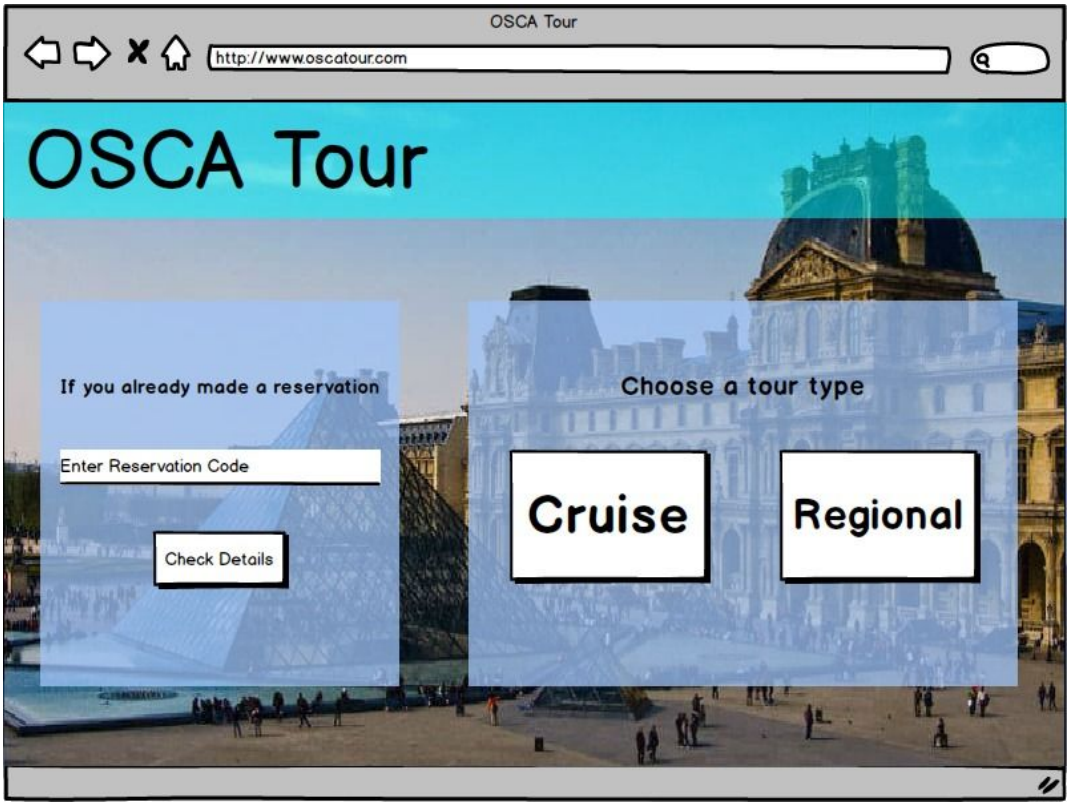
In this system admin's mission is create, delete and edit tours. In our system admin can update rate which is done by customer by using their own experience about a tour. Moreover admin can select and assign subtours for a specified tour.

3.3 Data Structure

We are using built-in data types of SQL such as int, varchar, char, float, boolean, long, date etc.

4. User interface design and corresponding SQL statements

4.1 Home Page



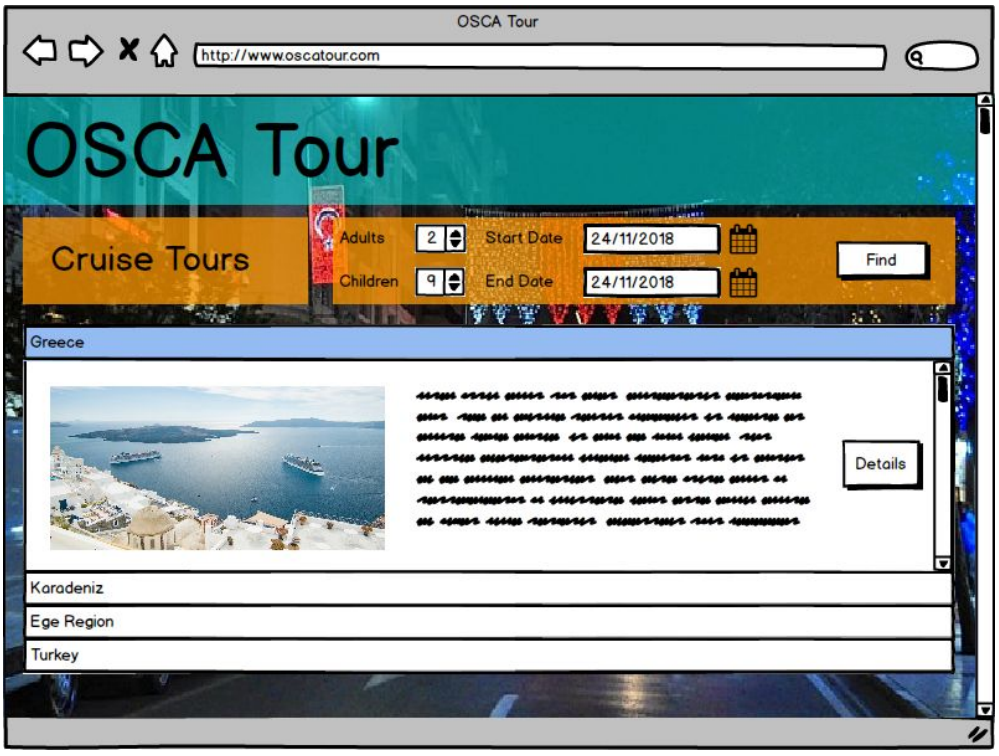
Inputs: @reservationcode

Process: Customers can display their reservation information from the home page with their reservation code. Also, customers can choose the tour type for their new reservations by selecting the relevant button.

SQL Statements:

To display the reservation information, SQL Query is given in the Section 4.9

4.2 Cruise Tours



Inputs: @adults, @children, @startdate, @enddate, @selectedtour

Process: After selecting cruise tour from the home page, all the cruise tours will be displayed. Above the page, there is a filter menu. Customers can filter the results by selecting the adult and children number, start and end date. Customers can see the detailed information of tours by clicking the button next to the brief tour description. @selectedtour will take the id of the selected tour and pass it to section 4.5.

SQL Statements:

Listing all the cruise tours:

SELECT c.cruise-tour-name, t.brief-description

FROM CruiseTour c NATURAL JOIN Tour t

WHERE t.start-date > CURDATE()

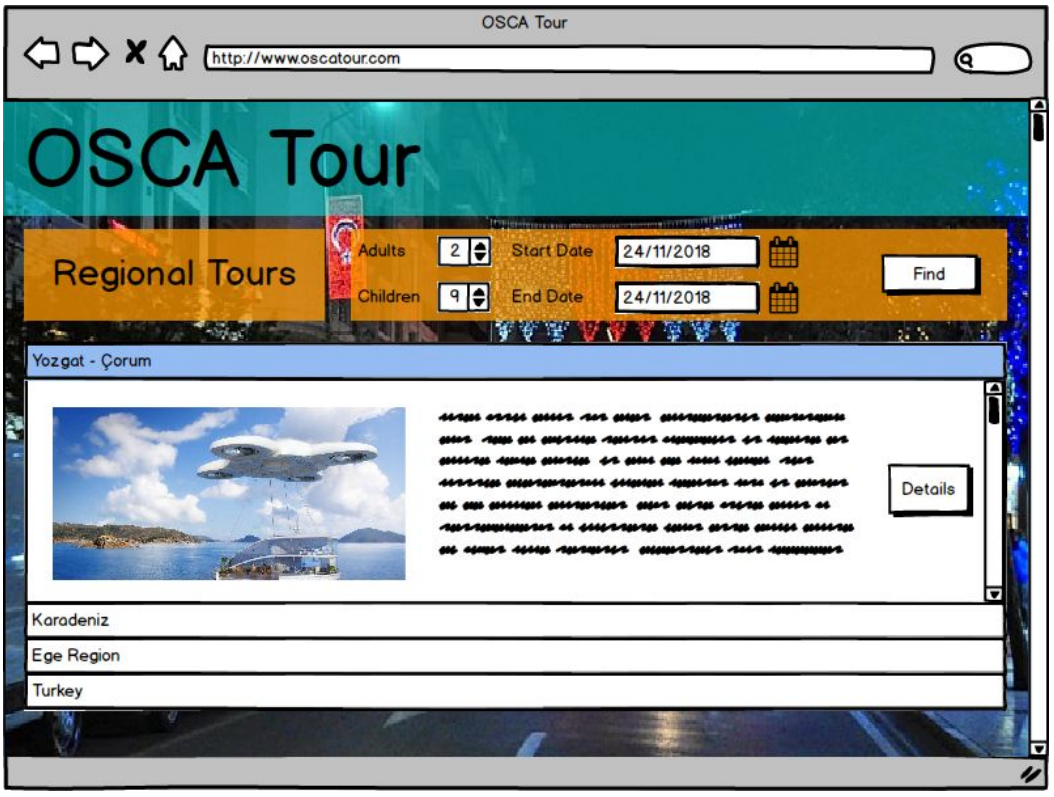
Filtering according to user's inputs:

SELECT c.cruise-tour-name, t.brief-description

FROM CruiseTour c NATURAL JOIN Tour t

WHERE t.start-date > @startdate AND t. finish-date < @enddate AND t.capacity >= (@adults + @children)

4.3 Regional Tours



Inputs:@adults, @children, @startdate, @enddate, @selectedtour

Process: After selecting regional tour from the home page, all the regional tours will be displayed. Above the page, there is a filter menu. Customers can filter the results by selecting the adult and children number, start and end date. Customers can see the detailed information of tours by clicking the button next to the brief tour description.

@selectedtour will take the id of the selected tour and pass it to section 4.5.

SQL Statements:

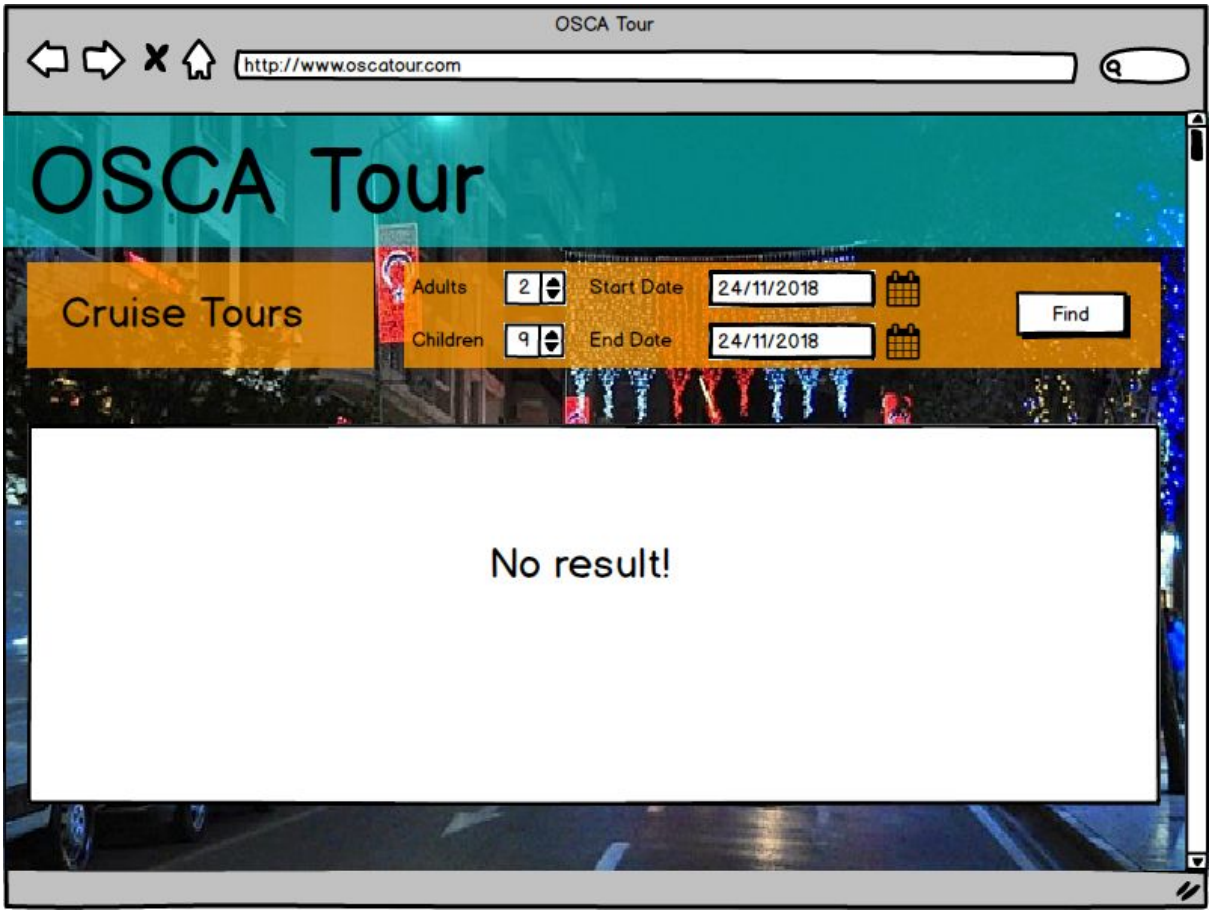
Listing all the regional tours:

```
SELECT r.regional-tour-name, t.brief-description
FROM RegionalTour r NATURAL JOIN Tour t
WHERE t.start-date > CURDATE()
```

Filtering according to user's inputs:

```
SELECT r.regional-tour-name, t.brief-description
FROM RegionalTour r NATURAL JOIN Tour t
WHERE t.start-date > @startdate AND t.finish-date < @enddate AND t.capacity >=
(@adults + @children)
```

4.4 No result Page



Inputs: @adults, @children, @startdate, @enddate

Process: After selecting cruise or regional tour, according to inputs, if there is no available tours, user will see this page.

SQL Statements:

If it is a regional tour:

SELECT r.regional-tour-name, t.brief-description

FROM RegionalTour r NATURAL JOIN Tour t

WHERE t.start-date > @startdate AND t.finish-date < @enddate AND t.capacity >= (@adults + @children)

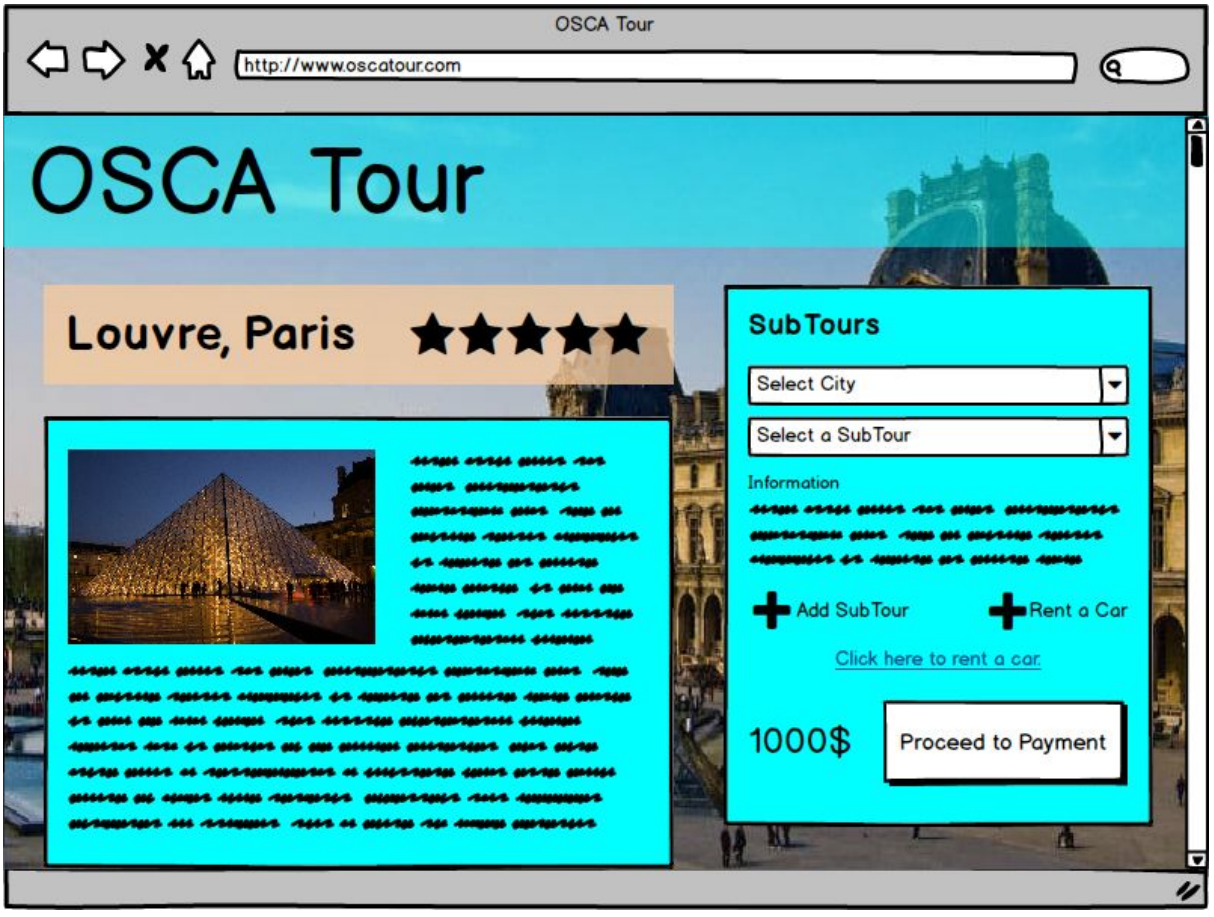
If it is a cruise tour:

SELECT c.cruise-tour-name, t.brief-description

FROM CruiseTour c NATURAL JOIN Tour t

WHERE t.start-date > @startdate AND t. finish-date < @enddate AND t.capacity >= (@adults + @children)

4.5 Tour Information Page



Inputs: @selectcity, @selectsubtour, @selectedtour

Process: When the customer clicked the details button from the previous page (section 4.2 / 4.3), this page will be shown. In this page, customer can see the detailed information about the tour which he/she selected. On the right side of the page, there is a menu that provides information about the subtours. If the customer

wants to attend a subtour, firstly s/he has to select a city and then s/he can add subtour(s) within the listed subtours. The price will be changed according to it. In this screen if user wants to rent a car, he/she must click rent a car button and a link will be popped. Link forwards user to the rent a car website.

SQL Statements:

Listing the description of the selected regional tour:

```
SELECT r.regional-tour-name, t.full-description, t.rating, t.startdate, t.enddate,
t.source, t.baseprice
FROM RegionalTour r NATURAL JOIN Tour t
WHERE t.tourID = @selectedtour
```

Listing the cities according to the selected tour:

```
SELECT city-name
FROM TourVisitsCity
WHERE tourID = @selectedtour
```

Listing subtours according to city:

```
SELECT s.event-name, s.description, s.subtour-price
FROM Subtour s
WHERE s.cityName in(SELECT c.cityName
                    FROM City c
                    WHERE c.cityName=@selectcity)
AND s.cityName in(SELECT tvc.city-name
                  FROM TourVisitsCity tvc
                  WHERE tvc.tourID = @selectedtour)
```

4.6 Subtour Selection

Inputs: @currentCust, @currentRes, @selectcity, @selectsubtour

Process: When the customer selects the city and selects the subtour, the selected subtour and its price will be shown in below. While s/he adds more subtours, all of them will be shown. Also, the price of the tour will be updated according to the number of adults and the selected subtours.
@currentCust is the custID that is given by the system when the user interacted with the system
@currentRes is the reservationID that is given by the system when the user wants to see the details of a tour

SQL Statements:

List all the subtours that customer selects:

```
INSERT INTO CusSelectsSub VALUES( @currentCus, @currentRes,
@selectsubtour, @selectcity)
SELECT s.event-name, s.price
FROM Subtour s
WHERE s.cityName in(SELECT cus.cityName
                    FROM CusSelectsSub cus
                    WHERE cus.cityName = @selectcity AND
                    cus.subtourID = @selectsubtour)
```

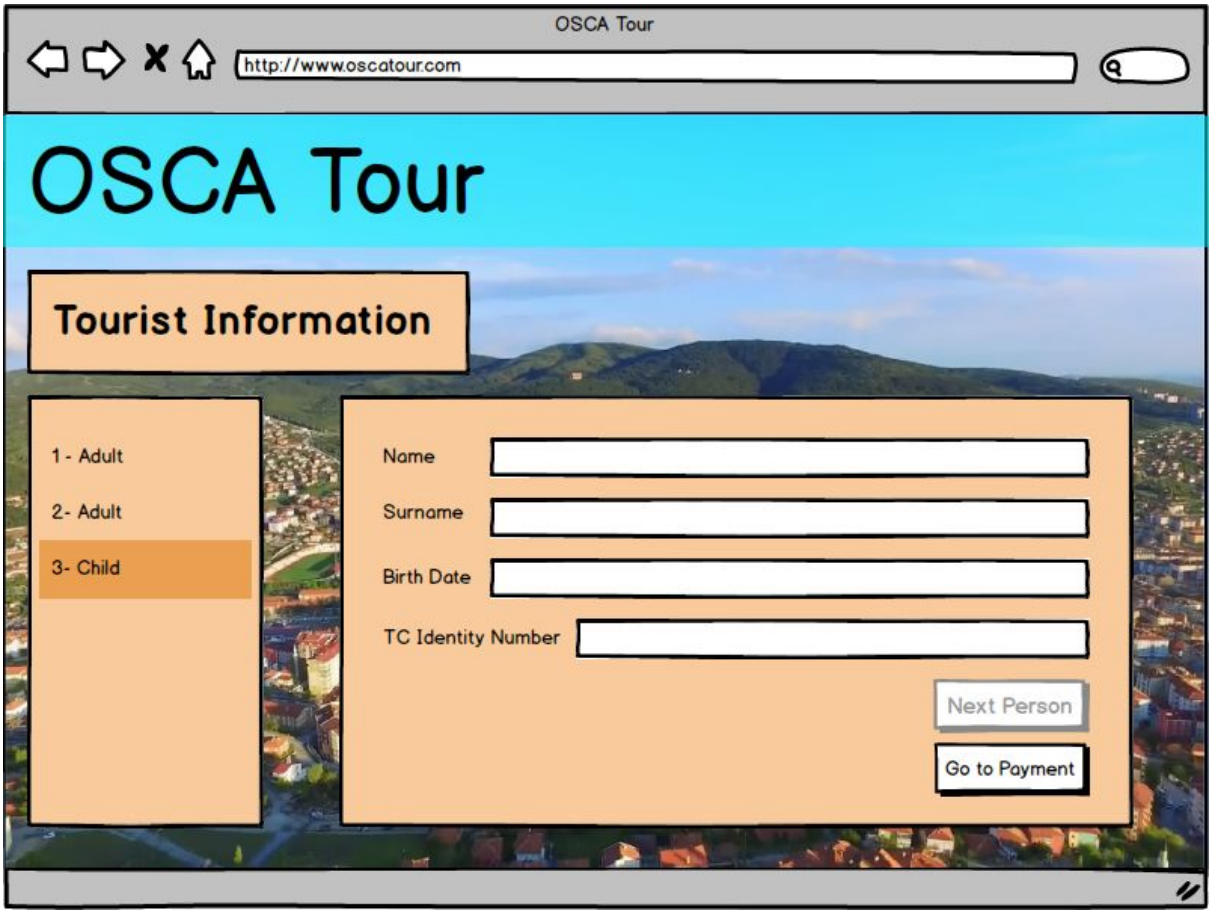
Update the price:

```
UPDATE Reservation
SET Price=Price + (SELECT subtour-price
                   FROM Subtour
                   WHERE subtour-id = @selectsubtour
                   AND subtour-city = @selectcity) *
                  (SELECT COUNT(d.depID)+1
                   FROM Dependant d NATURAL JOIN Customer
                   WHERE custID = @currentCust AND
              (DATEDIFF(CURDATE(),d.birthdate))/365 > 12
                  GROUP BY custID
                 )
              )
WHERE reservationID = @currentRes
```

Show the current price to the customer:

```
SELECT price
FROM Reservation
WHERE reservationID = @currentRes
```

4.7 Tourist Information Page



Inputs: @currentCust, @name1, @surname1, @birthdate1, @TC1, @name2, @surname2, @birthdate2, @TC2,@name3, @surname3, @birthdate3, @TC3

Process: In this page, user enters the needed information about all the attendees in order to reservations to be made by system. In our example, there are two adults and one child. Customer should give the information about first person and click the 'next person' button, and do these for the others. After all the information is given, s/he must make a payment by clicking the 'go to payment' button.

SQL Statements:

Inserting the Customer (The one who will pay for trip / First Person):

INSERT INTO Customer VALUES(@currentCust, @TC1,@name1 + @surname1, 1,”1”, @birthdate1)

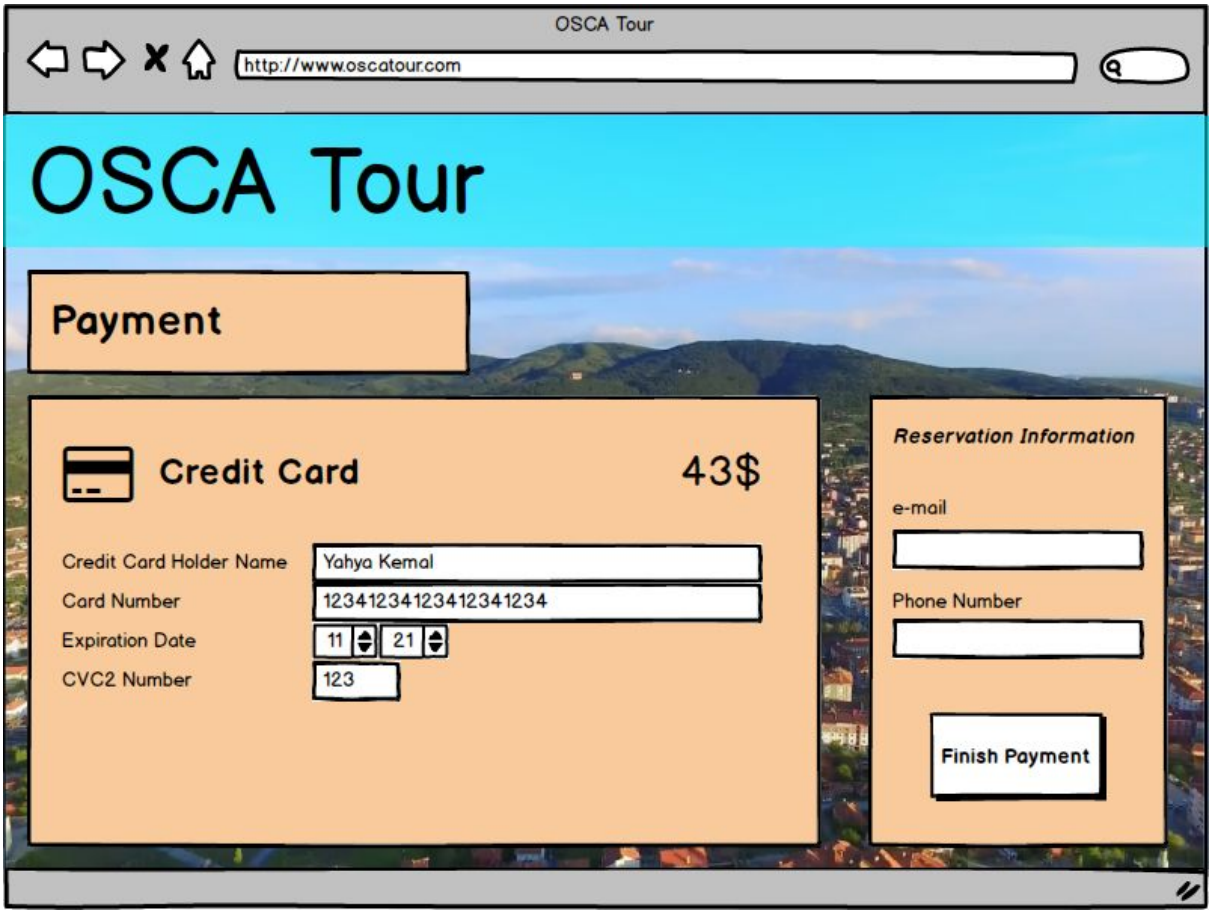
Inserting the Dependant 1 who is not a child:

INSERT INTO Dependant VALUES(@currentCust, @dep2ID, @TC2, @name2 + @surname2, @birthdate2)

Inserting the Dependant 2 who is a child:

INSERT INTO Dependant VALUES(@currentCust, @dep3ID, @TC3, @name3 + @surname3, @birthdate3)

4.8 Payment Page



Inputs: @creditcardholdername, @cardnumber, @expirationdatemonth, @expirationdateyear, @cvc2no, @email, @phone

Process: This is the payment page. User must give the needed information in order to pay the price of the tour. Right side of the page, user must give his/her email and phone number in order to get the reservation code.

SQL Statements:

Update the customer information:

```
UPDATE Customer
SET phone=@phone, mail = @email
WHERE custID = @cusID
```

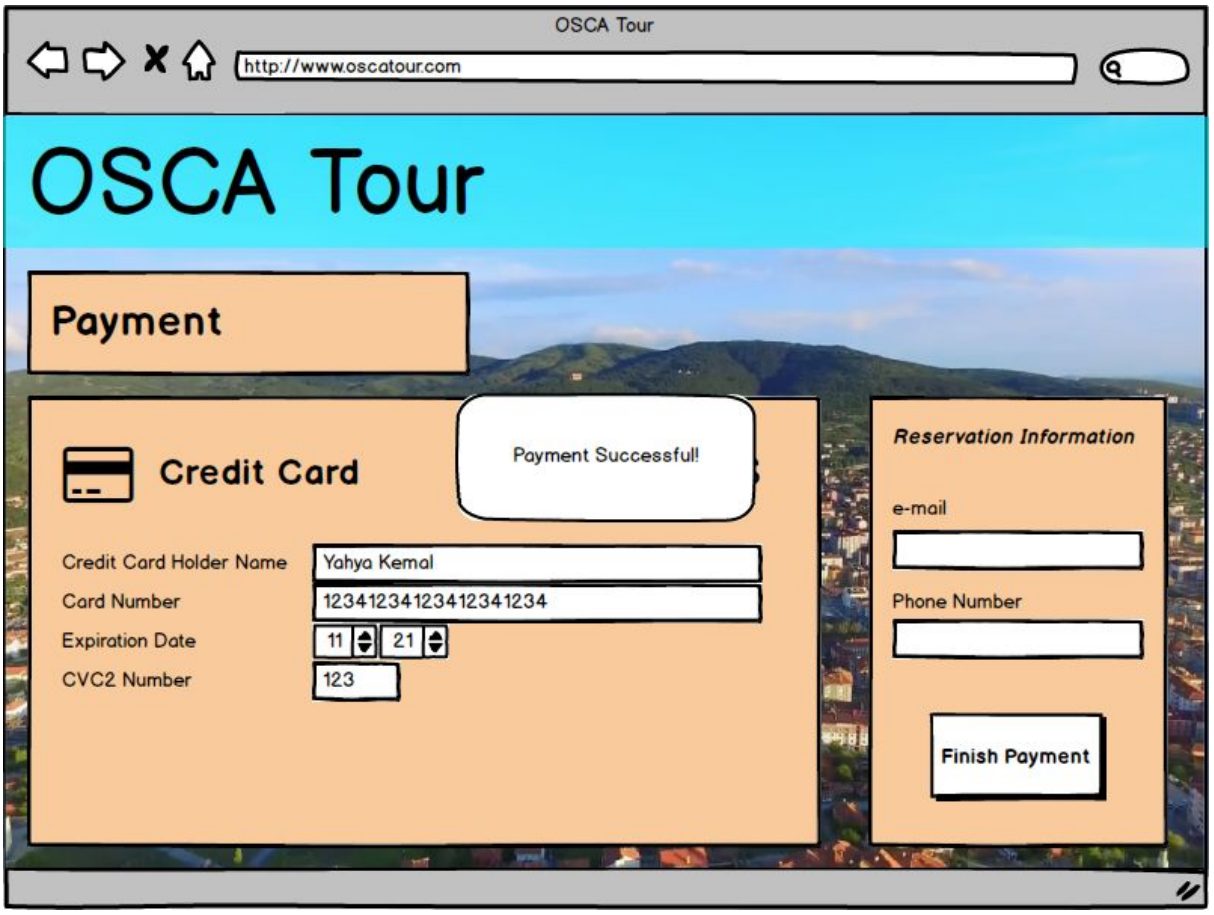
Apply promotion for the customer if there is any promotion:

```
UPDATE Reservation
SET Price = Price - Price * (SELECT promotion-rate
                    FROM Promotion NATURAL JOIN Customer
                    WHERE custID = @custID
                   )
```

Show the total price for the customer to pay:

```
SELECT Price
FROM Reservation
WHERE ReservationID = @ReservationID
```

- Osca Tour does not hold the customer's credit card information for the security priorities.



If user pays and finishes the payment, a message will pop up and says that the payment is successful.

4.9 Reservation Information

Inputs: @reservationID

Process: In the home page, if the customer enters the reservationID code ,this reservation information page will be shown. In here, customers can see the details about their reservation. Customers can cancel their reservations until the one week before of the tour's start date.

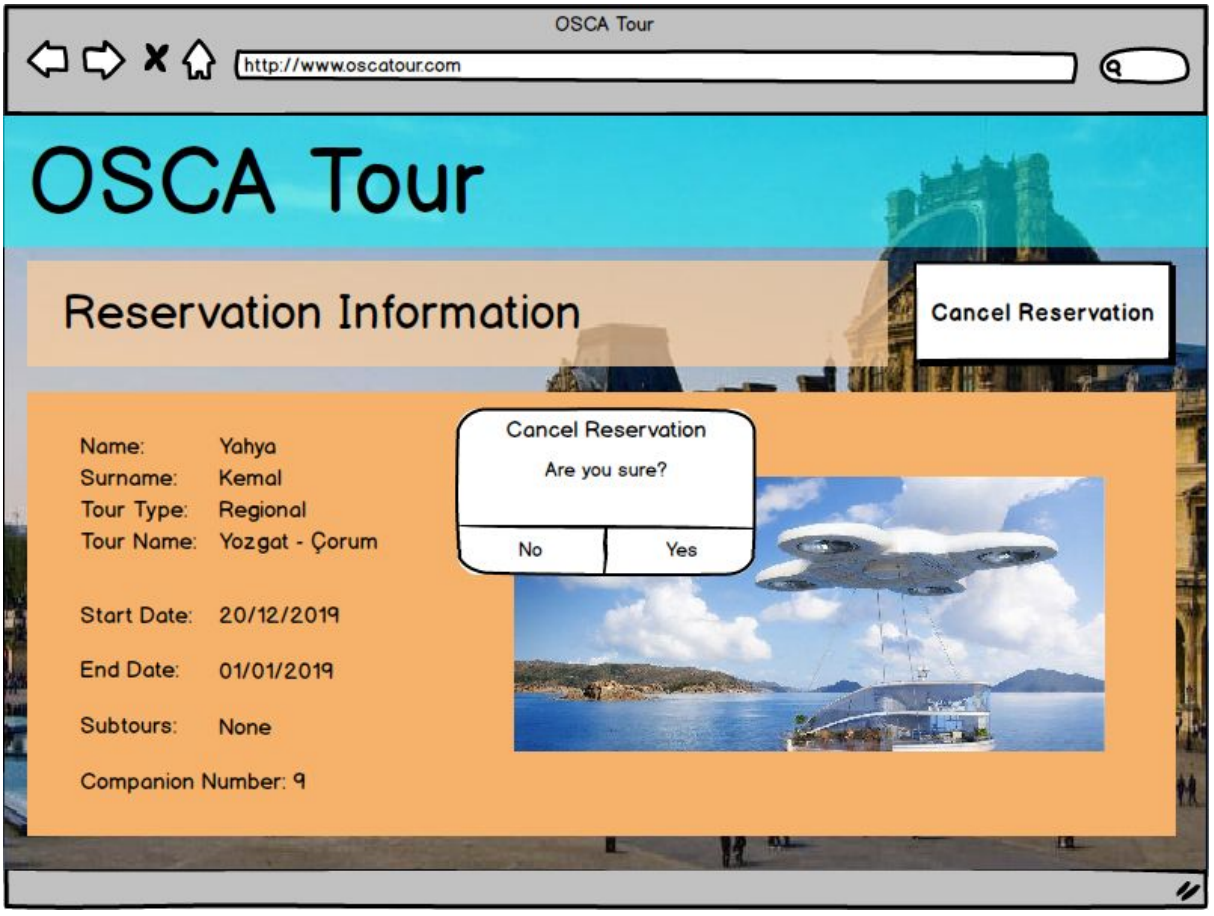
SQL Statements:

SELECT c.name, d.name, t.name, t.fulldescription, r.price, t.source, t.startdate, t.enddate

FROM ((Reservation r natural join Tour t) natural join (Customer c natural join Dependant d))

WHERE r.reservationID = @reservationID

4.10 Cancellation of the reservation



Inputs: @reservationID

Process: If the remaining time for the reservation to start is more than one week, customers are able to cancel their reservations. If the remaining time is proper for the deletion, cancel reservation button become active. When the customer clicks this button, it will ask whether the customer is sure or not and according to the response, transaction will be made.

SQL Statements:

```
DELETE
FROM Reservation
WHERE reservationID = @reservationID AND
reservationID in (SELECT r.reservationID
                  FROM Reservation r NATURAL JOIN Tour
            (DATEDIFF(CURDATE(),t.startdate))>7)
```

If the customer selects the ‘yes’ button for the approval question, “reservation cancelled” pops up.

4.10 Rating the tour

If the reservation date is started, customers can rate their reservation. Rate reservation button becomes active if the date is proper for the rate.

Inputs: @rate

Process: Customers can rate their reservation by giving the stars up to 5.

SQL Statements:

```
INSERT INTO CusRatesTour VALUES(CustID, tourID, @rate)

UPDATE TOUR
SET Rating = (
        SELECT Avg(Rating)
        FROM CusRatesTour NATURAL JOIN Tour
        WHERE tourID in(SELECT tourID
                        FROM Reservation
                        WHERE reservationID = @reservationID))
```

5. Advanced Database Components

5.1 Reports

Average age of people for the tours

```
CREATE VIEW avgAgeForTours AS (
SELECT tourID, Avg(birth-date)
FROM Tour natural join Reservation natural join Customer natural join Dependant
GROUP BY tourID)
```

Number of hotels per city

```
CREATE VIEW avgHotelPerCity AS (
SELECT city-name, Count(hotel-name)
FROM HotelPlacedCity
GROUP BY city-name)
```

5.2 Views

View for displaying the reservation information

```
CREATE VIEW ResInfo AS(
	SELECT c.name, d.name, t.name, t.full-description, r.price, t.source,
	t.startdate, t.enddate
	FROM ((Reservation r natural join Tour t) natural join (Customer c natural join
	Dependant d))
	WHERE r.reservationID = @reservationID
)
```

View for listing the subtours with their prices for a selected city in the selected tour:

```
CREATE VIEW SubtourList AS (
        SELECT s.event-name, s.price
        FROM Subtour s
        WHERE s.cityName in(SELECT cus.cityName
                            FROM CusSelectsSub cus
                            WHERE cus.cityName = @selectcity AND
                            cus.subtourID = @selectsubtour)
```

5.3 Triggers

- Tour Rate will be changed after every rate by a customer
- Tour Capacity will be changed after each reservation
- CustID is given for the current user when s/he first interacts with the system
- ReservationID is created when the customer wants to see detailed information for a tour
- While the customer adds new subtours for a selected tour, price should be updated as well
- If there is a promotion, it must be applied to the total price in the payment page
- System should not require money for the children who are under 12 years old

5.4 Constraints

- Customers cannot cancel their reservations if the remaining time for the tour to start is less than 1 week
- Customers cannot rate their reservation if the tour has not started yet
- Customers cannot select their hotels within a tour.
- Customers cannot select their rooms for hotel tours.
- Customers cannot select transportation types between cities.
- Customers cannot change the arrival days for any city in the tour.

- Customers cannot make reservations without making payment in the last step.
- Customers must be an adult to make a reservation.

5.5 Stored Procedures

- Customers will be messaged and mailed before 1 day of the start date
- Admin will be notified when the stars of a tour is decreased under 3 stars